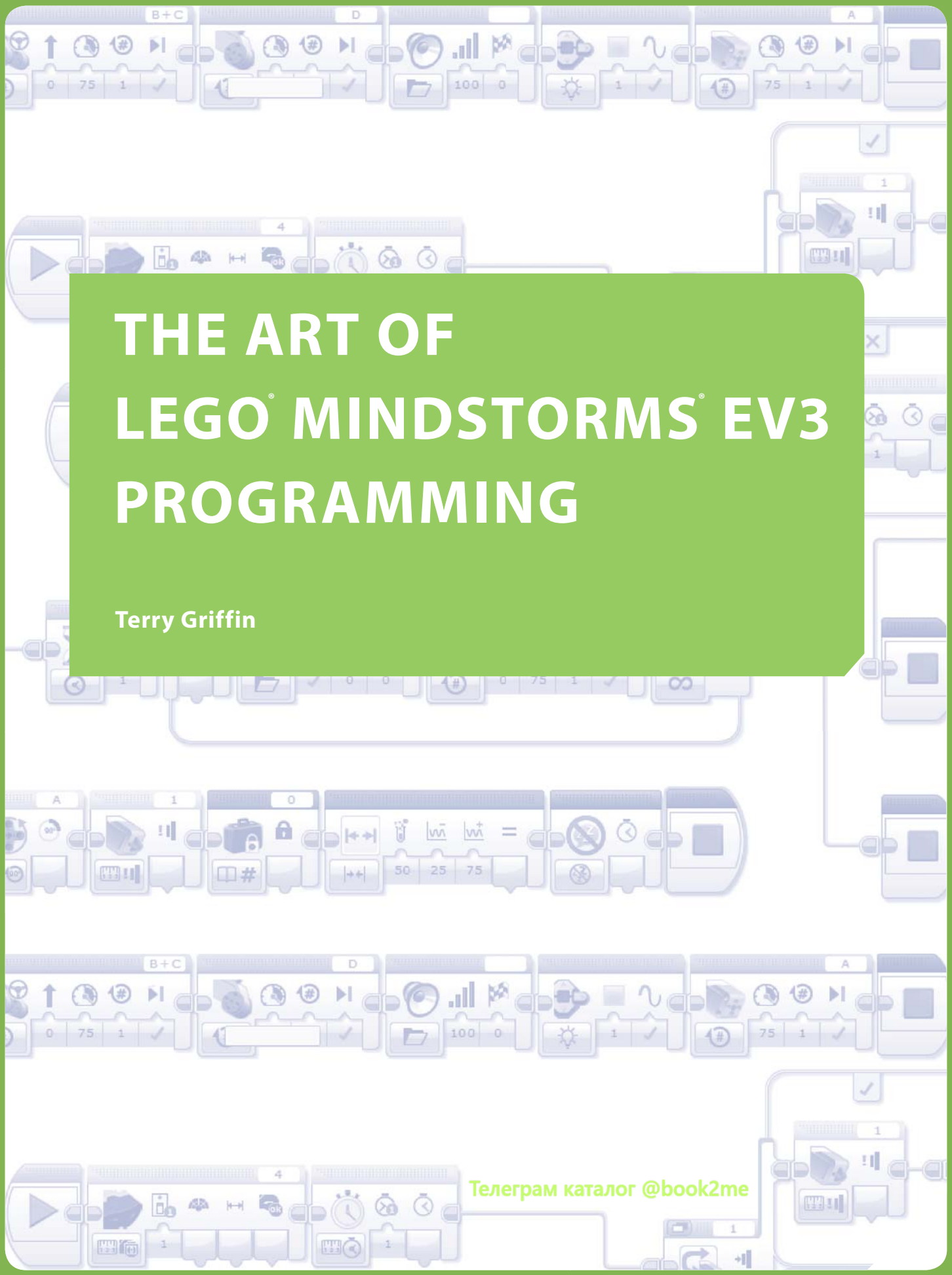
Терри Гриффин

ИСКУССТВО ПРОГРАММИРОВАНИЯ LEGO® MINDSTORMS® EV3

БОМБОРА
ИЗДАТЕЛЬСТВО

no starch press

no starch press



THE ART OF LEGO® MINDSTORMS® EV3 PROGRAMMING

Terry Griffin

ИСКУССТВО ПРОГРАММИРОВАНИЯ LEGO® MINDSTORMS® EV3

Терри Гриффин

БОМБОРА
ИЗДАТЕЛЬСТВО
Москва 2022

УДК 004.4-053.2
ББК 32.973.26-018
Г85

The Art of LEGO MINDSTORMS EV3 Programming
Terry Griffin

Гриффин, Терри.

Г85 Искусство программирования LEGO MINDSTORMS EV3 / Терри Гриффин ; [перевод с английского М. А. Райтмана]. — Москва : Эксмо, 2022. — 272 с. : ил. — (Подарочные издания. Компьютер).

ISBN 978-5-04-095834-4

Открыв книгу, ты поразишься, на что на самом деле способны роботы, собранные из обычных деталей LEGO. Сконструировав своего робота, ты научишь его самым невероятным трюкам и оснастишь искусственным интеллектом с помощью датчиков. Ты узнаешь все тонкости программирования, размещая и настраивая блоки в интерфейсе программы LEGO Mindstorms. Освоишь такие понятия, как операторы, выражения, переменные, циклы, логические значения и многие другие. Эта книга — твой пропуск в мир серьезного программирования и робототехники. Подходит для домашней и образовательной версии программного обеспечения LEGO Mindstorms, а значит, может использоваться как дома, так и в кружках робототехники.

УДК 004.4-053.2
ББК 32.973.26-018

ISBN 978-5-04-095834-4

Посвящается моей семье, благодаря которой вся работа обретает смысл.

И Белле, ради которой я встаю по утрам.

Об авторе

Терри Гриффин уже более 20 лет работает инженером-программистом и создает программное обеспечение для управления различными типами машин. Он получил степень магистра в области компьютерных наук в Университете штата Массачусетс и преподавал программирование школьникам и взрослым. Будучи давним поклонником конструктора LEGO, он написал книгу «*The Art of LEGO MINDSTORMS NXT Programming*», чтобы помочь жене, преподавателю естествознания и математики, применять этих невероятных роботов в своей работе. Терри Гриффин работает в Инновационном центре ионной микроскопии компании Carl Zeiss, где разрабатывает программное обеспечение для управления микроскопами, отслеживающими заряженные частицы.

Благодарности

Я хотел бы поблагодарить членов моей семьи за терпение, которое они проявили, пока я писал эту книгу. Я выражаю особую благодарность своей жене Лиз за бесчисленные часы, потраченные на проверку текста, а также за то, что она мирилась с роботами, заполонившими столовую.

Эта книга не была бы закончена без помощи и поддержки Билла Поллока и сотрудников издательства No Starch Press. Мне было очень приятно работать с Сеф Крамер, Лорел Чун и Дженнифер Гриффит-Дельгадо. Их знания и опыт сыграли важную роль в завершении этого проекта.

Я также хотел бы поблагодарить моих технических рецензентов, Даниэля Бенедеттелли и Роба Торока. Их знание платформы EV3 и робототехники в целом позволило сделать материал актуальным и технически корректным.

О технических рецензентах

Даниэль Бенедеттелли известен во всем мире благодаря своим оригинальным роботам LEGO, среди которых роботы-решатели кубика Рубика и роботы-гуманоиды. Будучи членом группы MINDSTORMS Community Partners (MCP), он участвовал в тестировании и разработке новых продуктов серии MINDSTORMS. Даниэль получил степень магистра робототехники и автоматизации в Сиенском университете в Италии. Он проводит образовательные презентации и семинары по робототехнике и информационно-коммуникационным технологиям по всему миру, преподает робототехнику в средней школе, а также разрабатывает модели LEGO для образовательных программ LEGO. Также он написал книгу «*The LEGO MINDSTORMS EV3 Laboratory*».

Роб Торок — преподаватель из Тасмании, Австралия, который еще в 2001 г. начал использовать роботов LEGO со своими учениками. Он наставляет команды, участвующие в соревнованиях RoboCup Junior и конкурсах FIRST Robotics Competition, а также ведет онлайн-курс по робототехнике SmartBots. В 2010 г. Роб провел шесть месяцев в Образовательном центре по распространению инженерных наук при Университете Тафтса (CEEO) в Бостоне и по сей день продолжает сотрудничество с ним. В настоящее время он совмещает деятельность с работой контент-редактора на сайтах **LEGOengineering.com** и **LEGOeducation.com.au**.

Оглавление

Введение

Книга, которую ты держишь в руках, призвана научить тебя писать программы для роботов LEGO MINDSTORMS EV3. Программное обеспечение EV3 представляет собой мощный инструмент, и, прочитав эту книгу, ты научишься использовать его максимально эффективно, приобретя навыки программирования, необходимые для создания собственных программ.

Для кого предназначена эта книга

Данная книга для тех, кто хочет научиться создавать программы для управления роботом EV3: будь то юный робототехник-энтузиаст, взрослый человек, преподающий робототехнику детям, родитель, тренер FIRST LEGO League, или учитель, использующий роботов EV3 для обучения. Одна из моих целей при написании этой книги заключалась в том, чтобы сделать материал доступным для юных учеников, но при этом достаточно подробно изложить нюансы программирования роботов EV3 для студентов и преподавателей.

Предварительные требования

Эту книгу можно использовать либо с домашней, либо с образовательной версией конструктора — для тестирования написанных программ будет применяться один универсальный робот. Программы, предназначенные для каждой из версий, имеют лишь несколько различий, о которых я в свое время расскажу. Почти вся приведенная в книге информация применима к обеим версиям конструктора.

Никаких предварительных навыков программирования не требуется. EV3 — мощное, но простое в использовании программное обеспечение, которое отлично подходит для начинающих программистов.

Чего ожидать от этой книги

Данная книга посвящена программированию роботов EV3, а не механическим аспектам их сборки. Все описанные программы предназначены для управления одним универсальным роботом или интеллектуальным модулем EV3. Ты научишься работать с основными частями программного обеспечения EV3 — блоками, шинами данных, файлами и переменными, а также узнаешь, как все эти фрагменты взаимодействуют между собой. Кроме того, ты познакомишься с некоторыми хорошими практиками программирования, вредными привычками, которых следует избегать, а также стратегиями отладки, позволяющими получать от процесса программирования больше удовольствия и меньше расстраиваться при возникновении ошибок.

В этой книге ты найдешь пошаговые инструкции и объяснения многих программ EV3, в том числе небольшие примеры, помогающие понять принцип их работы, а также готовые изощренные программы, предназначенные для демонстрации сложного поведения. Тебе также будут предложены задачи по программированию, поощряющие самостоятельное изучение и использование пройденных концепций на практике.

Книга начинается с введения, в котором описывается набор EV3 и программное обеспечение, которое ты будешь использовать для написания собственных программ, и нескольких вводных глав об этой книге и работе с ней. Затем следуют инструкции по сборке тестового робота. Следующие несколько глав посвящены знакомству с основами программного обеспечения EV3, кульминацией которого является программа для прохождения лабиринта, описанная в гл. 7. Далее следуют несколько глав, в которых описаны более сложные функции языка программирования. В конце книги представлена сложная программа движения вдоль линии, предполагающая использование ПИД-регулятора. Ниже приведено краткое содержание каждой главы.

Глава 1. LEGO и роботы: отличная комбинация

Первая глава представляет собой краткое знакомство с программным обеспечением LEGO MINDSTORMS EV3. В ней также рассмотрены некоторые важные различия между домашней и образовательной версиями конструктора и их значение для этой книги.

Глава 2. Среда программирования EV3

Эта глава посвящена обзору возможностей программного обеспечения EV3. На двух простых примерах продемонстрирован процесс написания и запуска программ на модуле EV3. Кроме того, рассмотрены основы изменения параметров блока, добавления комментариев и использования вкладки **Представление порта** (Port View).

Глава 3. TriBot: тестовый робот

Изучив эту главу, ты создашь универсального тестового робота TriBot, которого можно будет использовать для тестирования программ из остальных глав книги.

Глава 4. Движение

В этой главе рассмотрены моторы EV3 и блоки, которые ими управляют. Ты создашь несколько программ, предназначенных для демонстрации типичного использования этих блоков и исследования некоторых распространенных ошибок.

Глава 5. Датчики

Эта глава посвящена датчикам EV3: датчику касания, датчику цвета, ультразвуковому, инфракрасному, гироскопическому датчику и датчику вращения мотора. Ты создашь программу для применения каждого датчика и узнаешь, как использовать вкладку **Представление порта** (Port View) для отслеживания значения датчика при разработке или выполнении программы.

Глава 6. Процесс выполнения программы

В этой главе основное внимание уделено блоку **Переключатель** (Switch), позволяющему программе принимать решения, и блоку **Цикл** (Loop), с помощью которого программа повторяет определенные действия. Ты будешь использовать эти блоки управления операторами для создания простой программы, позволяющей роботу двигаться вдоль линии.

Глава 7. Программа WallFollower: путешествие по лабиринту

После знакомства с основами программирования EV3 можно приступать к решению более сложных задач. В этой главе ты спроектируешь, напишешь и отладишь большую программу, благодаря которой робот сможет найти выход из лабиринта.

Глава 8. Шины данных

Шины данных — одна из наиболее мощных функций программирования EV3. В этой главе объяснено, что такое шины данных, а также описаны способы их эффективного использования. На примерах программ показано, как применять шины данных для получения информации от датчика и как использовать датчик для управления мотором.

Глава 9. Шины данных и блок «Переключатель»

В этой главе рассмотрены продвинутые функции блока **Переключатель** (Switch), которые становятся доступны при использовании шин данных. Ты также научишься применять шины данных для передачи данных в блок **Переключатель** (Switch) и извлечения их из него.

Глава 10. Шины данных и блок «Цикл»

Из этой главы ты узнаешь, как использовать шины данных с блоком **Цикл** (Loop). Ты создашь программу, которая заставляет робота осуществлять поиск по прямоугольной спирали. Для этого ты применишь новые методы, предусматривающие применение счетчиков циклов и условий выхода из цикла.

Глава 11. Переменные

В этой главе рассмотрены блоки **Переменная** (Variable) и **Константа** (Constant). Прочитав ее, ты научишься добавлять и управлять переменными для хранения и обновления значений.

Глава 12. Мои блоки

«Мой блок» — это твой собственный контейнерный блок, созданный в результате объединения других блоков. В этой главе ты научишься создавать контейнеры «Мой блок», использовать их в своих программах и повторно применять в различных проектах.

Глава 13. Математика и логика

В этой главе рассмотрены блоки, имеющие отношение к математике и логике: **Математика** (Math), **Округление** (Round) и **Случайное значение** (Random), **Логические операции** (Logic Operations), **Интервал** (Range). В процессе совершенствования некоторых программ, созданных в предыдущих главах, ты познакомишься с продвинутыми методами использования этих блоков.

Глава 14. Индикаторы, кнопки и экран модуля EV3

Из этой главы ты узнаешь, как использовать блок **Кнопки управления модулем** (Brick Button) для управления программой и как применять блок **Индикатор состояния модуля** (Brick Status Light) для управления цветными индикаторами модуля EV3. Ты также научишься использовать блок **Экран** (Display), создав простую программу для рисования.

Глава 15. Массивы

Эта глава посвящена массивам и способам их использования при программировании EV3. Ты разработаешь программу, позволяющую передать роботу TriBot список команд для выполнения.

Глава 16. Файлы

В этой главе содержится информация, как использовать файлы для хранения информации в модуле EV3, как управлять памятью модуля EV3 и передавать файлы с модуля EV3 на компьютер и обратно. Ты создашь программу, которая использует файл для сохранения и восстановления настроек программы.

Глава 17. Ведение журнала данных

Из этой главы ты узнаешь, как использовать модуль EV3 в качестве регистратора данных. Рассмотрены основы сбора и анализа данных. В результате ты сможешь использовать функцию регистрации данных для лучшего понимания принципа работы блока **Рулевое управление** (Move Steering).

Глава 18. Многозадачность

Модуль EV3 может параллельно выполнять несколько групп блоков. Ты узнаешь о том, как эффективно использовать несколько последовательностей и как избежать некоторых распространенных проблем.

Глава 19. Программа LineFollower с ПИД-регулятором

В заключительной главе рассмотрены продвинутые функции программирования EV3 для создания сложной программы, заставляющей робота двигаться вдоль линии. Ты научишься использовать пропорционально-интегрально-дифференцирующий (ПИД) регулятор для создания робота, быстро и точно двигающегося вдоль линии.

Приложение А. Совместимость платформ NXT и EV3

Из приложения ты узнаешь, как использовать более ранний продукт NXT MINDSTORMS с новым набором EV3.

Приложение Б. Веб-сайты, посвященные набору EV3

В этом приложении ты найдешь список веб-сайтов, содержащих информацию о программировании EV3.

Как лучше всего использовать эту книгу

Для получения максимальной пользы от этой книги по мере ее изучения следует выполнять пошаговые инструкции по созданию примеров программ на своем компьютере. Учиться программировать нужно на практике, написание и проведение экспериментов с программами дадут тебе гораздо больше, чем просто чтение о них.

Программы и сопровождающие их обсуждения будет легче всего понять, если читать главы по порядку. Несколько представленных в первых главах программ будут доработаны в последующих главах, когда ты больше узнаешь о программировании роботов EV3. Прочитав эту книгу, ты получишь знания и навыки, необходимые для того, чтобы стать экспертом по использованию среды программирования EV3.

1

LEGO и роботы: отличная комбинация

Добро пожаловать в мир робототехники! Еще недавно робота можно было встретить только в научно-фантастическом рассказе. Сегодня роботы стали реальностью и выполняют самые разнообразные важные задачи, включая изучение других планет, исследование глубоководных вулканов, сборку автомобилей и проведение хирургических операций. На рис. 1.1 изображен марсоход Curiosity. Теперь в хозяйственном магазине можно приобрести даже робота, который будет мыть полы, пока ты спишь!

Рис. 1.1. Марсоход Curiosity (изображение предоставлено NASA/JPL–Caltech)

LEGO MINDSTORMS EV3

Набор LEGO MINDSTORMS EV3 позволяет создать своего собственного робота. На самом деле ты можешь создать множество роботов. На рис. 1.2 изображен простой робот, с помощью которого ты можешь исследовать свою гостиную.

Рис. 1.2. Робот для исследования гостиной

С набором EV3 очень интересно играть, однако это не просто игрушка. Учителя средних и старших классов используют такие наборы для преподавания науки и техники. У компании LEGO Group есть даже образовательное подразделение LEGO Education, предоставляющее ресурсы учителям, которые применяют продукты LEGO в своей работе.

Студенты со всего мира, участвующие в таких образовательных конкурсах, как FIRST LEGO League (FLL), Всемирная олимпиада роботов (World Robot Olympiad) и RoboCup Junior, используют наборы MINDSTORMS для создания роботов, решающих поставленные задачи.

Набор EV3 поставляется в двух версиях: домашней и образовательной. Домашняя версия набора LEGO MINDSTORMS (31313) продается в магазинах и предназначена для широкой публики. Образовательная версия набора LEGO MINDSTORMS (45544) продается через

дистрибьюторов LEGO Education школам, преподавателям и командам лиги FLL. Эти две версии немного отличаются сочетанием входящих в набор деталей и датчиков LEGO. Кроме того, образовательная версия программного обеспечения содержит некоторые дополнительные функции для проведения научных экспериментов с помощью модуля EV3. При изучении этой книги различие в наборе деталей не проблема: ты можешь создать робота, используя домашнюю или образовательную версию конструктора. Роботы, собранные из этих двух наборов, будут несколько отличаться друг от друга (например, размером колес), однако эти различия не будут иметь большого значения.

Набор EV3 относится к третьему поколению конструкторов LEGO MINDSTORMS. Многие компоненты конструкторов предыдущего поколения NXT можно использовать с наборами EV3. Для получения дополнительной информации обратись к приложению А.

Набор LEGO MINDSTORMS EV3

Набор EV3 включает в себя интеллектуальный модуль EV3, три мотора, несколько датчиков, инструкции по загрузке программного обеспечения MINDSTORMS EV3 и детали LEGO для сборки робота. Как было отмечено в предыдущем разделе, конкретное сочетание деталей и датчиков зависит от используемой версии набора.

К строительным деталям относятся зубчатые колеса, оси, штифты и балки линейки LEGO TECHNIC (рис. 1.3). Они идеально подходят для создания роботов, потому что из этих прочных и легких компонентов можно собрать сложные подвижные части. К тому же для улучшения своих роботизированных творений ты можешь легко добавить свои собственные детали из других наборов, например TECHNIC, BIONICLE и даже традиционных конструкторов LEGO.

Рис. 1.3. Балки и штифты

Интеллектуальный модуль EV3 (программируемый блок или просто EV3) — это мозг робота. По сути, модуль EV3 представляет собой небольшой компьютер, который ты будешь программировать, чтобы научить свои творения двигаться. Вместо полноразмерного монитора и клавиатуры он имеет небольшой экран, набор кнопок и порты для подключения моторов и датчиков. Запрограммировать модуль EV3 можно напрямую, используя его собственные возможности, а можно создать программу с помощью программного обеспечения EV3 для операционной системы Windows или macOS, а затем загрузить ее в модуль. После запуска программы модуль EV3 собирает данные с датчиков и управляет моторами в соответствии с инструкциями, указанными в программе.

С помощью моторов EV3 обычную модель LEGO можно превратить в движущегося робота. Два больших мотора позволяют легко создавать мобильных колесных или гусеничных роботов. Кроме того, с помощью больших моторов или среднего мотора можно создавать роботизированные руки, краны, катапульты и другие приспособления. Многим роботам два больших мотора требуются для перемещения, а третий — для выполнения другой функции. Однако некоторые роботы используют все три мотора для выполнения различных задач, поэтому совсем не перемещаются.

Датчики EV3 позволяют роботу реагировать на окружающую среду в соответствии с данными ему командами. Среди датчиков EV3 есть ультразвуковой, инфракрасный, датчик касания, датчик цвета, гироскопический датчик и датчик вращения мотора. Последний встроен в каждый мотор EV3; остальные датчики представляют собой отдельные детали. Функции датчиков следующие:

Ультразвуковой датчик. Измеряет расстояние до объекта или препятствия. Может обнаруживать присутствие другого ультразвукового датчика.
Инфракрасный датчик. Измеряет расстояние до объекта или препятствия. Также может определить расстояние до удаленного инфракрасного маяка, направление, в котором он находится, и нажатые на нем кнопки.
Датчик касания. Обнаруживает нажатие кнопки, расположенной на его передней части. Позволяет роботу «ощутить» столкновение с объектом или прикосновение объекта к нему.
Датчик цвета. Определяет цвет объектов и измеряет яркость направленного на него света. Кроме того, этот датчик может измерять яркость отраженного света и яркость внешнего освещения.
Гироскопический датчик. Измеряет вращательное движение робота, а также скорость вращения и угол его наклона.
Датчик вращения мотора. Определяет количество совершенных мотором оборотов. Каждый мотор EV3 содержит встроенный датчик вращения.

Домашняя версия конструктора включает датчик касания, датчик цвета, инфракрасный датчик и удаленный инфракрасный маяк. В образовательную версию входят два датчика касания, датчик цвета, ультразвуковой и гироскопический датчик. Это означает, что у тебя будет либо ультразвуковой, либо инфракрасный датчик. Два этих датчика можно использовать для измерения расстояния от робота до объекта.

В большинстве приведенных в книге программ ультразвуковой и инфракрасный датчики будут взаимозаменяемы.

Компания LEGO Group также производит датчик температуры (продается отдельно), а другие компании создают дополнительные датчики для наборов EV3. Например, среди продуктов компаний HiTechnic, Vernier, Dexter Industries и Mindsensors можно найти датчик-компас, датчик ускорения и барометрический датчик.

Программное обеспечение LEGO MINDSTORMS EV3

Программное обеспечение EV3 представляет собой графическую среду программирования, в которой содержатся все инструменты, необходимые для написания программы для робота EV3. Приложение такого типа часто называют *интегрированной средой разработки* или *ИСР* (integrated development environment, IDE). ИСР EV3 считается *графической* средой программирования, поскольку для создания программы используются цветные значки — *блоки*. Существуют блоки для управления моторами, использования датчиков и выполнения многих других действий. Ты будешь создавать программу, перетаскивая блоки по экрану, соединяя их между собой и изменяя их параметры.

Программное обеспечение EV3 отличается замечательным балансом между простотой применения и мощностью. С его помощью можно легко написать не только простые, но и весьма сложные программы. Поначалу некоторые продвинутые функции могут показаться трудными для понимания, однако после небольшой практики они станут более понятными.

Программное обеспечение, прошивка и аппаратные средства

Программа является одним из трех компонентов, совместная работа которых позволяет управлять роботом. Создаваемая тобой программа называется *программным обеспечением*, которое представляет собой набор инструкций, выполняемых компьютером. В данном случае компьютером является интеллектуальный модуль EV3. Программное обеспечение отличается тем, что в него легко внести изменения. Эта гибкость позволяет создавать разнообразные программы, используя только модуль EV3, три мотора и несколько датчиков.

Организация блоков на виртуальном холсте — это способ создания программ, удобный для людей, однако для выполнения программы модулю EV3 необходимо кое-что еще. Созданная тобой программа представляет собой *исходный код*, который нужно преобразовать в набор инструкций для модуля EV3. Затем эти инструкции должны быть перенесены из компьютера в модуль. После преобразования и загрузки программу можно будет запустить.

Программа, работающая непосредственно на модуле EV3, представляет собой *прошивку*, которая редко изменяется и фактически является частью устройства. Прошивка EV3 функционирует так же, как операционная система компьютера или смартфона, например Windows, iOS, Linux или Android. Прошивка — это программа, которая издает звук при включении модуля, управляет экраном и реагирует на нажатие кнопок на модуле EV3. При подключении EV3 к компьютеру, среда MINDSTORMS взаимодействует с прошивкой этого модуля.

ПРИМЕЧАНИЕ **Компания LEGO периодически выпускает обновления прошивки EV3 для добавления новых функций или устранения проблем. Если твой компьютер подключен к Интернету, приложение MINDSTORMS проверит наличие обновления и предложит его загрузить.**

Модуль EV3 — это аппаратное обеспечение, на котором работает твоя программа. *Аппаратное обеспечение* включает в себя физические компоненты компьютера: модуль, моторы, датчики и строительные детали LEGO. Аппаратное обеспечение не изменяется; ты можешь по-разному организовать и использовать эти компоненты, однако возможности каждого из них остаются прежними.

Искусство и инженерия

Для меня самым захватывающим этапом процесса создания робота является написание программы, позволяющей вдохнуть в него жизнь. Компьютерное программирование представляет собой сочетание искусства и инженерии. Мы используем принципы *инженерии*, когда совершаем ряд логических шагов для решения практической задачи. По мере изучения этой книги, особенно наиболее объемных программ, представленных ближе к концу, ты освоишь технические принципы и методы программирования, которые помогут тебе создавать качественные программы (и избежать формирования некоторых распространенных вредных привычек). Однако основной процесс написания программы для решения конкретной задачи часто является

скорее искусством, чем инженерным делом. Программы не всегда создаются поэтапно, и часто их написание требует изрядной доли творчества и изобретательности. На мой взгляд, именно использование творческого подхода делает программирование таким интересным занятием.

Однако в процессе программирования также можно разочароваться, когда что-то идет не так, как предполагалось. При возникновении сбоя в программе бывает сложно найти его причину. В этой книге я покажу тебе, как находить и исправлять ошибки в твоих программах. Просто помни о том, что раскрытие тайны — это весело!

Качества хорошей программы

Многие решения, принятые при создании программ, будут зависеть от твоего индивидуального вкуса, более того, со временем ты выработаешь свой собственный стиль программирования. Практически всегда существует несколько правильных способов решения проблемы. Тем не менее есть три правила, которые можно использовать для оценки качества программы. Программа должна делать следующее:

1. Выполнять нужную функцию.
2. Быть легко изменяемой.
3. Быть понятной тем, кто знает язык программирования, используемый для ее создания.

Первое правило кажется довольно очевидным, однако в нем есть еще кое-что. Прежде чем ты сможешь убедиться в работоспособности программы, сначала тебе нужно определиться с ее *требованиями*, т. е. с полным описанием того, что эта программа должна делать. Если ты создаешь программу для школьного проекта или конкурса FLL, то можешь ознакомиться с этими требованиями перед началом работы. Если же ты создаешь робота просто ради интереса, — можешь составить список требований по мере реализации своего проекта. В любом случае тебе нужно определиться с назначением своего робота, прежде чем судить о его работоспособности.

Второе правило существует потому, что после начала работы над программой требования часто меняются. Возможно, ты обнаружишь, что не можешь решить задачу так, как запланировал изначально, или решишь расширить список требований для решения более сложной задачи. Хорошо, если ты сможешь легко изменить свою программу, чтобы адаптировать ее к новым требованиям. Программу, которую легко модифицировать, с большей вероятностью можно будет использовать для решения похожих задач. Повторное применение существующих программ вместо создания новых может сэкономить много времени.

Третье правило заключается в том, чтобы сделать программу максимально простой и понятной. Излишне сложные программы содержат больше ошибок, их труднее использовать повторно. Чтобы сделать программу более понятной, можно добавить в нее комментарии, поясняющие принцип ее работы. Уместные комментарии — это простой способ сделать программу полезной для других программистов.

Что ты узнаешь из этой книги

Для того чтобы стать успешным программистом, необходимы знание и практика. В этой книге освещены три области знаний, необходимых успешному программисту EV3:

Поведение каждого из блоков. Изучение того, как работает каждый из блоков, — это первый шаг к их использованию в программе. Несмотря на множество блоков, каждый из которых имеет несколько параметров, изучить их не сложно. Файл справки EV3 содержит подробное описание каждого блока, а создание небольших тестовых программ, позволяющих разобраться в их функционировании, — это довольно простое (и интересное) занятие.
Объединение нескольких блоков в рабочую программу. Для этого тебе нужно будет узнать о ходе выполнения программы, шинах данных и переменных. Вероятно, именно здесь ты столкнешься с определенными сложностями. Изучение некоторых нюансов, касающихся работы программы EV3, поможет тебе избежать путаницы, с которой сталкиваются многие пользователи, когда они переходят от простых программ к более сложным.
Общие практики программирования. Три приведенных выше правила — это только начало. Читая эту книгу, ты узнаешь и о других концепциях, которые могут быть полезны вне зависимости от используемого языка программирования или типа создаваемой программы.

Программирование — один из тех видов деятельности, освоение которых немыслимо без практики. Многие из концепций, с которыми тебе предстоит разобраться, обретут смысл только тогда, когда ты увидишь их в действии. Чем больше программ ты напишешь, тем комфортнее тебе будет с ними работать.

Онлайн-сообщество LEGO MINDSTORMS

В Интернете существует процветающее сообщество энтузиастов LEGO, объединяющее сайты, на которых представлены сотни инновационных конструкций роботов. Например, сайт mindBOARDS (**www.mindboards.net**) хорошо известен своими форумами, на которых пользователи могут обмениваться идеями и находить ответы на вопросы. К этим ресурсам можно обратиться, когда ты не понимаешь, почему твой робот работает не так, как надо. Быстрый поиск по форумам часто позволяет найти ответ. При отсутствии уже готового решения ты можешь задать вопрос, описывающий твою конкретную проблему. Список полезных сайтов, имеющих отношение к конструктору MINDSTORMS, приведен в приложении Б.

Что дальше?

Из следующей главы ты узнаешь о среде программирования EV3 и некоторых базовых концепциях программирования. На простых примерах программ увидишь процесс использования этой среды. В последующих главах ты познакомишься с другими блоками и концепциями программирования на примере все более сложных программ.

Исходный код для всех приведенных в книге программ можно загрузить по ссылке:

http://addons.eksmo.ru/it/Lego_mindstorms_EV3.zip

2

Среда программирования EV3

В этой главе описана среда программирования EV3 и представлено несколько простых программ. Мы начнем с основ, рассмотрев примеры программ, в которых используется только интеллектуальный модуль EV3 без моторов и датчиков. О программировании датчиков ты узнаешь из главы 4, о применении датчиков — из главы 5.

Экскурсия по программному обеспечению MINDSTORMS

При запуске программы MINDSTORMS EV3 появится начальный экран — *лобби*. Домашняя и образовательная версии предусматривают разные начальные экраны, однако функционируют они одинаково. На лобби ты можешь создать или открыть проект, получить доступ к руководству пользователя и файлу справки, а также прочитать инструкции по созданию примеров роботов. Лобби для домашней версии показано на рис. 2.1, а для образовательной версии — на рис. 2.2.

Прежде чем приступить к написанию первой программы, рассмотрим основные области среды MINDSTORMS EV3. Для создания нового проекта выбери команду меню **Файл** (File) ⇒ **Новый проект** (New Project) (**Файл** (File) ⇒ **Новый проект** (New Project) ⇒ **Программа** (Program) в образовательной версии). Появится экран, показанный на рис. 2.3.

ПРИМЕЧАНИЕ В этой книге использованы изображения из домашней версии среды MINDSTORMS EV3, работающей в операционной системе Windows 10.

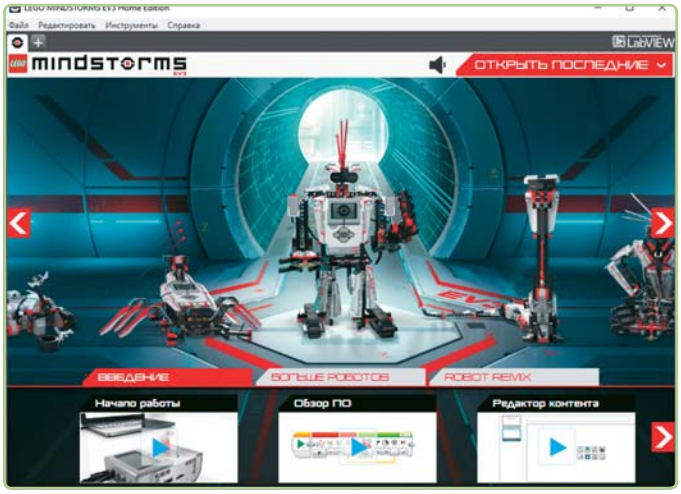


Рис. 2.1. Лобби для домашней версии программного обеспечения EV3

Рис. 2.2. Лобби для образовательной версии программного обеспечения EV3

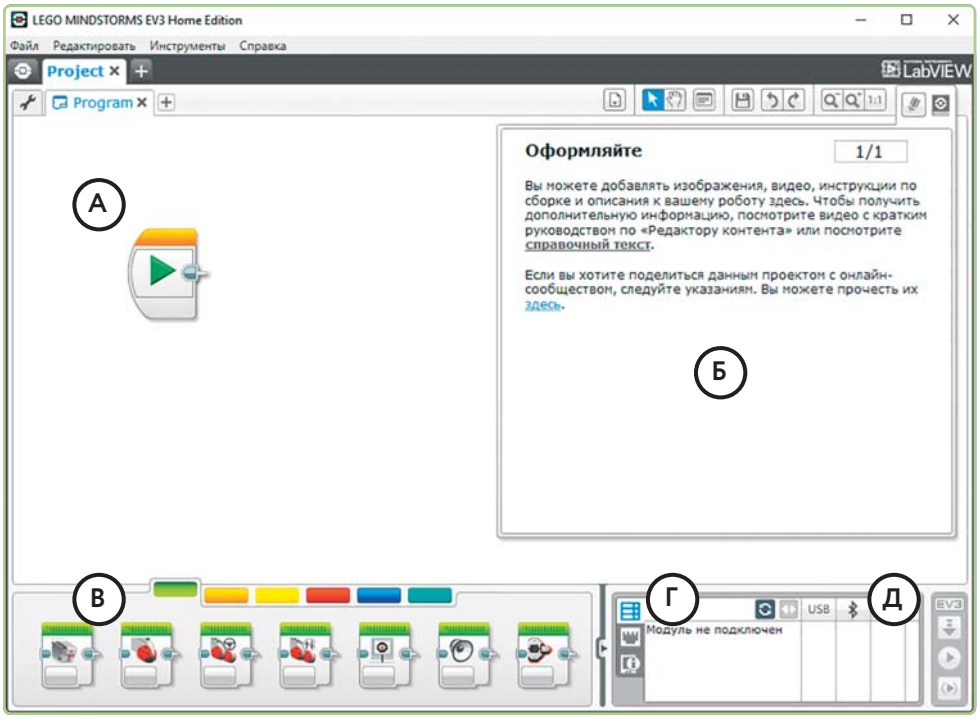


Рис. 2.3. Среда программирования MINDSTORMS EV3

А: Область программирования

Большую часть экрана занимает *область программирования*, в которой ты будешь создавать свою программу. Для переключения между открытыми проектами можно использовать вкладки в верхней части окна. На рис. 2.3 показан один открытый проект под названием **Project**. Небольшой значок MINDSTORMS слева от вкладки **Project** позволяет вернуться на лобби.

Один проект может содержать несколько программ. Под вкладкой с названием проекта расположена другая группа вкладок, которые можно использовать для выбора программы. На рис. 2.3 показана одна открытая программа под названием **Program**. О переименовании программ и проектов поговорим далее. Щелчок по значку в виде гаечного ключа открывает страницу **Свойства проекта** (Project Properties), которая также будет описана далее.

Б: Редактор контента

Редактор контента позволяет документировать работу над проектом, например создать презентацию, включающую текст, изображения и видео. Ты можешь добавить описание работы программы, инструкции по сборке робота или видео, демонстрирующее робота в действии. Презентация сохраняется вместе с проектом, поэтому тебе не нужно сохранять ее отдельным файлом.

Когда в использовании редактора контента нет необходимости, его можно закрыть, щелкнув по небольшому значку MINDSTORMS в его верхнем правом углу. Это освободит место в области программирования для работы над программой.

В: Палитра программирования

Палитра программирования расположена в нижней части области программирования и содержит блоки, использующиеся для создания программ. Для облегчения работы блоки разделены на шесть категорий и маркированы цветом. Для переключения между ними можно использовать цветные вкладки. Слева направо располагаются шесть групп блоков: *блоки действий* (зеленые), *блоки управления операторами* (оранжевые), *блоки датчиков* (желтые), *блоки операций с данными* (красные), *блоки дополнений* (темно-синие) и пользовательские блоки (контейнеры) — *мои блоки* (светло-голубые).

Г: Страница аппаратных средств

На *странице аппаратных средств* отображается информация о модуле EV3. Она представлена в трех разделах, между которыми можно переключаться с помощью кнопок слева (рис. 2.4). Верхняя вкладка, **Информация о модуле** (Brick Information), содержит информацию о состоянии модуля, уровне заряда аккумулятора, версии прошивки и объеме используемой памяти. Средняя вкладка **Представление порта** (Port View) содержит информацию о подключенных к модулю датчиках и моторах, а нижняя вкладка **Доступные модули** (Available Bricks) — обо всех подключенных к программному обеспечению EV3 модулях.

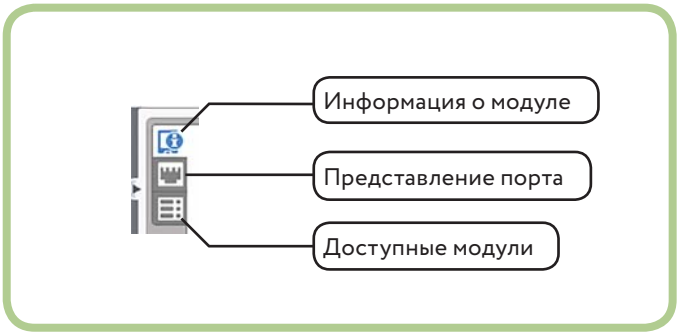


Рис. 2.4. Вкладки страницы аппаратных средств

Д: Кнопки загрузки и запуска программы

Кнопки загрузки и запуска (рис. 2.5) позволяют перенести программу с компьютера в память модуля EV3 и запустить ее. Процесс переноса программы в память модуля называется *загрузкой*.

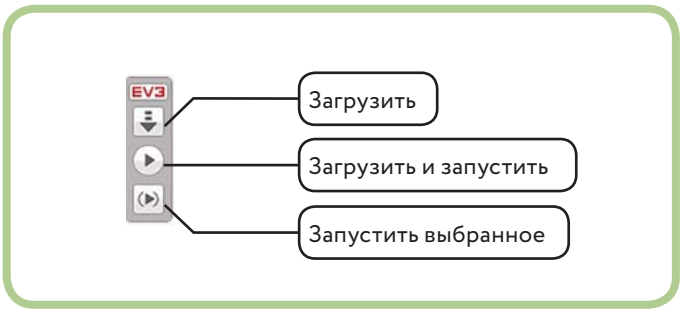


Рис. 2.5. Кнопки загрузки и запуска программ

Средняя кнопка **Загрузить и запустить** (Download and Run) позволяет загрузить и сразу запустить программу. При нажатии верхней кнопки **Загрузить** (Download) программа переносится в память модуля, но не запускается. После загрузки программу можно запустить, используя кнопки на самом модуле. Это удобно, если требуется переместить или отсоединить кабель робота перед запуском программы. Нижняя кнопка **Запустить выбранное** (Run Selected) позволяет загрузить и запустить только выбранные блоки, что упрощает процесс поиска и устранения ошибок в программе.

Написание программы для модуля EV3

При создании нового проекта область программирования уже содержит блок **Начало** (Start) (рис. 2.6). Для создания программы перетаскивай блоки с палитры программирования на область программирования, начиная с блока **Начало** (Start), и соединяй их в линию. Каждый блок имеет несколько параметров, от значений которых зависит его поведение. При запуске программы модуль EV3 выполняет каждый блок по порядку слева направо. Обычно блоки выполняются по одному за раз, т. е. каждый блок должен завершить свою работу до того, как запустится следующий. Однако, как будет показано далее, из этого правила есть некоторые исключения, когда несколько блоков могут выполняться одновременно. Программа завершается после выполнения последнего блока.

*Рис. 2.6. Блок **Начало** (Start)*

Общая структура блока

Прежде чем писать первую программу, следует разобраться с настройкой *параметров* блока. В нижней части каждого блока находится группа элементов управления, с помощью которых можно определить его поведение. Каждый блок имеет уникальные параметры, однако все блоки объединяет общая структура, что значительно облегчает использование среды EV3.

Рассмотрим элементы управления блока **Звук** (Sound). Этот блок довольно типичен, поэтому все обсуждаемые здесь концепции будут применимы к большинству других блоков. На рис. 2.7 показаны параметры блока **Звук** (Sound) после его добавления в программу и до внесения каких-либо изменений.

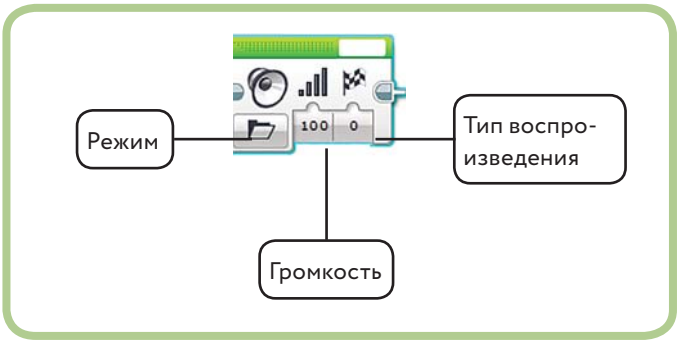


*Рис. 2.7. Параметры блока **Звук** (Sound)*

Первый элемент управления **Режим** (Mode) позволяет выбрать *режим* блока. Режим блока определяет его основную функцию. Четыре режима блока **Звук** (Sound) позволяют воспроизвести звуковой файл, ноту или тон, а также остановить воспроизведение любого звука. Параметры блока изменяются в соответствии с выбранным режимом. На рис. 2.7 показан блок **Звук** (Sound) с выбранным режимом **Воспроизвести файл** (Play File). В этом режиме можно настроить громкость и тип воспроизведения, который сообщает программе, как следует воспроизводить звук — один раз с приостановкой программы, один раз с одновременным выполнением следующих блоков или многократно, пока программа не завершится или не встретит блок **Звук** (Sound) в режиме **Остановка** (Stop).

Остальные режимы предусматривают другие параметры. На рис. 2.8 показан блок **Звук** (Sound) с выбранным режимом **Воспроизвести тон** (Play Tone). Этот режим позволяет установить частоту и продолжительность воспроизводимого тона.

*Рис. 2.8. Блок **Звук** в режиме **Воспроизвести тон** (Play Tone)*

Для изменения режима или параметра щелкни по текущему значению. Способ ввода нового значения зависит от типа параметра. В одних случаях требуется просто ввести число. В других для выбора значения может использоваться список или ползунковый регулятор. В режиме **Воспроизвести ноту** (Play Note) выбор ноты для блока **Звук** (Sound) осуществляется с помощью небольшой клавиатуры (рис. 2.9).

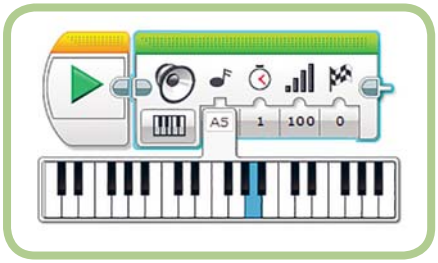


*Рис. 2.9. Выбор ноты для блока **Звук** (Sound)*

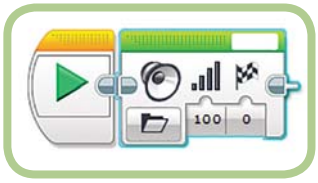
Твоя первая программа

В своей первой программе ты будешь использовать блок **Звук** (Sound) для того, чтобы модуль EV3 приветствовал тебя. Для начала запусти программное обеспечение MINDSTORMS EV3 и создай новый проект. Редактор контента тебе не потребуется, поэтому лучше закрыть его, чтобы освободить место в области программирования. Выполни следующие действия, чтобы добавить блок **Звук** (Sound) в свою новую программу:

1. Выбери блок **Звук** (Sound) на палитре программирования.

2. Перетащи блок в область программирования. Помести блок **Звук** (Sound) справа от блока **Начало** (Start), который уже должен находиться в области программирования.

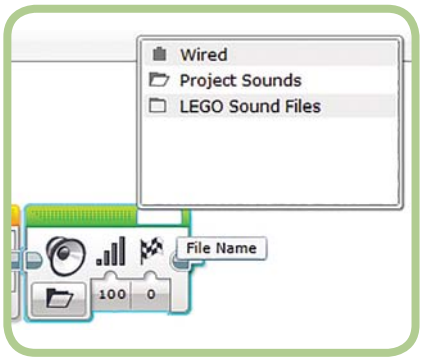
Твоя программа должна выглядеть так:



Если ты случайно возьмешь не тот блок или поместишь его не туда, куда нужно, выбери команду меню **Редактировать** (Edit) ⇒ **Отменить** (Undo) и начни заново.

Для большинства параметров блока **Звук** (Sound) можно оставить значения по умолчанию. Все, что тебе нужно сделать, — это выбрать звуковой файл для воспроизведения. Программное обеспечение EV3 поставляется с обширным набором звуковых файлов, организованных в папках подобно файлам на жестком диске компьютера.

3. Щелкни по белому полю в правом верхнем углу блока **Звук** (Sound). Откроется окно со списком звуковых файлов.



4. Щелкни по папке **Звуковые файлы LEGO** (LEGO Sound Files), чтобы открыть ее, а затем выбери папку **Связь** (Communications). Прокрути список вниз и выбери файл **Hello**. В белом поле в правом верхнем углу блока **Звук** (Sound) теперь должно отображаться слово *Hello*.

Сохранение результатов работы

Прежде чем продолжить, дай своей программе название и сохрани проект. В этой книге на каждую главу, как правило, приходится один проект под названием *Chapter2*, *Chapter3* и т. д. Каждый проект содержит все программы, написанные в соответствующей главе. Измени имя своей программы на *Hello* и сохрани проект под названием *Chapter2*.

Для изменения назнания проекта дважды щелкни по вкладке с именем программы. Введи новое имя *Hello*

вместо выделенного текста. Вкладки должны выглядеть как на картинке.

Теперь сохрани проект, выбрав команду меню **Файл** (File) ⇒ **Сохранить проект** (Save Project). При первом сохранении проекта открывается диалоговое окно, позволяющее выбрать местоположение и имя файла проекта. Введи имя *Chapter2* и щелкни по кнопке **Сохранить** (Save), чтобы создать файл *Chapter2.ev3*. Этот файл содержит всю информацию о твоей программе, включая конфигурацию и расположение используемых блоков. Файлы с расширением *.ev3* используются только в среде MINDSTORMS; файл такого формата нельзя редактировать, используя другие программы.

ПРИМЕЧАНИЕ **Сохраняй свою работу как можно чаще. Выполняй сохранение перед загрузкой своей программы в модуль и, конечно же, перед тем, как ответить на телефонный звонок или отправиться выгуливать собаку. Необходимость восстанавливать результаты нескольких часов работы, потерянные из-за пренебрежения сохранением, очень раздражает!**

СОЗДАНИЕ РЕЗЕРВНЫХ КОПИЙ

Время от времени при изменении программы вместо ее улучшения ты можешь создать ужасную путаницу, вернуть которую в исходное состояние будет невозможно. Например, ты можешь забыть, как была организована твоя программа, когда она работала.

Профессиональные разработчики программного обеспечения используют специальные инструменты, называемые *системами управления исходным кодом*, для сохранения версий своей работы, чтобы избежать этой проблемы. Однако ты можешь обеспечить себе такие же преимущества, сохраняя копии рабочих версий своей программы по мере работы над ней. Используй одну из таких резервных копий при возникновении проблемы. Создавать копию следует после добавления каждой новой функции и перед внесением значительных изменений. Например, если ты работаешь над программой, решающей четыре задачи, то можешь сохранить копию под названием *Task1* после создания работающей первой части. При работе над второй задачей ты можешь сохранить копию под названием *Task1_Task2*. Таким образом, в случае необходимости ты всегда сможешь вернуться на шаг назад.

Запуск программы

После сохранения программы можно попробовать ее запустить. Первым делом убедись в том, что модуль EV3 включен и подсоединен к твоему компьютеру с помощью USB-кабеля, Bluetooth или Wi-Fi-соединения. Соединение USB легко установить: для этого достаточно подключить модуль EV3 к компьютеру с помощью кабеля. Однако при его использовании модуль EV3 должен находиться рядом с компьютером. Соединение Bluetooth или Wi-Fi сложнее настроить, но оно позволяет роботу свободно перемещаться. Файл справки EV3 содержит инструкции по подключению модуля к компьютеру. Ты можешь открыть его, выбрав команду меню **Справка** ⇒ **Отобразить справку EV3** (Help ⇒ Show EV3 Help).

Щелкни по средней кнопке **Загрузить и запустить** (Download and Run) для загрузки и запуска своей программы. В ответ на это модуль EV3 должен сказать “Hello!”

Свойства проекта

Проект EV3 может содержать несколько программ, и в каждой из них может быть множество звуковых файлов или изображений. Страница **Свойства проекта** (Project Properties) содержит обзор проекта и позволяет управлять всеми ресурсами, которые в нем используются. Щелкни по значку в виде небольшого гаечного ключа слева от имени программы, чтобы открыть страницу **Свойства проекта** (Project Properties). На рис. 2.10 показана страница **Свойства проекта** (Project Properties) для проекта *Chapter 2* с несколькими заполненными полями.

На странице **Свойства проекта** (Project Properties) можно добавить название и описание проекта, а также просмотреть все используемые в нем программы, изображения, звуковые файлы и контейнеры «Мои блоки». Продолжив чтение этой

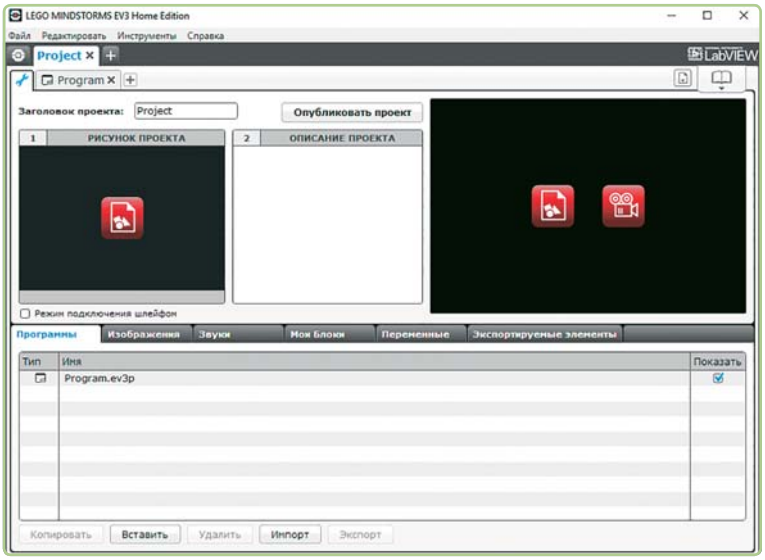


*Рис. 2.10. Страница **Свойства проекта** (Project Properties)*

книги, ты узнаешь, как с помощью этого окна можно импортировать и экспортировать программы и контейнеры «Мои блоки» для их использования в нескольких проектах.

Твоя вторая программа

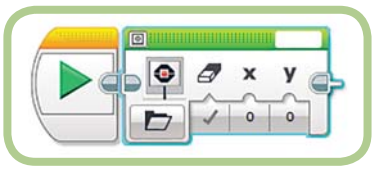
Твоя вторая программа под названием *HelloDisplay* будет похожа на программу *Hello* за исключением того, что вместо блока **Звук** (Sound) в ней будет использоваться блок **Экран** (Display) для отображения слова *Hello* на экране модуля EV3. Твоя первая попытка создания данной программы окажется неудачной. Это сделано намеренно, чтобы дать тебе возможность посмотреть, что происходит, когда программа не работает должным образом, а также научиться ее исправлять. Для создания исходной версии программы выполни следующие действия:

1. Выбери команду меню **Файл** (File) ⇒ **Новая программа** (New Program), чтобы добавить в проект новую программу.

2. Измени имя программы на *HelloDisplay*. Вкладки в верхней части области программирования должны выглядеть так:

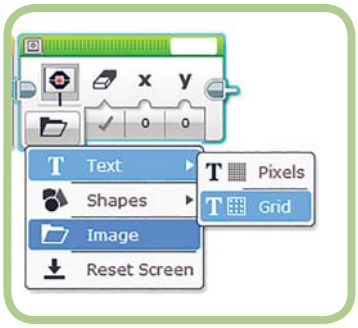
3. Перетащи блок **Экран** (Display) с палитры программирования.

4. Помести блок **Экран** (Display) рядом с блоком **Начало** (Start). Твоя программа должна выглядеть так:



Теперь нужно настроить блок **Экран** (Display) так, чтобы на экране модуля отобразилось слово “Hello”. Блок **Экран** (Display) предусматривает несколько параметров (подробнее в гл. 14). По умолчанию блок настроен на отображение изображения, поэтому первое, что нужно сделать, это настроить его для отображения текста.

5. Щелкни по изображению папки в левом нижнем углу блока. Затем выбери режим **Текст** (Text) ⇒ **Сетка** (Grid).



По умолчанию в блоке отображается слово “MINDSTORMS”, поэтому следующим шагом является его изменение на “Hello”.

6. Щелкни по тексту в белом поле в верхнем правом углу блока **Экран** (Display) и замени слово *MINDSTORMS* на ***Hello***.

Теперь программа должна выглядеть так, как показано на рис. 2.11.

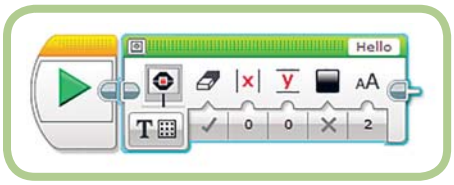


Рис. 2.11. Программа HelloDisplay

Теперь загрузи и запусти свою программу. После загрузки программы модуль EV3 издает звук, однако слово *Hello* не отображается на экране. Что случилось?

Дело в том, что эта программа содержит программную ошибку. *Отладка* — это процесс поиска и исправления ошибок. Как и все программисты, ты потратишь много времени на отладку. С первой попытки очень редко удается написать программу без каких-либо ошибок. Запуск программы, выявление и устранение проблем — все это неотъемлемые этапы процесса программирования. Исправление ошибок может показаться утомительным, однако это занятие также может быть невероятно полезным. Думай об этом как о решении головоломки и помни, что процесс программирования всегда должен быть интересным!

Один из способов исправления этой программы заключается в добавлении блока **Ожидание** (Wait) после блока **Экран** (Display). Этот блок можно использовать для того, чтобы программа приостановила работу на пять секунд. Этого будет достаточно для того, чтобы можно было прочитать слово на экране модуля.

Блок **Ожидание** (Wait) находится на оранжевой вкладке палитры программирования. На рис. 2.12 показаны блоки управления операторами с выделенным блоком **Ожидание** (Wait).

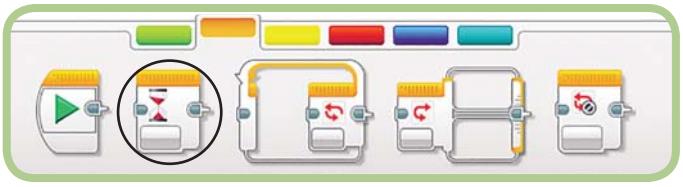


*Рис. 2.12. Блок **Ожидание** (Wait) на палитре с блоками управления операторами*

Для исправления программы выполни следующие действия:

1. Помести блок **Ожидание** (Wait) справа от блока **Экран** (Display). Твоя программа должна выглядеть так, как показано на рис. 2.13.

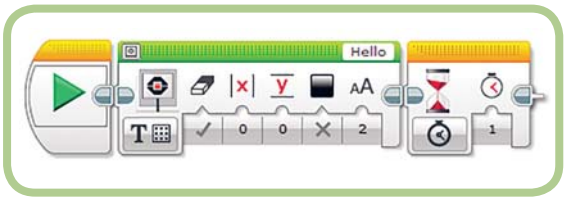


*Рис. 2.13. Программа после добавления блока **Ожидание** (Wait)*

2. По умолчанию блок **Ожидание** (Wait) приостанавливает выполнение программы на одну секунду. Чтобы успеть прочитать текст на экране, измени значение под небольшим значком в виде часов с 1 на **5**. Это приостановит выполнение программы на пять секунд. Блок **Ожидание** (Wait) с измененным параметром показан на рис. 2.14.

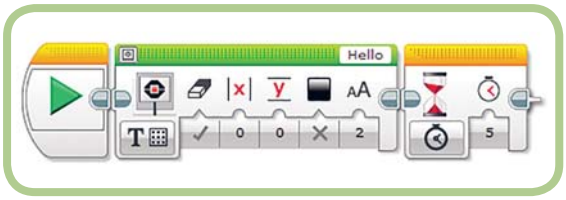


*Рис. 2.14. Блок **Ожидание** (Wait), приостанавливающий выполнение программы на пять секунд*

Теперь после загрузки и запуска программы слово "*Hello*" будет отображаться на экране модуля в течение пяти секунд, что нам и требуется. Добавление блока **Ожидание** (Wait) позволило решить эту проблему.

ПРИМЕЧАНИЕ Почему в первой программе не возникла такая же проблема? В отличие от блока **Экран** (Display), в блоке **Звук** (Sound) по умолчанию выбран вариант **Ожидать завершения** (Wait for Completion) (рис. 2.15), который приостанавливает программу на время воспроизведения звука. Если выбрать вариант **Воспроизвести один раз** (Play Once) вместо варианта **Ожидать завершения** (Wait for Completion), то первая программа даст сбой так же, как и вторая.

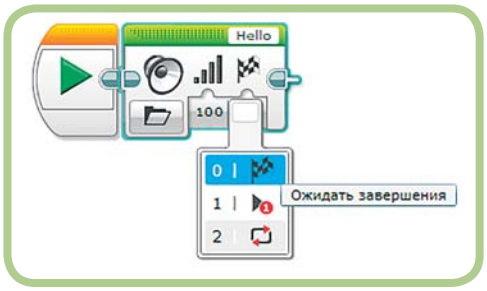


*Рис. 2.15. Блок **Звук** (Sound) с выбранным вариантом **Ожидать завершения** (Wait for Completion)*

Комментарии

Программисты используют *комментарии* для добавления описательного текста в свои программы. В этих текстах они объясняют принцип работы программы или причину, по которой было принято то или иное решение в процессе ее создания. Например, можно было бы добавить к предыдущей программе комментарий, объясняющий, почему в нее был добавлен блок **Ожидание** (Wait).

Из гл. 1 ты узнал, что хорошая программа должна быть легко изменяема и понятна другим программистам. Хорошие комментарии важны для достижения обеих этих целей. Очень сложно понять, как работает программа, просто глядя на параметры каждого блока. Краткое описание на простом человеческом языке сделает программу намного более понятной. Представь, что тебе нужно описать свою программу другу. Ты не будешь просто перечислять используемые блоки; вместо этого ты опишешь, что делает программа в целом, вероятно, подробно остановившись на самых сложных ее аспектах. Комментарии также помогают запомнить, почему программа была написана определенным образом, что упрощает повторное использование собственных программ.

Комментарии не влияют на работу программы, поскольку модуль EV3 полностью игнорирует их. Его интересует только порядок и параметры блоков, из которых состоит программа.

Добавление комментария

Попробуй добавить комментарий в программу *HelloDisplay*, чтобы объяснить, почему в нее был включен блок **Ожидание** (Wait). Комментарий может быть таким: «Приостанови выполнение программы на 5 секунд, чтобы пользователь успел прочитать текст на экране». Для добавления комментариев в программу щелкни по кнопке **Комментарий** (Comment) (💬) на панели инструментов.

Выполни следующие действия, чтобы добавить комментарий:

1. Щелкни в области программирования над блоком **Ожидание** (Wait). Новый комментарий появится здесь после выбора инструмента **Комментарий** (Comment).

2. Щелкни по кнопке **Комментарий** (Comment) на панели инструментов. Небольшое окно с комментарием появится там, где до этого был выполнен щелчок (рис. 2.16).

Рис. 2.16. Новое окно для комментария

3. Щелкни в центре окна для комментария, чтобы выделить его.

4. Введи комментарий: **Приостанови выполнение программы на 5 секунд, чтобы пользователь успел прочитать текст на экране**. Нажми клавишу **Enter**, чтобы перейти на следующую строку, когда ширина текста превысит ширину блока (комментарий будет легче читать, если он не будет слишком выдаваться за пределы блока).

На рис. 2.17 показана программа с добавленным комментарием. Теперь любой, кто посмотрит на эту программу, поймет, зачем в ней используется блок **Ожидание** (Wait).

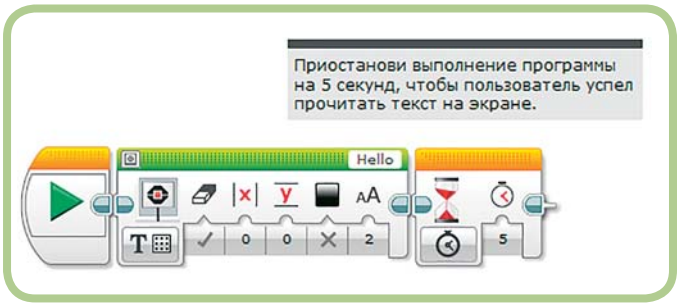


*Рис. 2.17. Комментарий, объясняющий причину добавления блока **Ожидание** (Wait)*

Нюансы работы с комментариями

При написании комментариев имей в виду:

— нажатие клавиши **Enter** при вводе комментария позволяет продолжить его на следующей строке;
— щелчок по комментарию позволяет выделить его;
— после выделения комментария щелчок по его тексту позволяет изменить комментарий;
— выделенный комментарий можно удалить, нажав клавишу **Delete**;
— размер окна с комментарием можно изменить с помощью маркеров, расположенных по его краям; они появляются при наведении указателя мыши на комментарий;
— выделенный комментарий можно переместить, перетащив его с помощью мыши.

Контекстная справка

Обилие блоков с многочисленными параметрами позволяет создавать самых разнообразных роботов. Тем не менее изучение всех этих блоков и их параметров может оказаться сложной задачей. Программное обеспечение EV3 предусматривает контекстную справку, которая может в этом помочь. Чтобы получить доступ к ней, выбери команду меню **Справка** (Help) ⇒ **Отобразить контекстную справку** (Show Context Help). Появится небольшое окно с кратким, но полезным описанием того элемента, на который наведен указатель мыши. Каждая тема содержит ссылку **Дополнительная информация** (More Information) на случай, если тебе потребуются более полные сведения. Если оставить это маленькое окно открытым, можно быстро ознакомиться со всеми вариантами, доступными при создании программы. Справка, отображаемая при выборе ноты для блока **Звук** (Sound) в режиме **Воспроизвести ноту** (Play Note), показана на рис. 2.18.

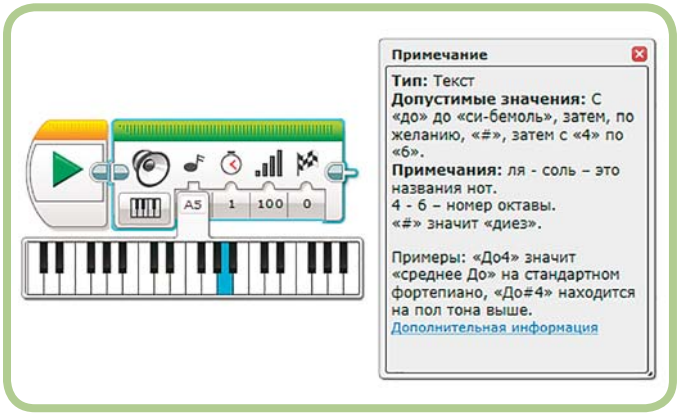


Рис. 2.18. Контекстная справка для выбора ноты

Заключение

Мы завершили знакомство со средой MINDSTORMS. Далее мы создадим простого робота под названием *TriBot*, который будет использовать программы из следующих глав. Затем ты узнаешь о множестве блоков, предусмотренных в программном обеспечении EV3, а также как их можно объединить, чтобы робот TriBot выполнял множество задач.

3

TriBot: тестовый робот

В этой главе описан процесс сборки простого трехколесного робота под названием TriBot, который использует все датчики EV3 (рис. 3.1). Этого робота можно применять для запуска примеров программ из остальных глав книги, а также для дальнейшего тестирования своих собственных программ. Кроме того, ты создашь простой подъемный рычаг для проведения экспериментов со средним мотором EV3.

Компоненты робота TriBot

Ты можешь создать робота, используя домашнюю или образовательную версию конструктора EV3. На большей части изображений в этой главе показаны компоненты домашней версии. Сначала рассмотрим различия между этими двумя наборами. В домашней версии конструктора используются колеса меньшего размера, чем в образовательной версии (рис. 3.2).

Домашняя версия конструктора предусматривает инфракрасный датчик и удаленный инфракрасный маяк, в то время как образовательная версия включает ультразвуковой датчик и гироскопический датчик. При использовании домашней версии можно пропустить шаги, связанные с гироскопическим датчиком. При использовании образовательной версии нужно заменить ультразвуковой датчик инфракрасным.

Компоненты каждого набора отличаются цветом. Так в домашней версии некоторые балки черного цвета, а в образовательной — серого. Цвет не имеет значения (кроме случаев, указанных в инструкциях). Просто убедись в том, что детали имеют правильный размер и форму. На рис. 3.3 и рис. 3.4 показаны компоненты, которые тебе понадобятся при использовании домашней и образовательной версии конструктора соответственно.

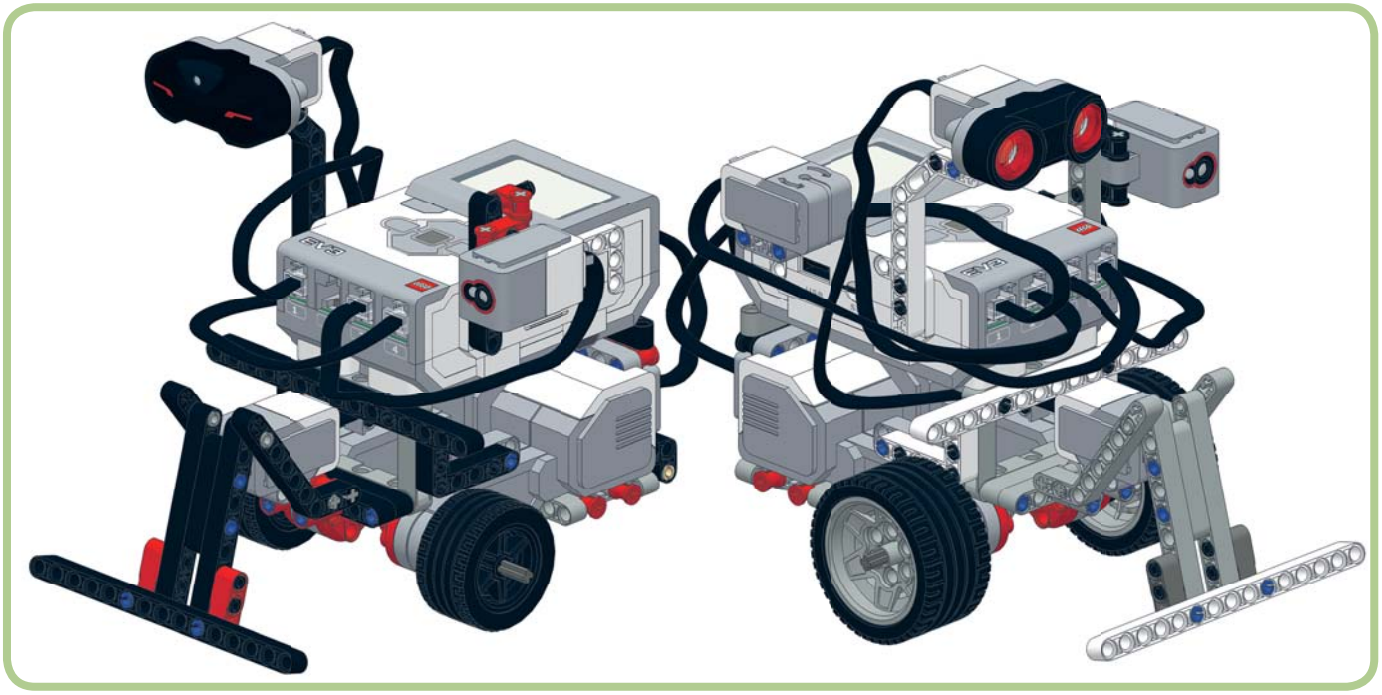


Рис. 3.1. Робот TriBot, собранный из деталей домашней версии конструктора (слева), и робот TriBot, собранный из деталей образовательной версии конструктора (справа)

Рис. 3.2. Колеса образовательной (слева) и домашней (справа) версии конструктора

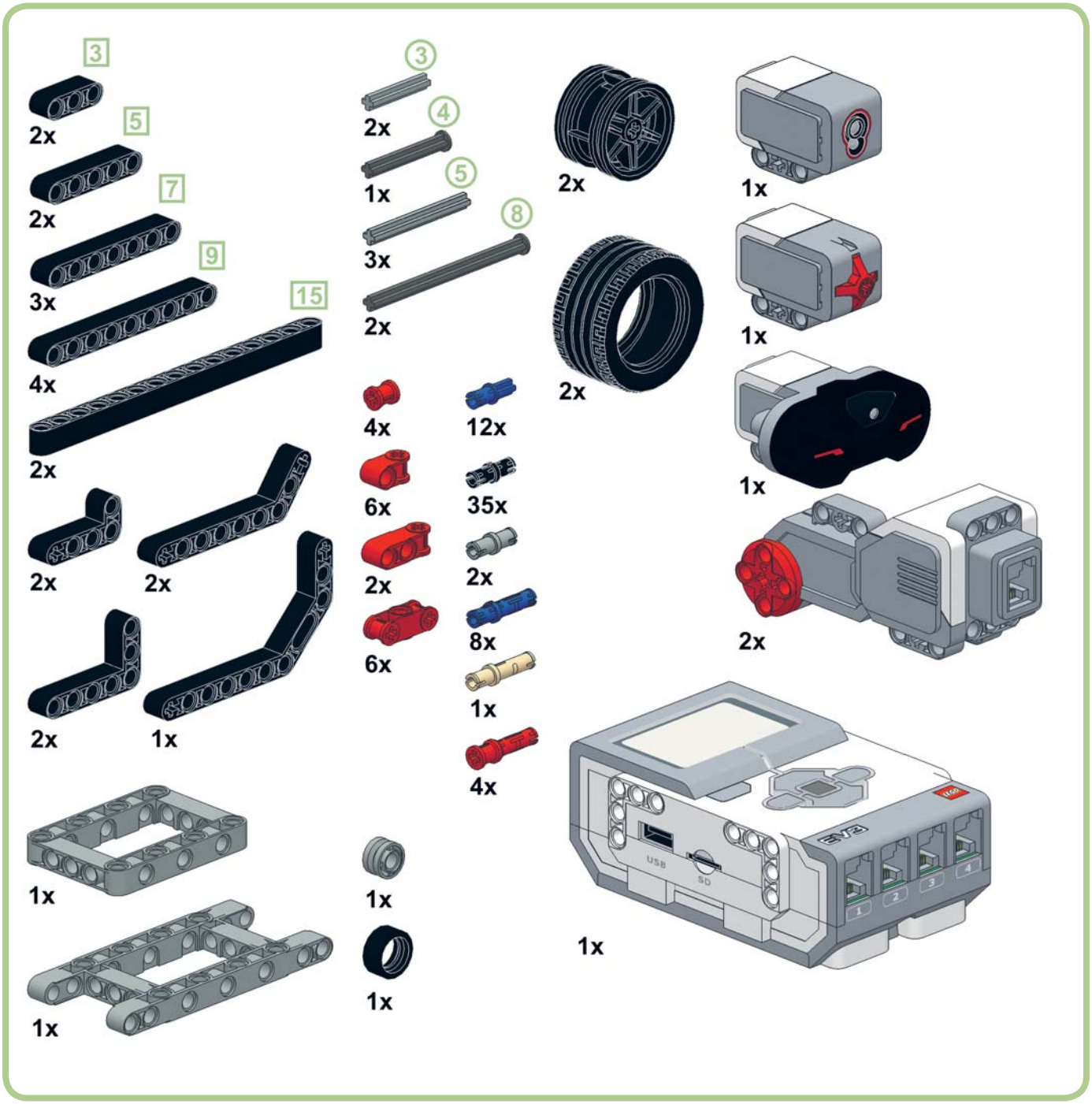


Рис. 3.3. Компоненты домашней версии конструктора (№31313), необходимые для сборки робота TriBot

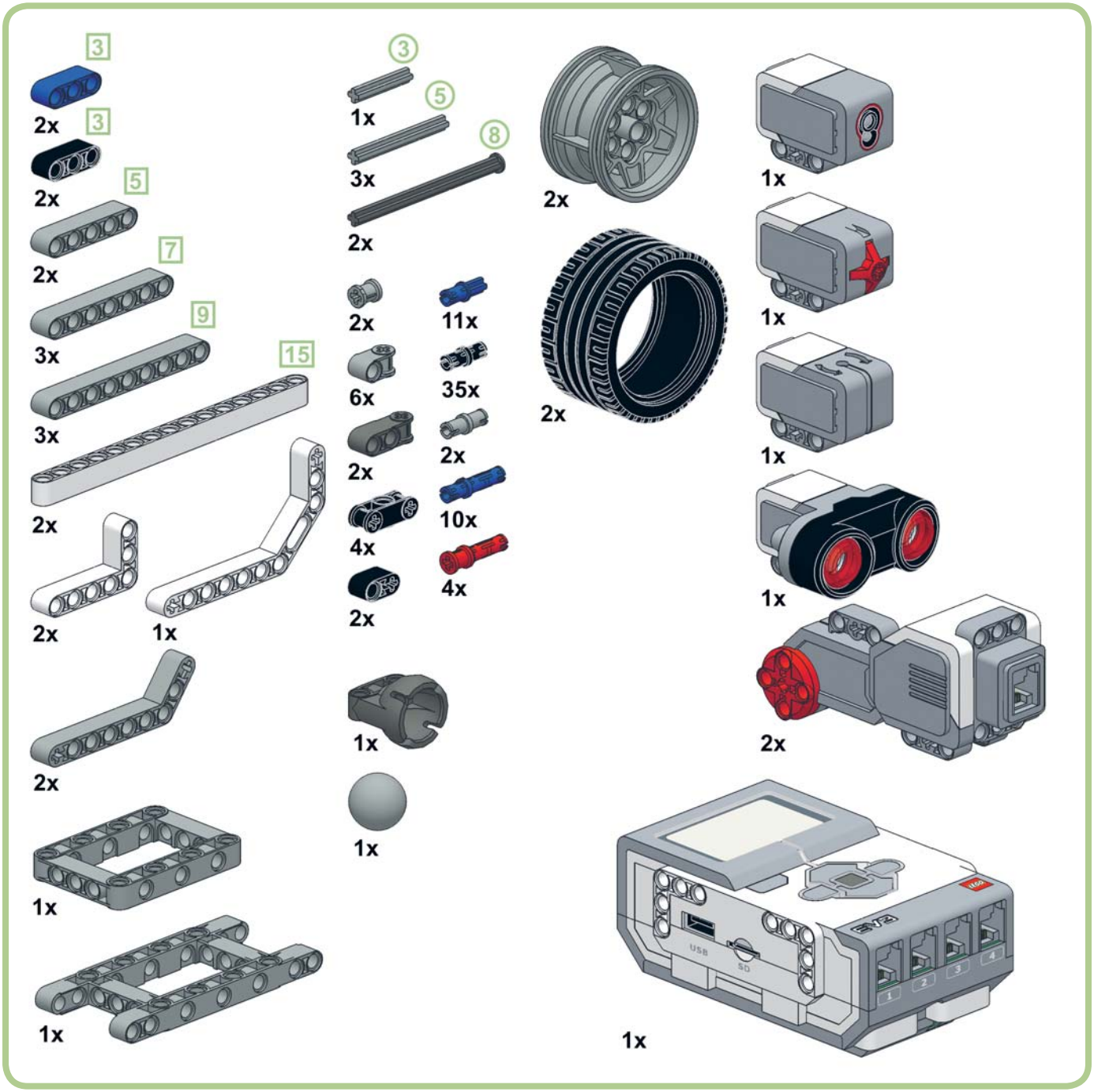
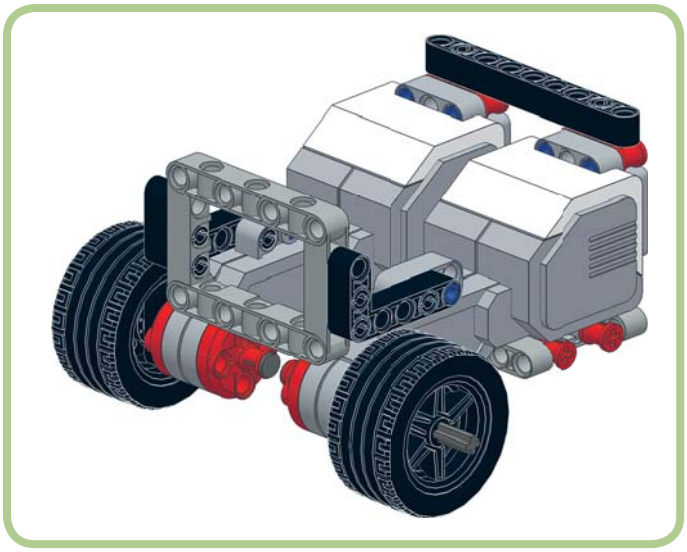


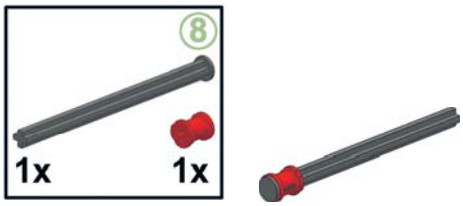
Рис. 3.4. Компоненты образовательной версии конструктора (№45544), необходимые для сборки робота TriBot

Сборка мотора и шасси

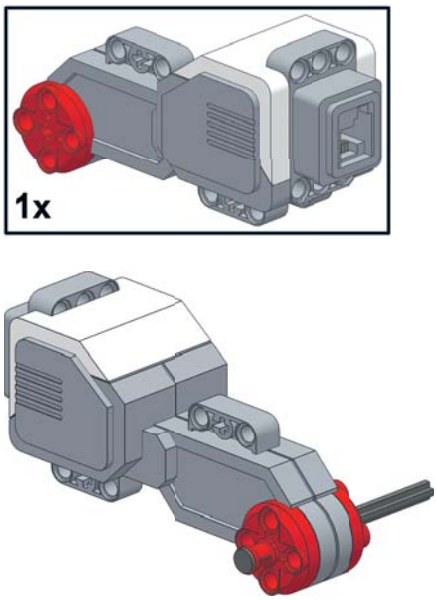
Сначала нужно собрать мотор и шасси.



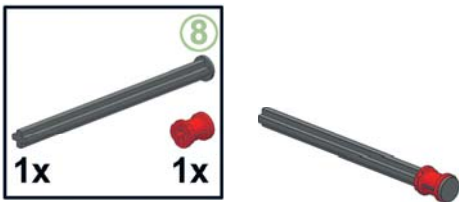
1



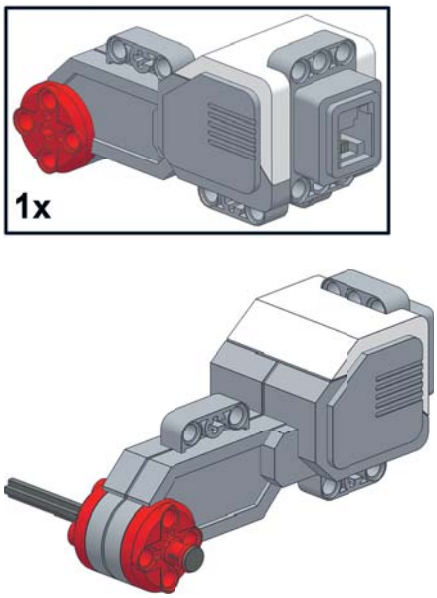
2



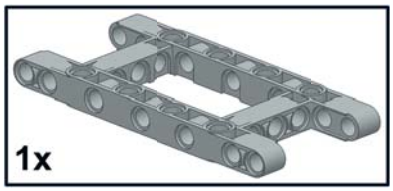
3

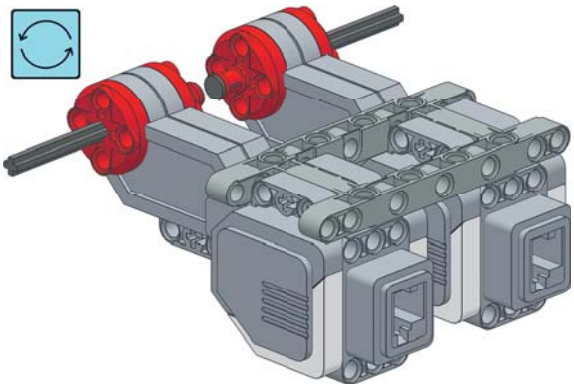


4



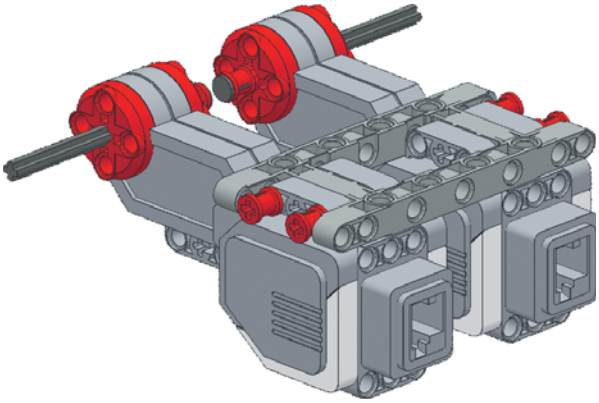
5





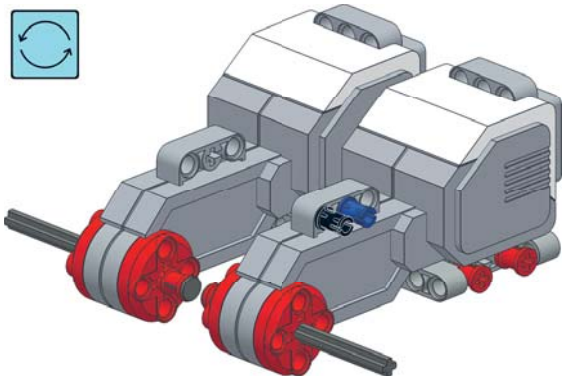
Переверни два мотора и поместии их рядом друг с другом. Затем добавь H-образную рамку.

6

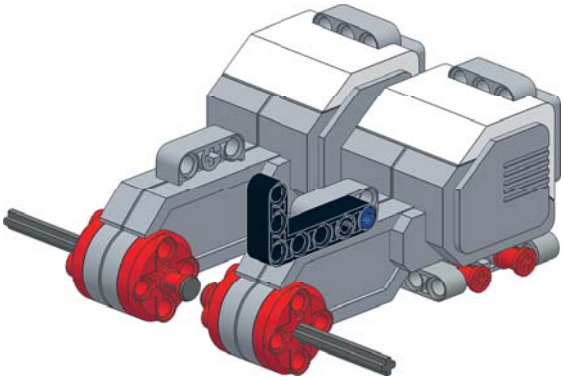


Используй четыре длинных красных штифта для соединения моторов и H-образной рамки.

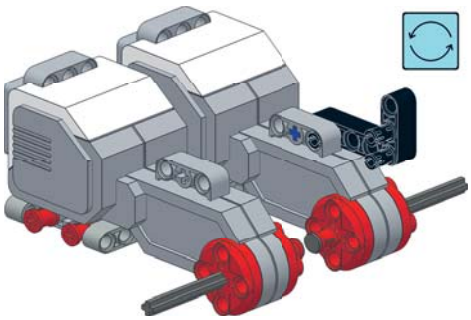
7



8

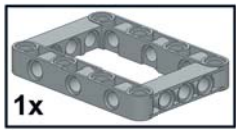


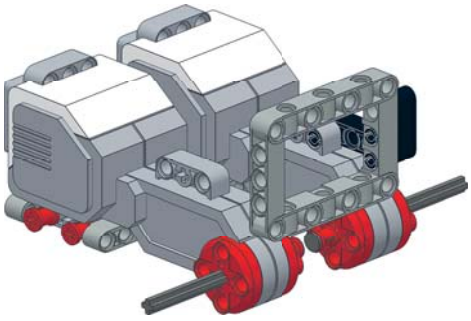
9



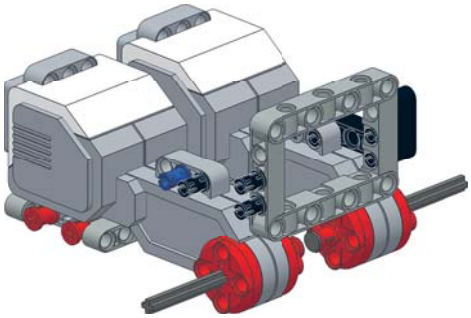
Соедини два штифта с черной Г-образной балкой, добавленной на шаге 8.

10

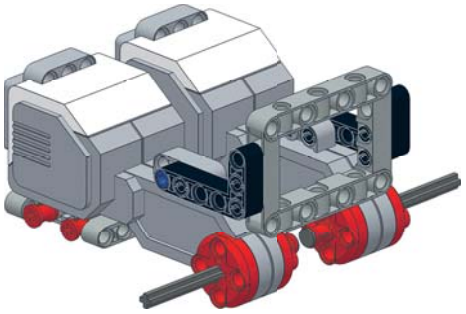




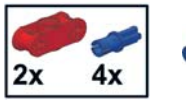
11

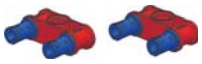


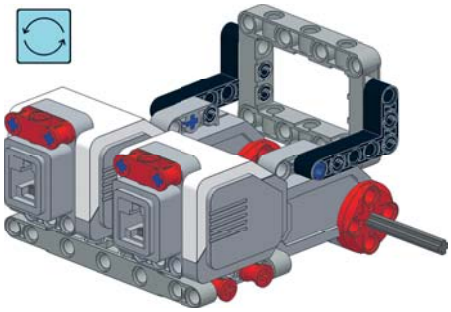
12



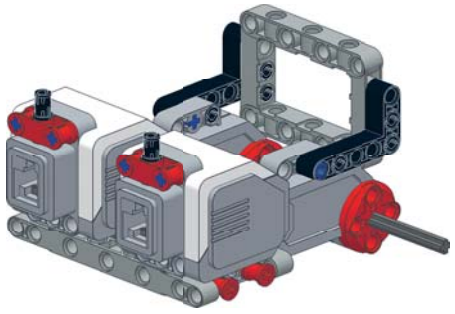
13



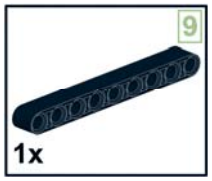


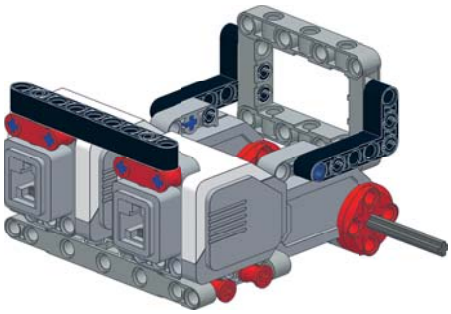


14

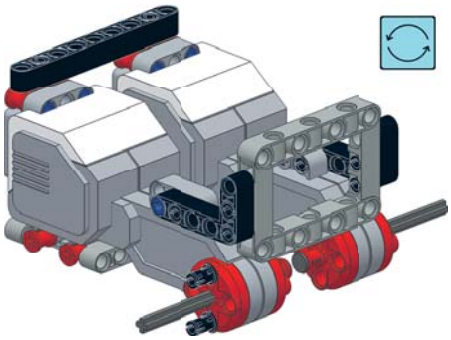


15



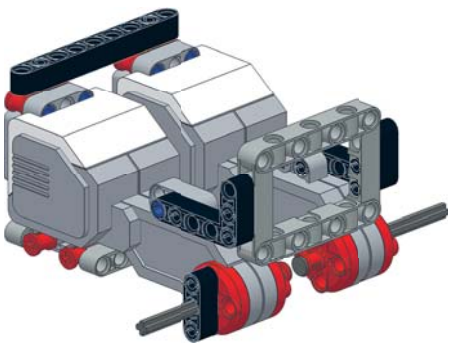


16

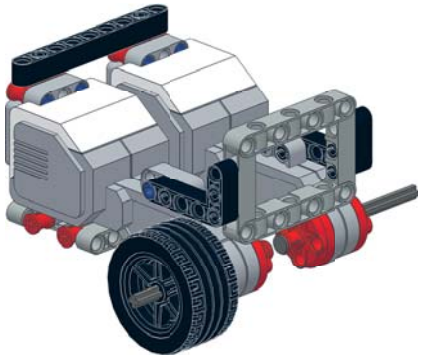


Теперь добавь небольшую балку к каждому мотору и присоедини колеса.

17

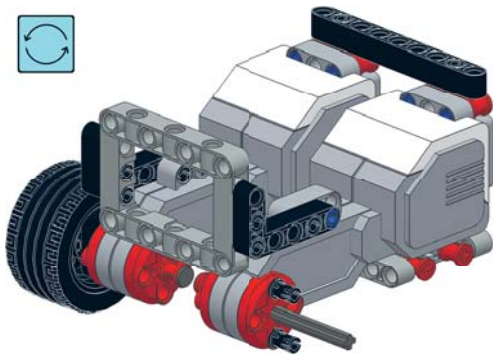


18

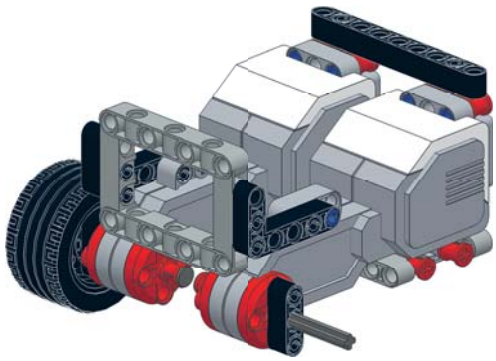


Убедись в том, что колесо прижато к балке с тремя отверстиями, прикрепленной к мотору. Эта балка предотвращает трение шины о другие части робота.

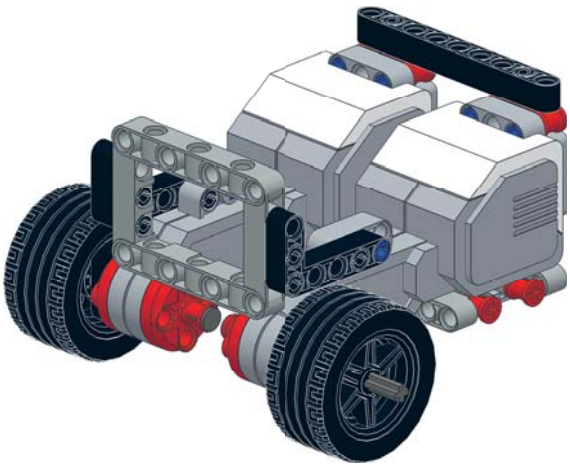
19



20



21

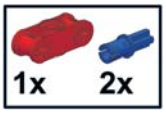


Сборка опорного ролика

В этом разделе показано, как собрать опорный ролик и прикрепить его к задней части робота TriBot. Далее приведены инструкции для домашней и образовательной версии конструктора. Следуй инструкции, относящейся к используемому набору.

Сборка опорного ролика из деталей домашней версии конструктора

1



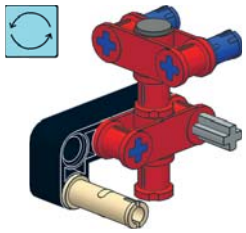
2

3

4

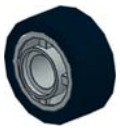
5

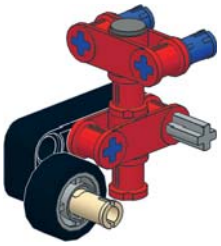
6



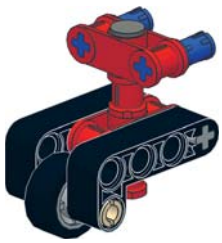
Убедись в том, что ты используешь желто-коричневый, а не синий штифт. Желто-коричневый штифт создает меньше трения, что позволяет ролику свободно вращаться.

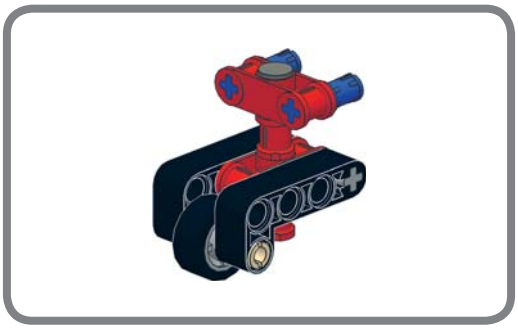
7



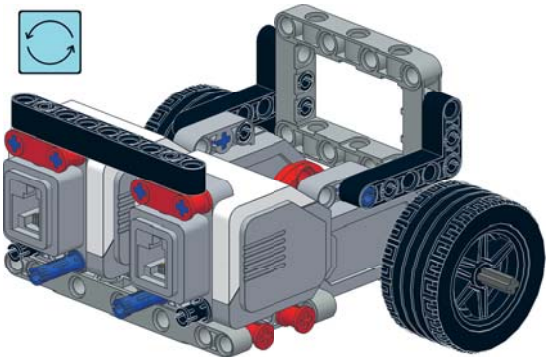


8

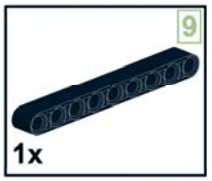


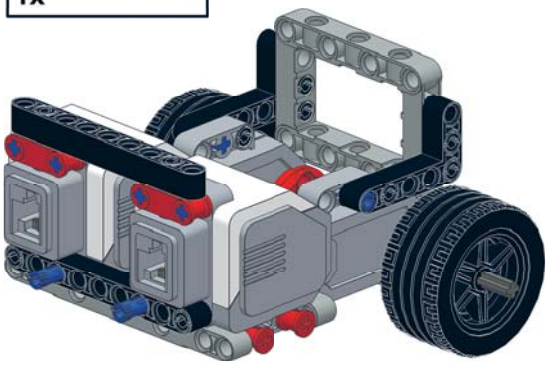


9

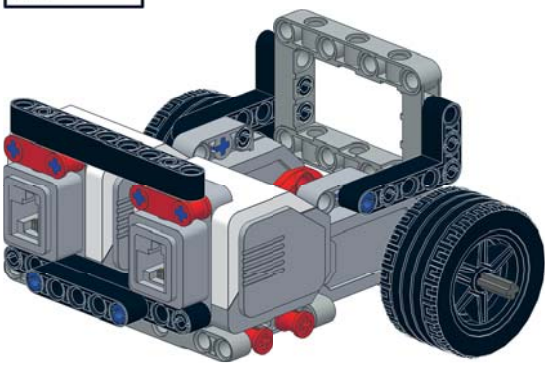


10

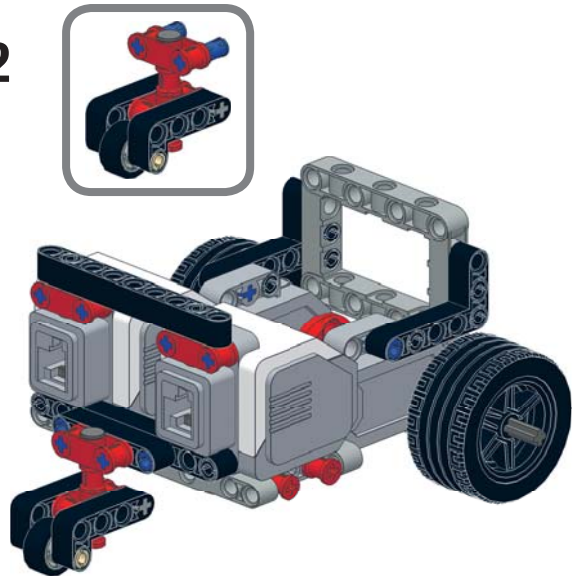




11



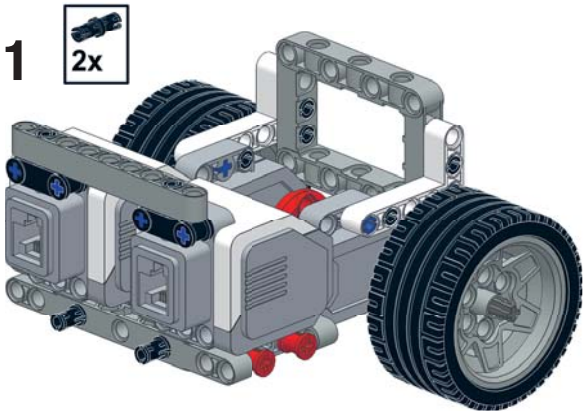
12



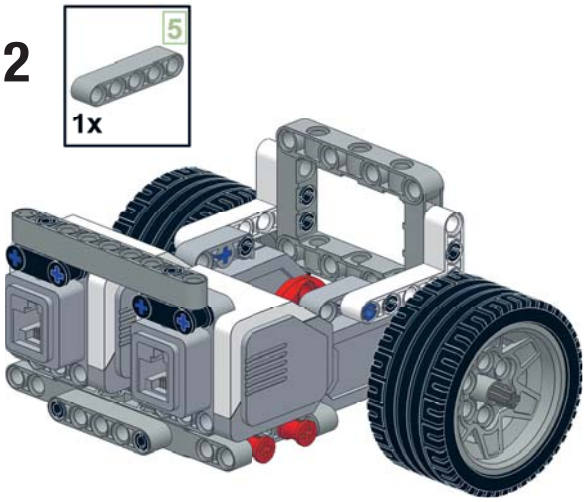
Сборка опорного ролика из деталей образовательной версии конструктора

Образовательная версия предусматривает специальную шаровую опору, с помощью которой можно добавить третье «колесо», используя меньшее количество деталей.

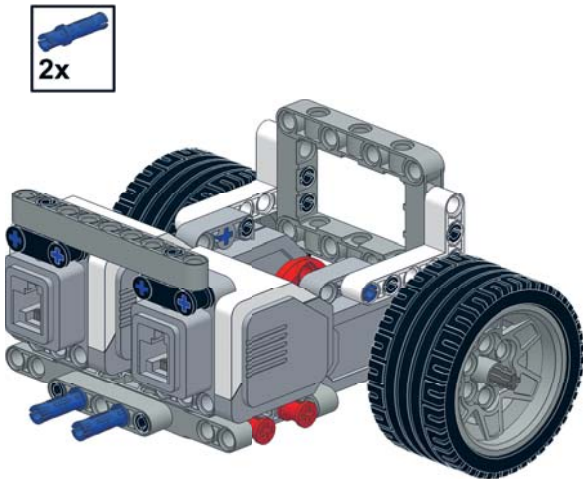
1



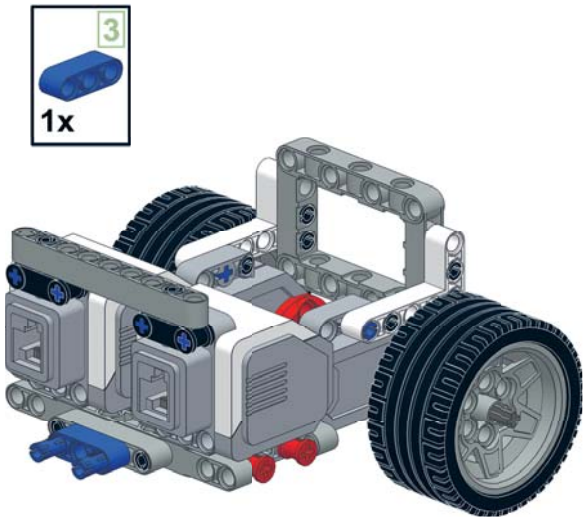
2



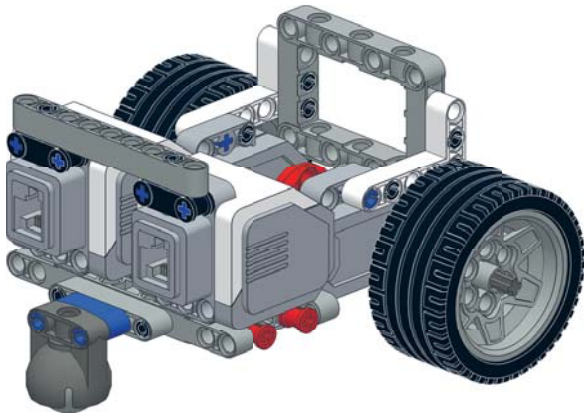
3



4



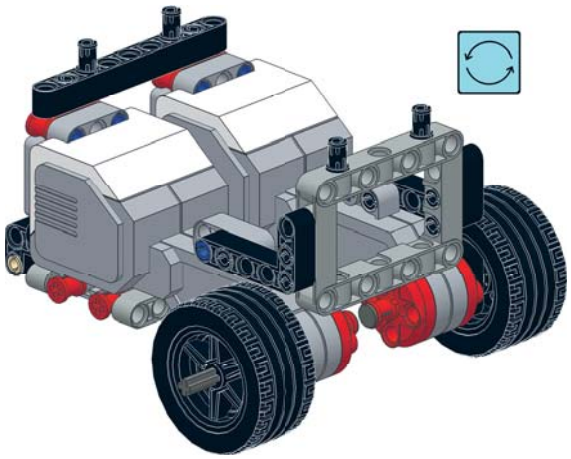
5



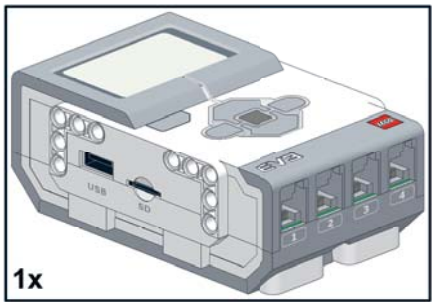
Установка модуля EV3

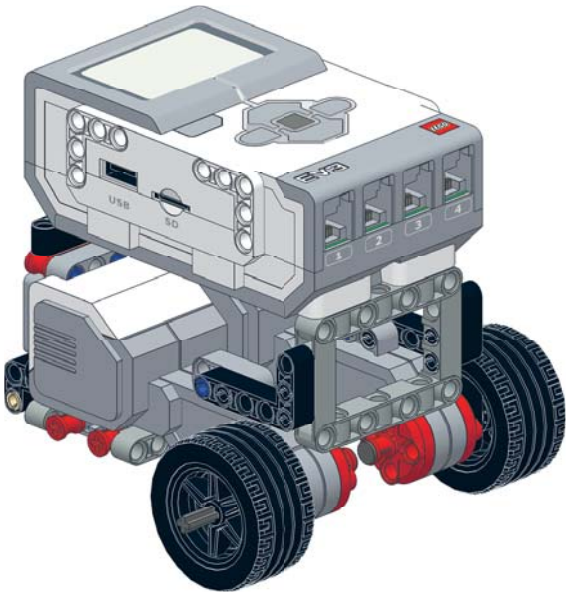
Теперь добавь четыре штифта к верхней части мотора и прикрепи модуль EV3.

1



2



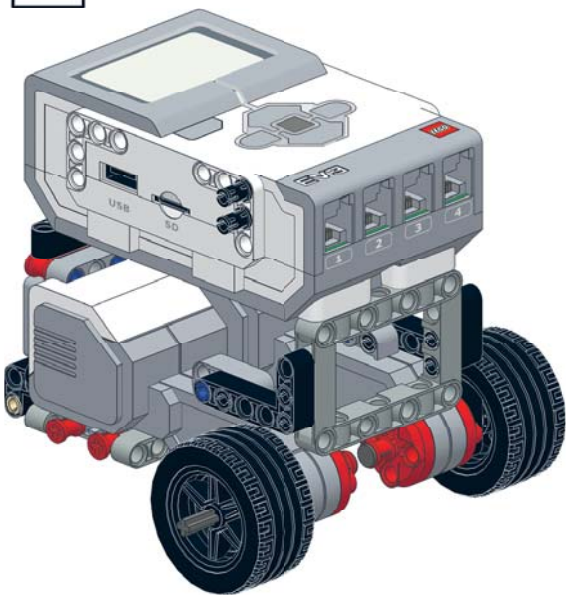


Монтаж инфракрасного или ультразвукового датчика

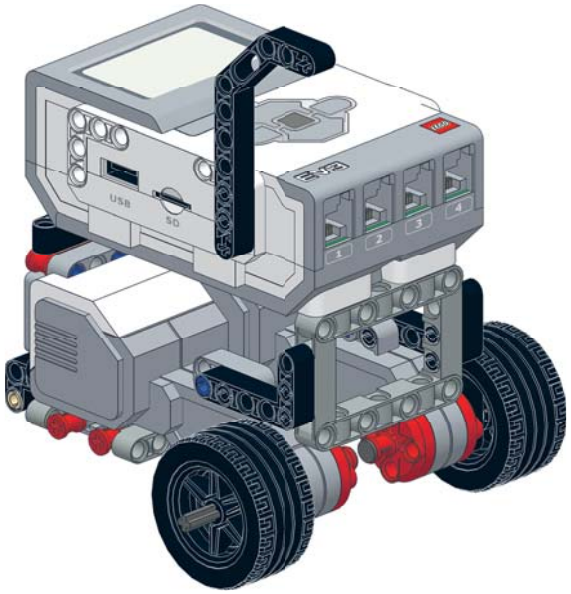
Теперь установи инфракрасный датчик.

На приведенных в этом разделе изображениях показан инфракрасный датчик из домашней версии конструктора. Замени его на ультразвуковой датчик при использовании образовательной версии.

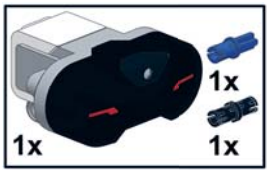
1

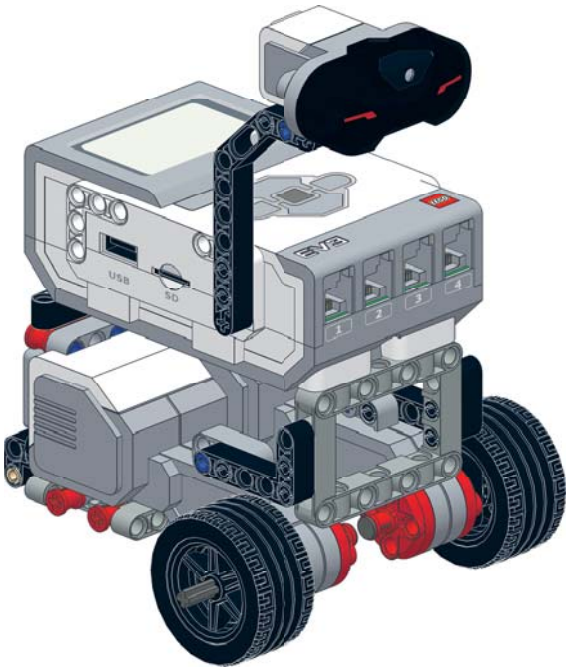


2



3

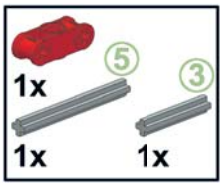




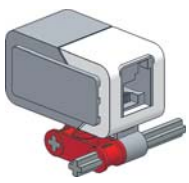
Монтаж датчика цвета

Теперь подключи датчик цвета. Сначала создай монтажный кронштейн.

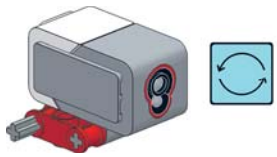
1



2

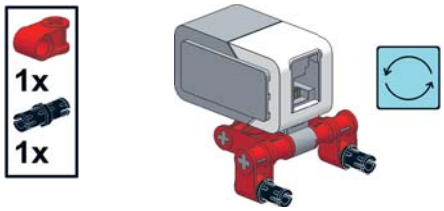


3

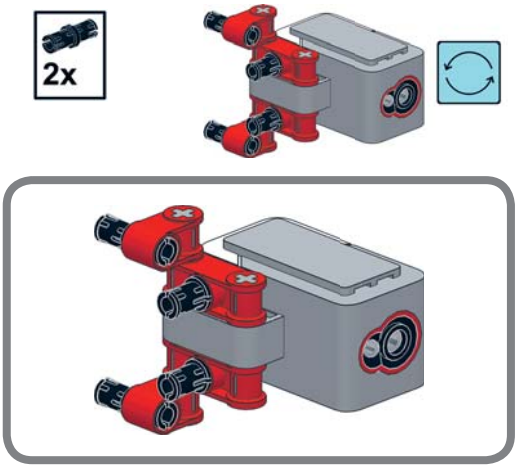


4

5

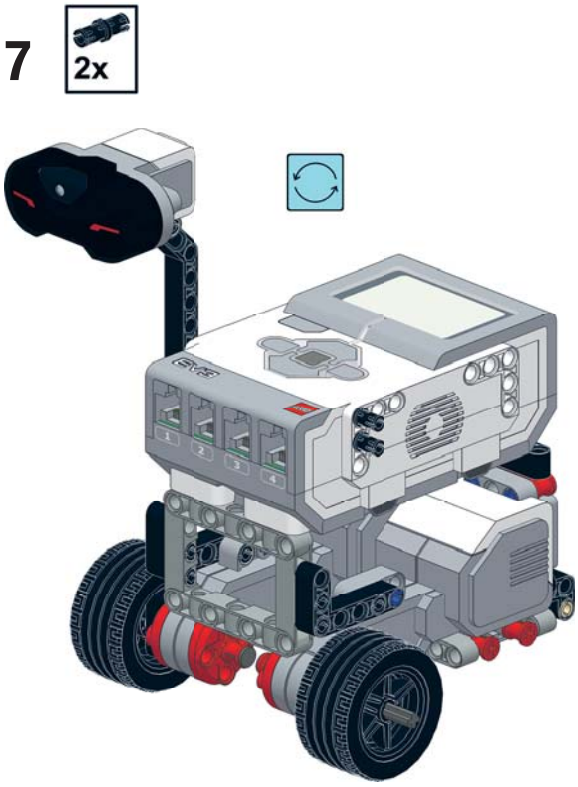


6

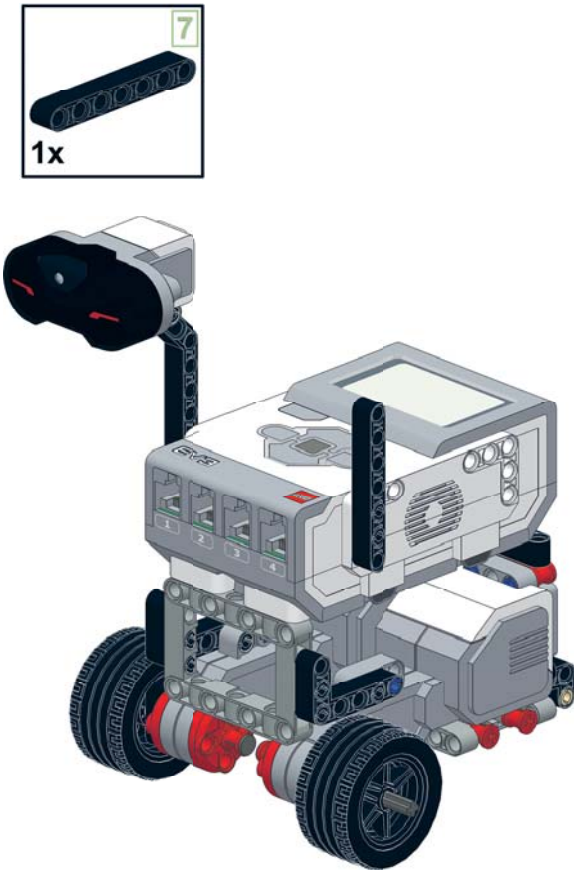


Присоедини балку к боковой части робота и прикрепи датчик.

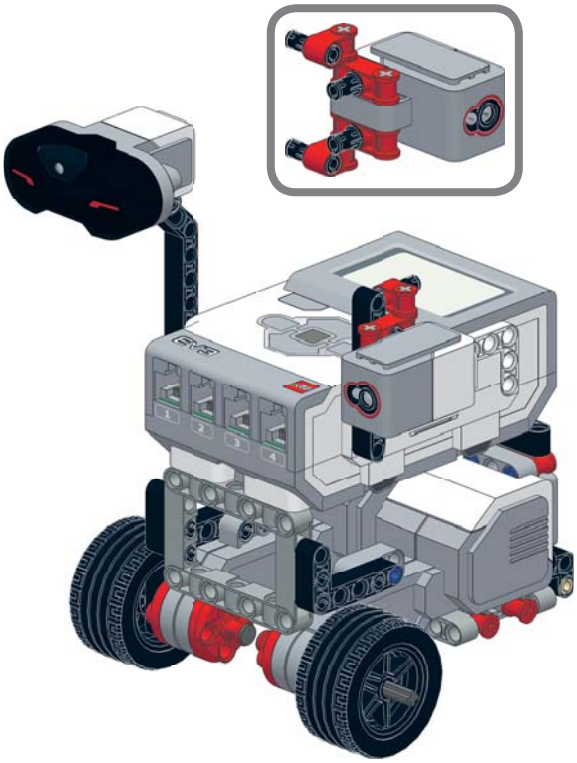
7



8



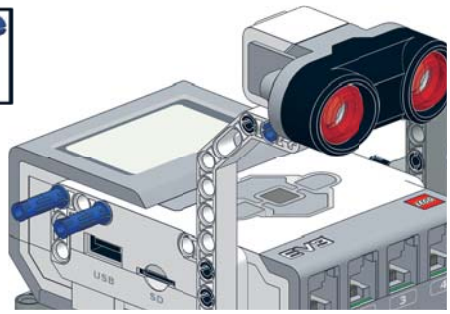
9



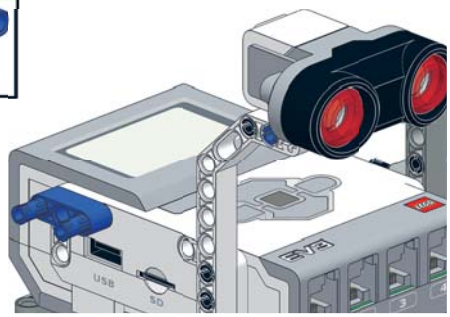
Монтаж гироскопического датчика (образовательная версия конструктора)

Если ты используешь образовательную версию конструктора, прикрепи гироскопический датчик к модулю с той же стороны, что и ультразвуковой датчик. (Пропусти эти шаги, если используешь домашнюю версию конструктора.)

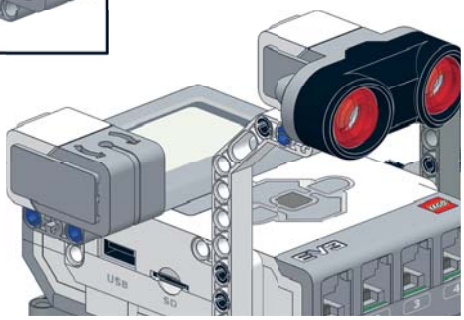
1



2



3

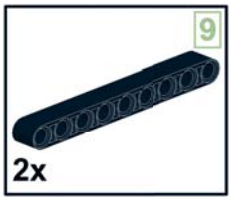


Сборка бампера для датчика касания

Теперь создай бампер для датчика касания. Начни со сборки рычага, который будет находиться перед датчиком касания.

1

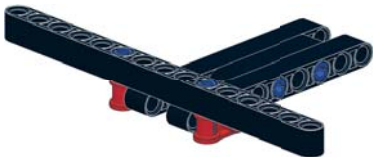
2



3

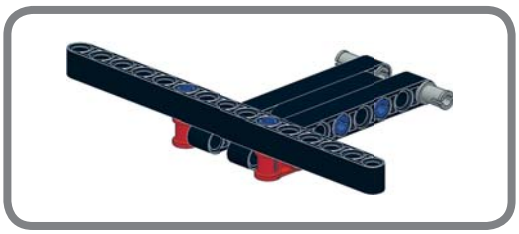
4

5



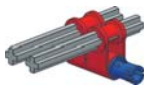
6

Убедись в том, что ты используешь серые, а не черные штифты. Серые штифты создают меньше трения, что позволяет рычагу свободно качаться.

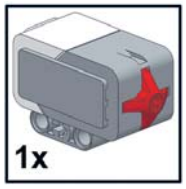


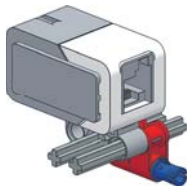
Собери остальную часть бампера.

7

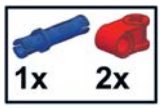


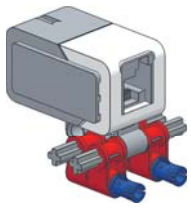
8



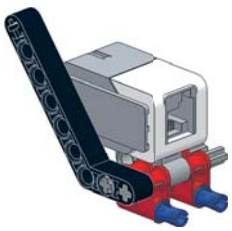


9

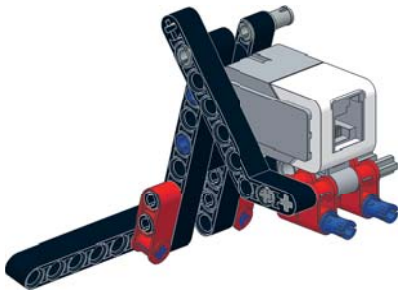




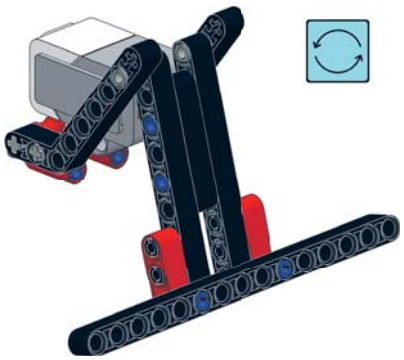
10



11

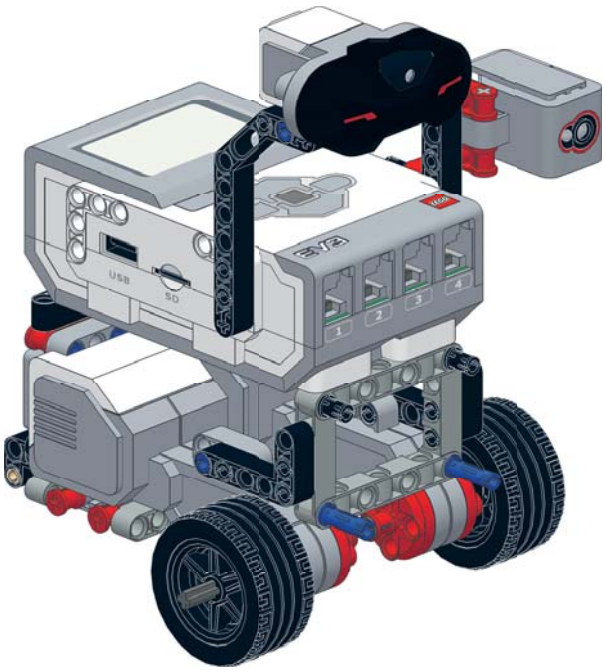


12

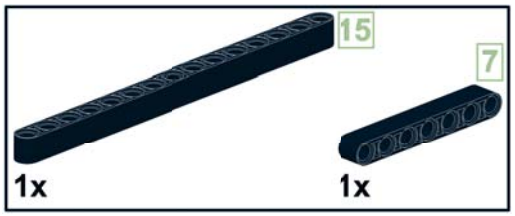


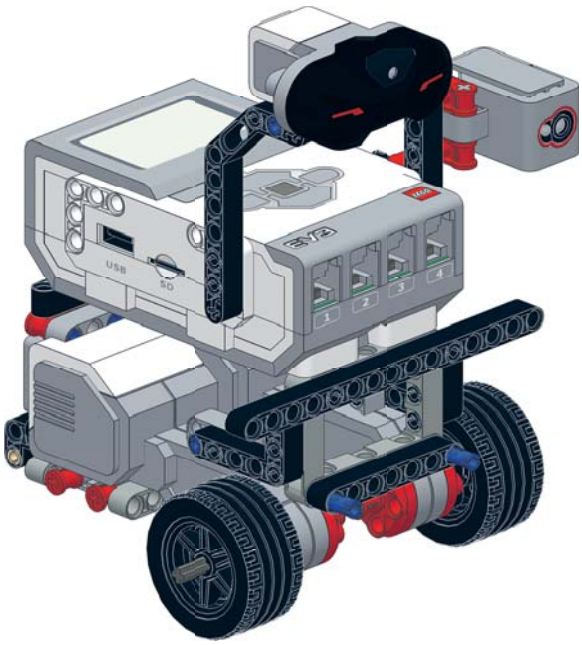
Прежде чем прикрепить бампер, добавь балки к передней части робота.

13



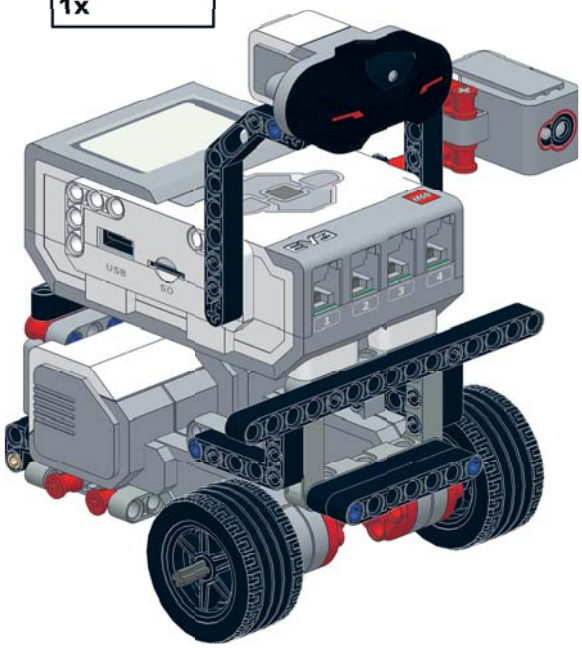
14





15

Теперь мы подключим кабели, а затем прикрепим бампер.



Подключение кабелей

Гораздо легче подключить кабели до прикрепления бампера. В табл. 3.1 перечислены характеристики кабелей для подключения моторов и датчиков.

Табл. 3.1. Характеристики кабелей

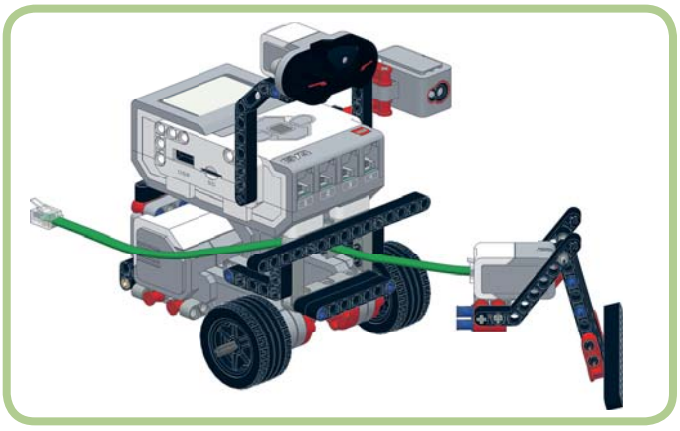
Мотор или датчик	Порт	Длина кабеля
Датчик касания	1	25 см
Гироскопический датчик	2	35 см
Датчик цвета	3	25 см
Инфракрасный или ультразвуковой датчик	4	35 см
Мотор на стороне инфракрасного (или ультразвукового) датчика	C	25 см
Мотор на стороне датчика цвета	B	25 см

Моторы используют порты B и C в соответствии с настройками по умолчанию блоков **Рулевое управление** (Move Steering) и **Независимое управление моторами** (Move Tank). Аналогично блоки, которые используют датчики, по умолчанию связаны с портами, перечисленными в табл. 3.1. Любой мотор или датчик может работать при подключении к любому порту (например, датчик касания будет работать одинаково хорошо, если подключить его к порту 4 или 1). Однако использование портов по умолчанию облегчает процесс написания программы и снижает вероятность ошибок, поскольку в этом случае не требуется менять настройки порта. Программы в остальной части этой книги предполагают подключение моторов и датчиков в соответствии с табл. 3.1.

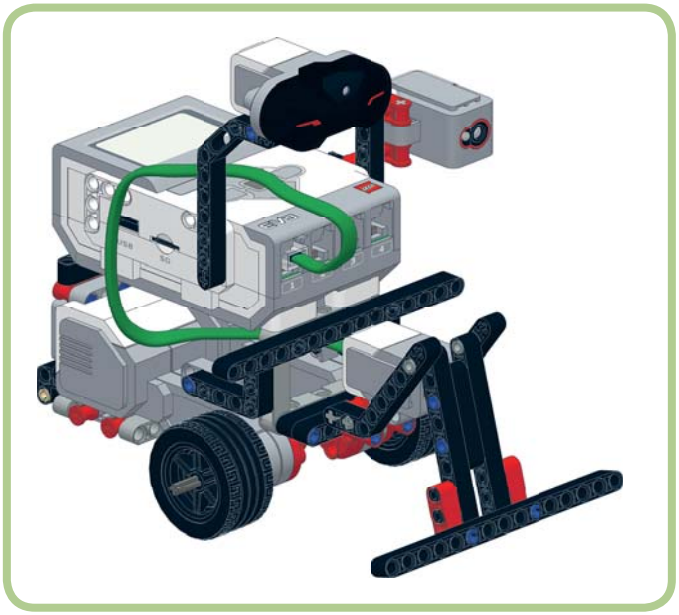
Далее описан процесс подключения кабелей. Для наглядности подключаемый кабель показан на рисунке зеленым; на самом деле все кабели EV3 черные.

Подключение датчика касания

Подключи короткий кабель (25 см) к датчику касания, протяни его через переднюю часть робота и вытяни сбоку.

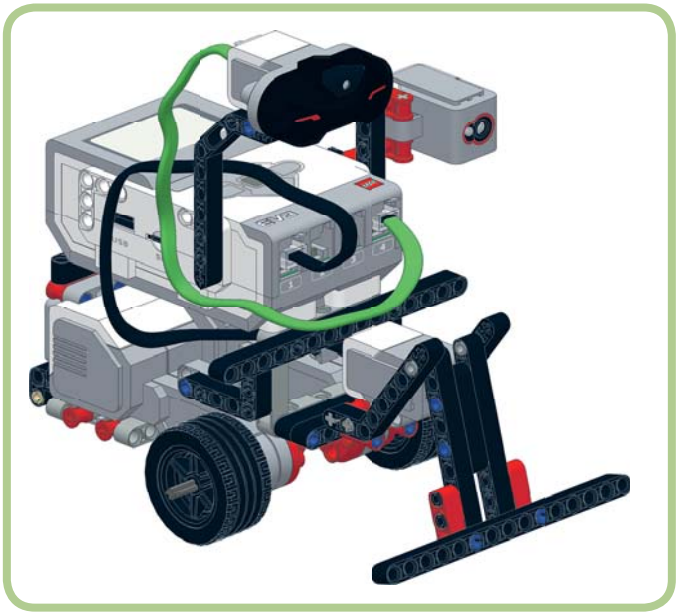


Прикрепи бампер к балке на передней части робота и подключи другой конец кабеля к порту 1.



Подключение инфракрасного или ультразвукового датчика

Используй средний кабель (35 см) для подключения инфракрасного или ультразвукового датчика к порту 4.

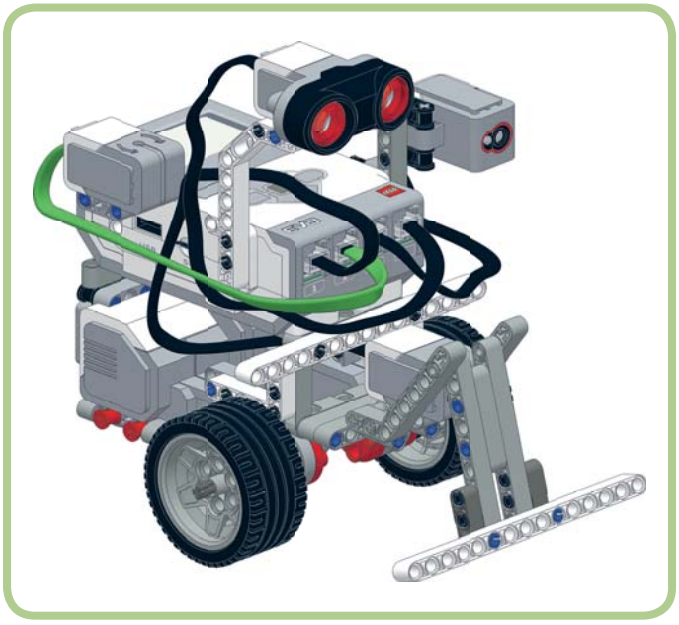


Подключение датчика цвета

Используй короткий кабель (25 см) для подключения датчика цвета к порту 3.

Подключение гироскопического датчика (образовательная версия)

Если ты используешь образовательную версию конструктора, подключи гироскопический датчик к порту 2 с помощью среднего кабеля (35 см).



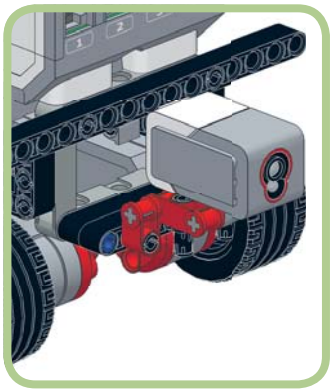
Подключение моторов

Используй два коротких кабеля (25 см) для подключения моторов к портам B и C. Чтобы описанные в этой книге программы работали, кабели должны пересекаться друг с другом. Если ты посмотришь на робота сзади, то увидишь, что мотор справа подключен к порту C, а мотор слева — к порту B.

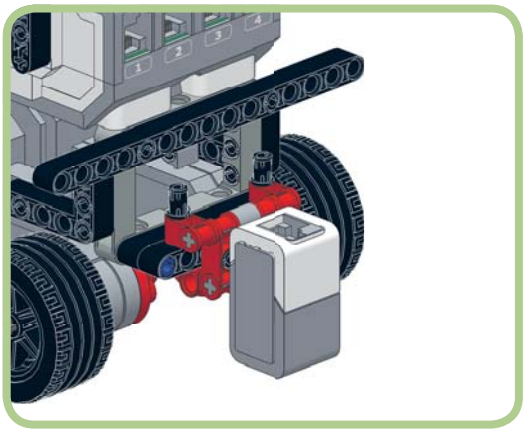
Альтернативное расположение датчика цвета

В некоторых программах требуется, чтобы на передней части робота TriBot находился датчик цвета (вместо датчика касания). В этих случаях установи датчик так, чтобы он был направлен вперед..

Датчик цвета также можно направить вниз для следования вдоль линии. При использовании домашней версии конструктора датчик автоматически оказывается на правильной высоте после его прикрепления к передней части робота TriBot. В образовательной версии шины более высокие, поэтому тебе придется добавить несколько деталей, иначе датчик будет находиться слишком далеко от пола.

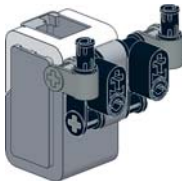


При использовании домашней версии прикрепи датчик цвета так, как показано на изображении ниже.

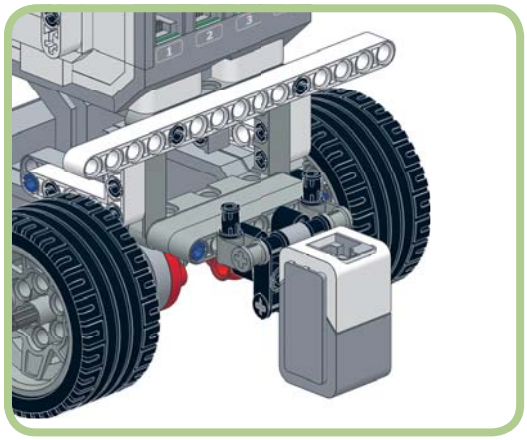


При использовании образовательной версии выполни следующие действия для подключения датчика цвета.

1 



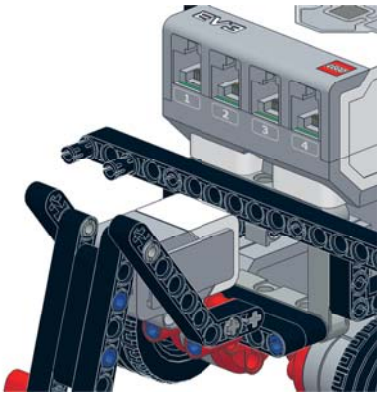
2 

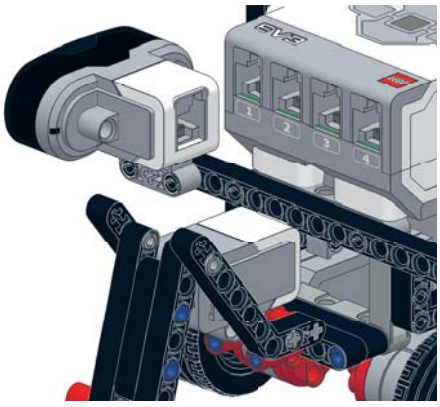


Альтернативное расположение ультразвукового или инфракрасного датчика

Некоторые программы требуют того, чтобы инфракрасный или ультразвуковой датчик был направлен в сторону от робота TriBot. Для этого добавь два штифта к длинной балке, расположенной поперек передней части робота, а затем перемести датчик.

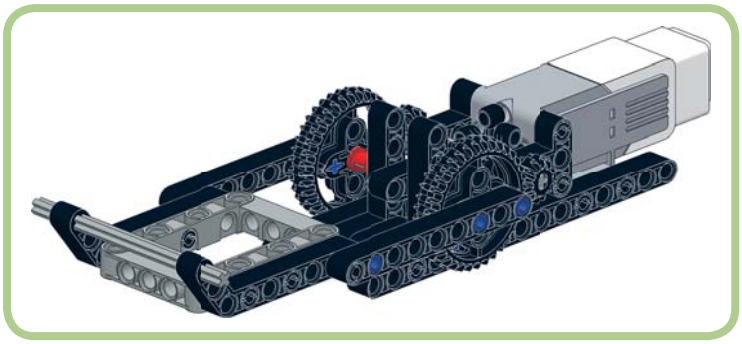
1 



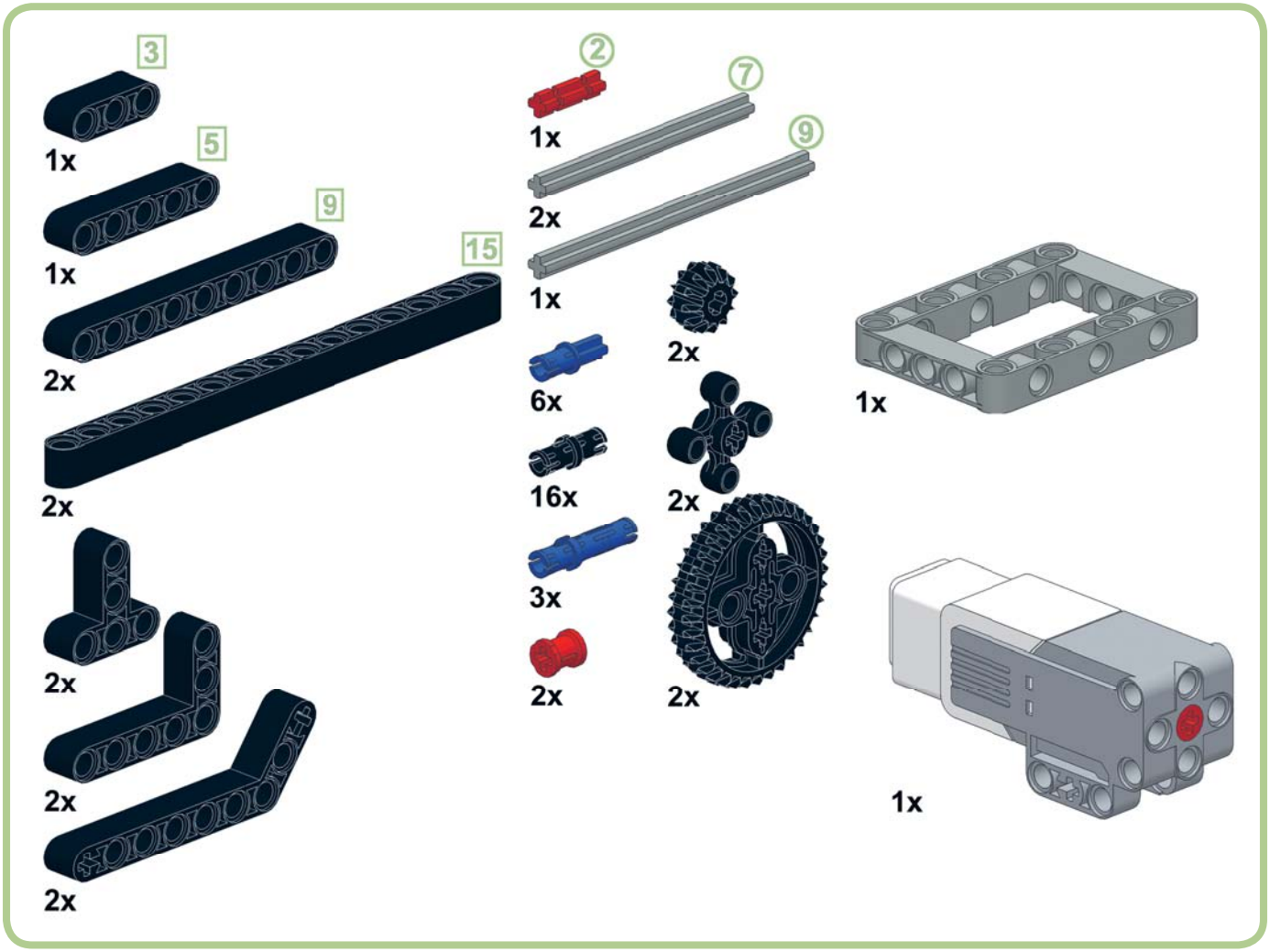
2 

Сборка подъемного рычага

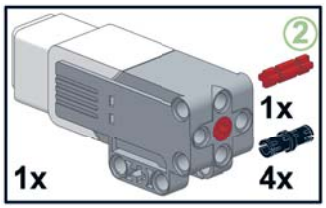
Теперь собери подъемный рычаг, используя средний мотор, как показано ниже:

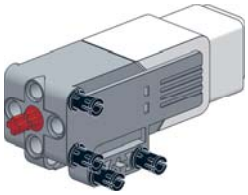


Вот детали, которые тебе понадобятся для сборки подъемного рычага:

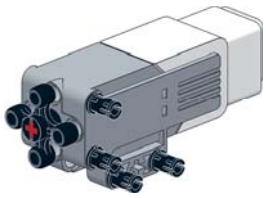


1

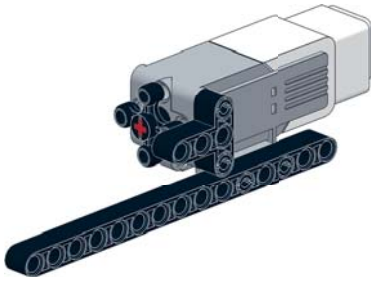




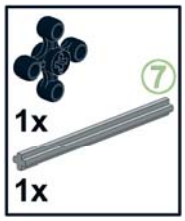
2

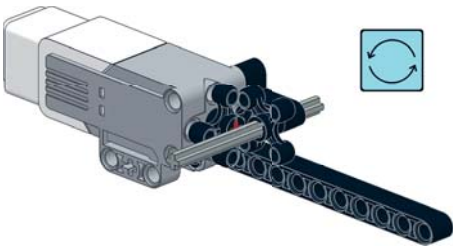


3

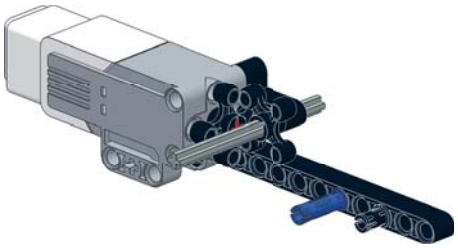


4

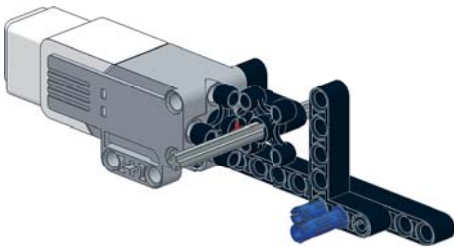




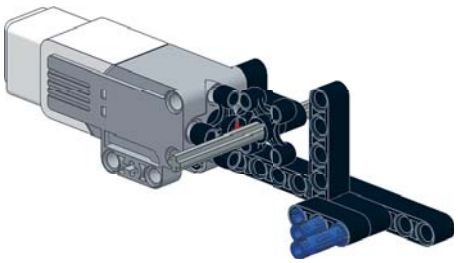
5



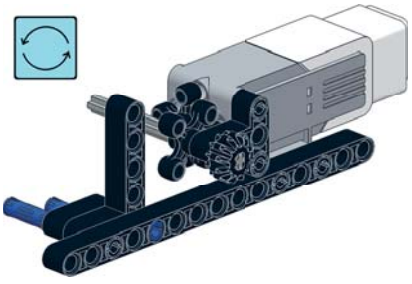
6



7



8

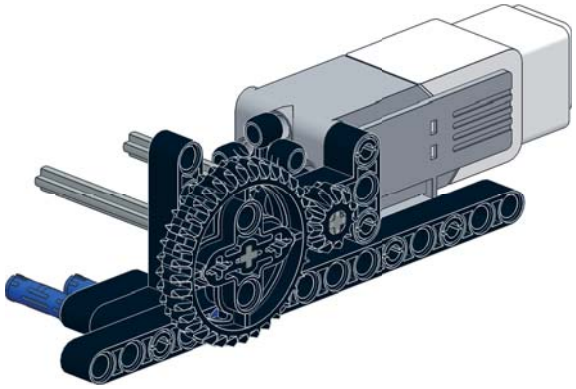


9

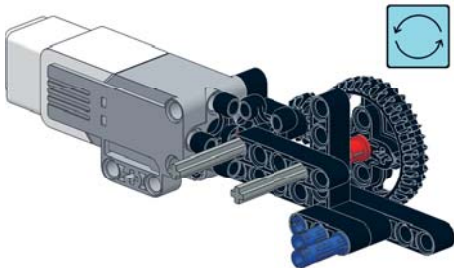
10

На следующем шаге при добавлении зубчатого колеса убедись в том, что два отверстия для осей расположены горизонтально.

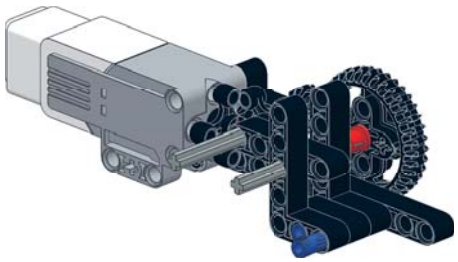
11



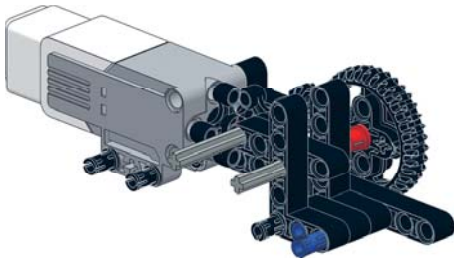
12



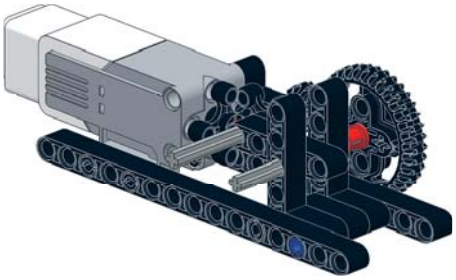
13



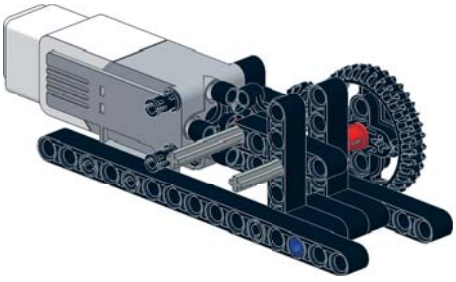
14



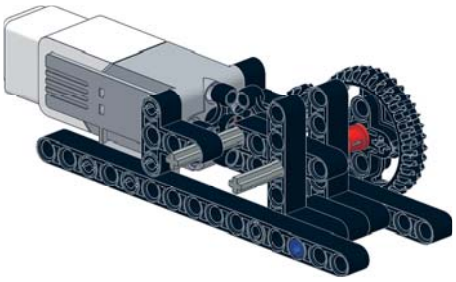
15



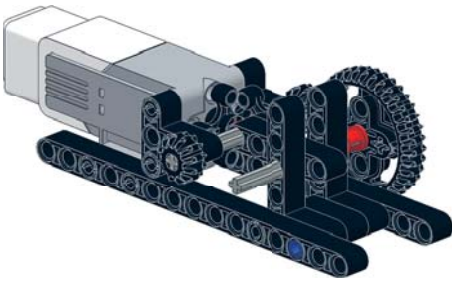
16



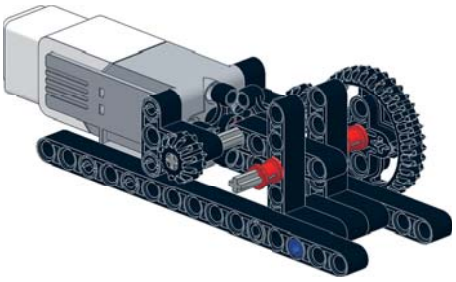
17



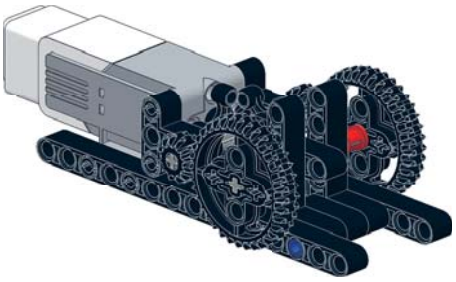
18



19

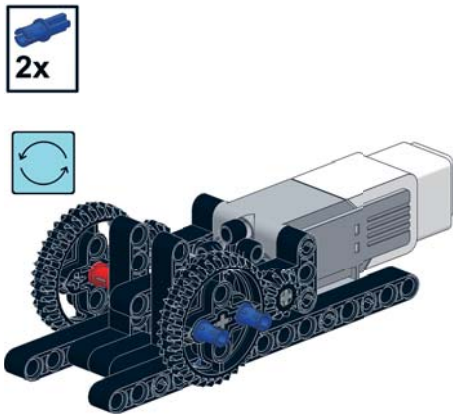


20

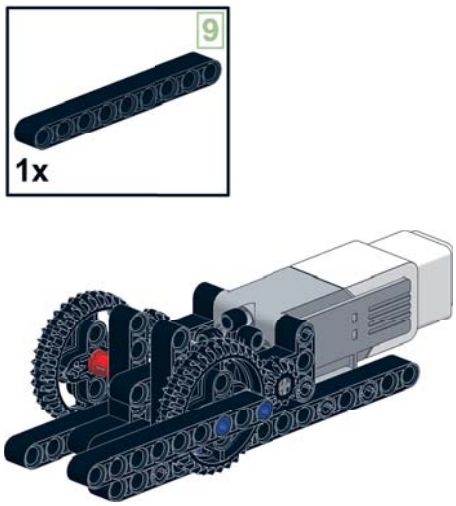


Наконец, прикрепи рычаг к двум большим зубчатым колесам.

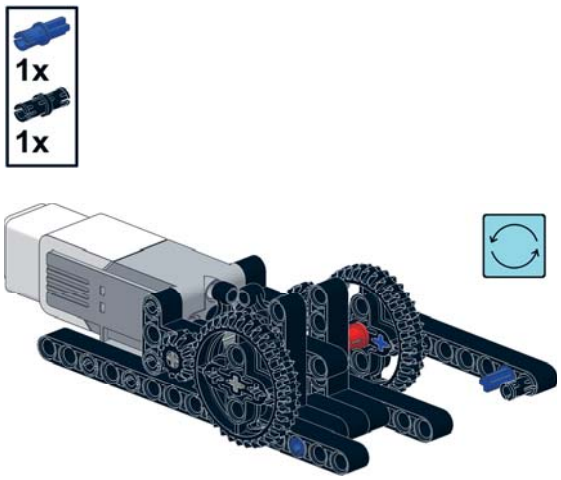
21



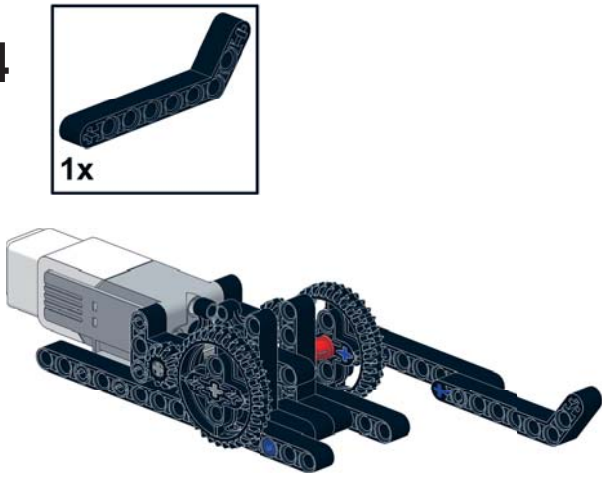
22



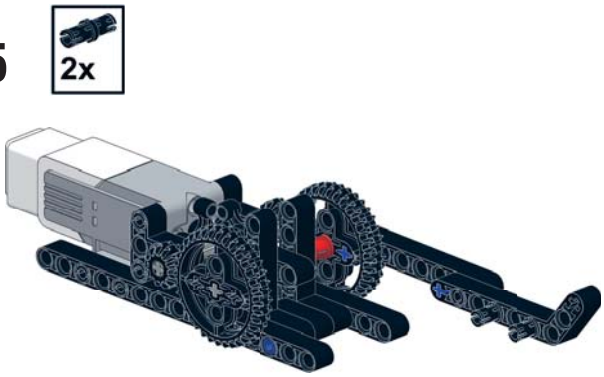
23



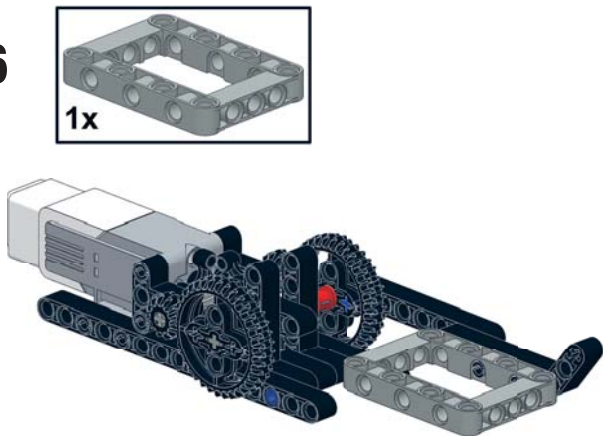
24



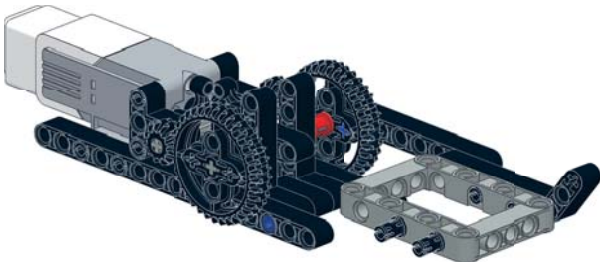
25



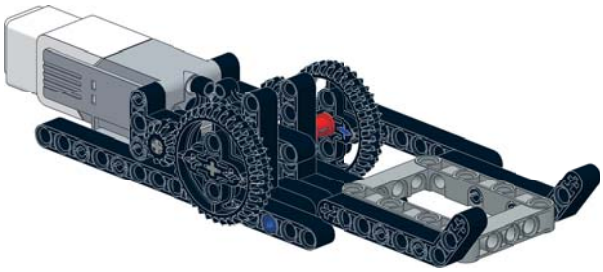
26



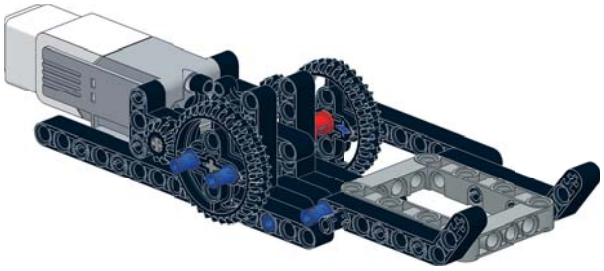
27



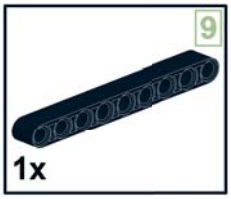
28

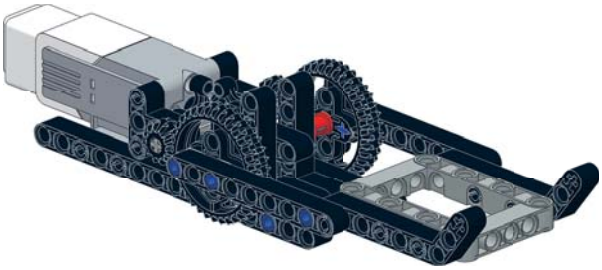


29

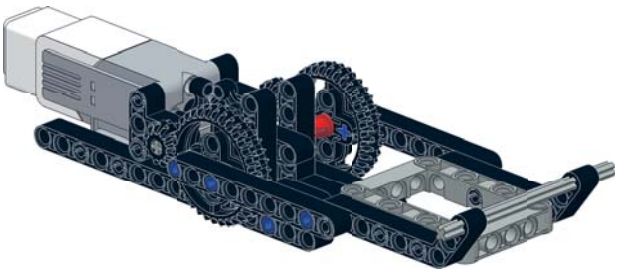


30





31



Заключение

Теперь у тебя есть работающий робот TriBot и подъемный рычаг, которые можно использовать для запуска программ, описанных в следующих главах книги. Первые программы предназначены для робота TriBot в исходной конфигурации с датчиком касания, расположенным спереди. Ты узнаешь, когда датчик нужно будет переместить в одно из альтернативных положений.

4

Движение

Наблюдение за движением своего творения — это одно из самых захватывающих мероприятий при создании робота. Используя набор EV3, можно создавать транспортные средства, роботизированные руки и захватные устройства, средневековые осадные орудия и многие другие движущиеся приспособления. Объединив его с другими наборами LEGO, ты сможешь создать почти неограниченное количество автономных или дистанционно управляемых моделей.

Движение этих устройств обеспечивается моторами EV3. Из этой главы ты узнаешь об этих моторах и о блоках программирования, которые ими управляют. Мы начнем с очень простых программ, постепенно переходя к более сложным.

Моторы EV3

С помощью *большого мотора* (рис. 4.1) легко создать движущегося робота. Его корпус имеет необычную форму, поскольку в дополнении к электрическому мотору он содержит набор шестеренок (зубчатых колес). Эти шестеренки регулируют скорость вращения и мощность мотора, что позволяет подключить колесо непосредственно к мотору без использования дополнительных зубчатых колес.

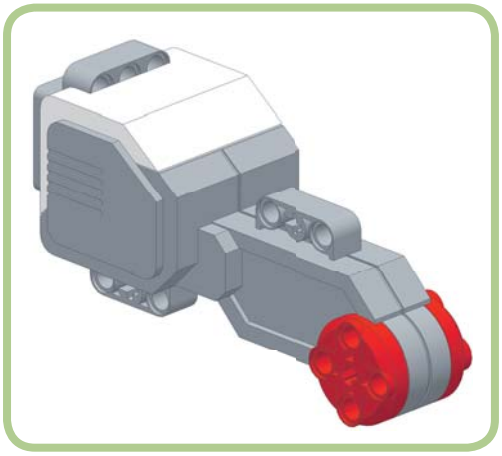


Рис. 4.1. Большой мотор

Средний мотор (рис. 4.2) отличается меньшим размером и более правильной формой, что упрощает его включение в конструкцию робота. Он вращается быстрее, чем большой, и идеально подходит для управления захватами или рычагами. Однако он не такой мощный, как большой мотор, поэтому не так эффективен для обеспечения движения мобильного робота.

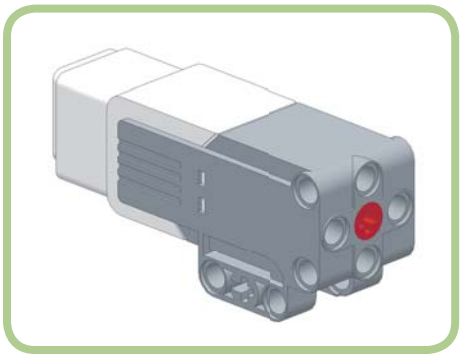


Рис. 4.2. Средний мотор

Каждый мотор EV3 имеет встроенный датчик вращения для измерения скорости вращения и отслеживания количества оборотов мотора. Этот датчик является частью мотора, поэтому для использования его не нужно подключать к одному из четырех портов для датчиков. Блоки, управляющие моторами (описаны далее), автоматически используют этот датчик для выполнения очень точных движений. Кроме того, этот датчик можно применять для управления движением робота. Более полная информация о датчиках, в том числе о датчике вращения мотора, приведена в гл. 5.

Блок «Рулевое управление»

Блок **Рулевое управление** (Move Steering) (рис. 4.3) позволяет управлять двумя большими моторами EV3. Именно этот блок обычно используется для перемещения робота. Он

предусматривает множество параметров и может выполнять широкий спектр задач.

*Рис. 4.3. Блок **Рулевое управление** (Move Steering)*

Блок **Рулевое управление** (Move Steering) автоматически синхронизирует работу двух моторов, постоянно регулируя скорость движения каждого из них, чтобы оба колеса работали одновременно. Этот блок использует датчики вращения мотора, чтобы перемещать робота по прямой линии, поворачивать его или даже вращать на месте в зависимости от значения параметра **Рулевое управление** (Steering).

Например, следующая простая программа перемещает робота TriBot вперед на небольшое расстояние:

1. Создай новый проект под названием *Chapter4*.
2. Измени название программы на *SimpleMove*.
3. Перетащи блок **Рулевое управление** (Move Steering) из палитры с блоками действий на область программирования. Для всех параметров оставь значения по умолчанию. Готовая программа должна выглядеть так, как показано на рис. 4.4.

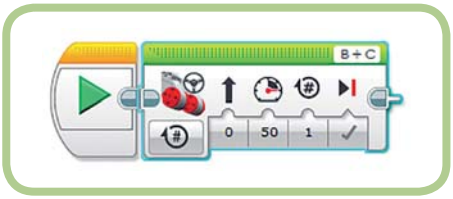


Рис. 4.4. Программа SimpleMove

4. Сохрани проект.
5. Загрузи и запусти программу. Твой робот должен передвинуться вперед на несколько сантиметров.

Значения по умолчанию для блока **Рулевое управление** (Move Steering) подошли для приведенной выше простой программы. Однако ты, вероятно, захочешь настроить его, изменив значения параметров в нижней и верхней части блока. Выбранные значения будут зависеть от конструкции и функций робота.

ПРИМЕЧАНИЕ **Здесь и далее мы не будем включать шаг, предусматривающий сохранение проекта, в описание программ. Однако проекты все равно следует сохранять довольно часто, особенно после внесения значительных изменений, а также перед загрузкой и запуском программы.**

Режим

Ты можешь указать блоку, что он должен делать, с помощью раскрывающегося списка выбора режима (рис. 4.5). Используй три режима **Включить на...** (On for...) для того, чтобы включить моторы на определенное количество оборотов, градусов или секунд перед запуском следующего блока программы. Режим **Включить** (On) включает моторы, после чего программа немедленно запускает следующий блок. Моторы остаются включенными до завершения работы программы или до их выключения другим блоком. Режим **Выключить** (Off) отключает моторы.

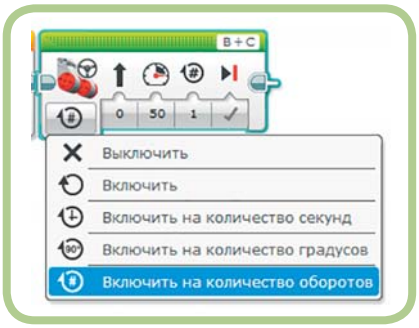


*Рис. 4.5. Раскрывающийся список выбора режима для блока **Рулевое управление** (Move Steering)*

После выбора режима блока отобразятся соответствующие этому режиму параметры. В следующих разделах описываются параметры блока **Рулевое управление** (Move Steering), а также несколько программ. Вместо того чтобы создавать для каждого эксперимента новую программу, создай одну программу и используй ее для каждого нового теста.

1. Создай новую программу и назови ее *Tester*.
2. Добавь в программу блок **Рулевое управление** (Move Steering).

На данном этапе программа *Tester* должна выглядеть так же, как программа *SimpleMove*, показанная на рис. 4.4.

Рулевое управление

Параметр **Рулевое управление** (Steering) (рис. 4.6) определяет направление движения твоего робота. Он может принимать значение от –100° до 100°, которое можно задать путем ввода числа или перемещения ползункового регулятора.

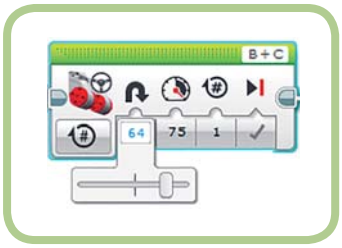


*Рис. 4.6. Параметр **Рулевое управление** (Move Steering)*

Перемести ползунковый регулятор в середину или введи значение **0**, чтобы робот двигался прямо. При этом модуль EV3 *попытается* заставить робота двигаться прямо, постоянно корректируя работу моторов, чтобы они двигались с одинаковой скоростью. На движение робота могут

повлиять многие факторы, а программа может контролировать только скорость движения каждого мотора. Если твой робот *несбалансирован*, т. е. одна его сторона тяжелее другой, то он будет отклоняться в сторону.

ПРИМЕЧАНИЕ **На движение робота будет влиять покрытие поверхности, по которой он движется, а также колеса и тип третьего опорного колеса. Заставить робота двигаться прямо практически невозможно, однако в большинстве ситуаций к этому результату можно максимально приблизиться.**

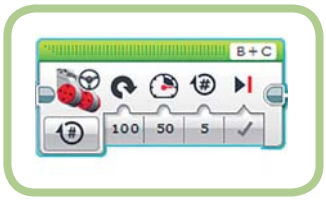
Если установить ползунковый регулятор в крайнее положение или ввести значение –100° или 100°, робот будет вращаться на месте. Это происходит из-за двух моторов, двигающихся с одинаковой скоростью, но в противоположных направлениях. Расстояние между двумя колесами определяет, как долго они должны поворачиваться, чтобы робот совершил полный оборот.

Если установить ползунковый регулятор **Рулевое управление** (Steering) где-то посередине между средним и крайним значением, робот сделает небольшой поворот. Чем ближе значение к –100° или 100°, тем круче будет поворот. Чтобы совершить поворот, модуль EV3 замедляет или останавливает работу одного из моторов. Для совершения очень резкого поворота он заставляет один мотор вращаться назад, а другой — вперед. При выборе режима **Включить на количество градусов** (On for Degrees) или **Включить на количество оборотов** (On for Rotations) заданное значение продолжительности будет относиться к мотору, который работает быстрее, т. е. к мотору, находящемуся с внешней стороны кривой.

При проведении экспериментов с разными значениями параметра **Рулевое управление** (Steering) одного оборота бывает недостаточно для того, чтобы оценить эффект от использования разных значений. Измени значение параметра **Обороты** (Rotations) на 5, чтобы лучше разобраться с движением робота.

1. Измени значение параметра **Обороты** (Rotations) на **5**.

2. Измени значение параметра **Рулевое управление** (Steering) на **100**. Вот как должна выглядеть программа после внесения этих изменений:



Загрузи и запусти программу. Робот TriBot должен вращаться по кругу. Попробуй использовать несколько других значений для параметра **Рулевое управление** (Steering), чтобы понять, как они влияют на движение робота.

Мощность

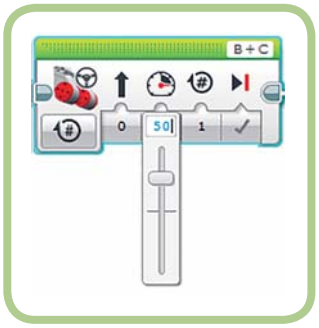


*Рис. 4.7. Параметр **Мощность** (Power)*

Параметр **Мощность** (Power) (рис. 4.7) определяет скорость вращения моторов. Установи значение параметра **Мощность** (Power), переместив ползунковый регулятор или введя значение в диапазоне от –100° до 100°. Положительные значения заставляют моторы вращаться вперед, а отрицательные — назад. При значении 100° моторы вращаются максимально быстро, а при значении 0° — не вращаются вообще.

Внесение следующих изменений заставит робота TriBot двигаться по прямой линии на полной скорости.

1. Установи значение **0**° для параметра **Рулевое управление** (Steering).

2. Установи значение **100**° для параметра **Мощность** (Power).

Загрузи и запусти измененную программу, чтобы оценить скорость движения робота, а потом попробуй использовать другие значения параметра **Мощность** (Power) при нулевом значении параметра **Рулевое управление** (Steering). Затем попробуй изменить значения параметров **Рулевое управление** (Steering) и **Мощность** (Power). Ты заметишь, что робот TriBot может быть немного нестабильным при попытке выполнить резкий поворот на большой скорости.

Продолжительность

При использовании одного из режимов **Включить на...** (On for...) нужно установить значение параметра **Продолжительность** (Duration) в оборотах, градусах или секундах. На рис. 4.8 изображен блок с выбранным режимом **Включить на количество секунд** (On for Seconds) и со значением параметра **Секунды** (Seconds), равным **1**. Обрати внимание на то, что маленький значок параметра **Продолжительность** (Duration) изменяется в соответствии с режимом. Например, в режиме **Включить на количество секунд** (On for Seconds) значок имеет вид небольших часов.

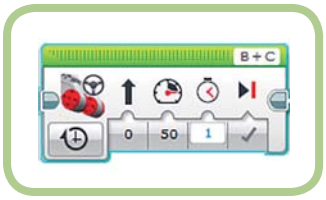


*Рис. 4.8. Параметр **Продолжительность** (Duration) в режиме **Включить на количество секунд** (On for Seconds)*

Связь между оборотами и градусами проста: 360° соответствуют одному обороту мотора. Обычно длительное движение проще измерять в оборотах, а короткое — в градусах, поскольку с небольшими числами легче работать.

При задании количества оборотов или градусов можно использовать отрицательное число, чтобы моторы вращались в обратном направлении. Значение параметра **Секунды** (Seconds) не может быть отрицательным.

Торможение в конце

Параметр **Тормозить в конце** (Brake at End) (рис. 4.9) сообщает блоку, как следует остановить движение и что делать с моторами после выполнения нужного количества оборотов. Первый вариант (галочка) быстро останавливает моторы и фиксирует их на месте. Второй вариант (крестик) позволяет моторам свободно вращаться, пока они не остановятся сами. Эти параметры часто называются **Тормозить** (Brake) и **Двигаться накатом** (Coast) соответственно.

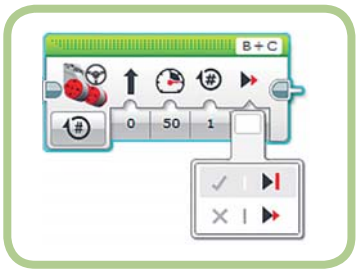


*Рис. 4.9. Параметр **Тормозить в конце** (Brake at End)*

Существующий вариант программы *Tester* в полной мере не демонстрирует эффект выбора параметра **Тормозить** (Brake). После перемещения робота вперед блок останавливает мотор, программа завершается, и модуль EV3 перестает тормозить работу мотора, однако при этом робот TriBot имеет достаточный импульс для того, чтобы продвинуться еще немного вперед. Для оценки эффекта данного параметра добавь блок **Ожидание** (Wait) в режиме **Время** (Time) для приостановки работы программы на две секунды после выполнения блока **Рулевое управление** (Move Steering). Это позволит роботу полностью остановиться до завершения программы.

— Для точной остановки робота используй вариант **Тормозить** (Brake) в режимах **Включить на количество оборотов** (On for Rotations) или **Включить на количество градусов** (On for Degrees). В этом случае мотор должен остановиться очень близко к заданному значению продолжительности. При использовании варианта **Двигаться накатом** (Coast) мотор просто замедляется после достижения значения продолжительности, поэтому перемещается немного дальше установленной цели.
— Используй вариант **Тормозить** (Brake), чтобы мотор не двигался после выполнения блока. Например, если мотор управляет захватом, используй вариант **Тормозить** (Brake), чтобы мотор остановился после захвата объекта; это предотвратит его выскальзывание.
— Торможение мотора расходует заряд аккумулятора, поэтому используй вариант **Тормозить** (Brake), только когда это абсолютно необходимо.

Порт

Параметр **Порт** (Port) (рис. 4.10) можно использовать для того, чтобы сообщить модулю EV3, к каким двум портам подключены моторы. После щелчка по левому или правому порту (см. B и C на рис. 4.10) появится раскрывающийся список, в котором можно выбрать порт. Верхний вариант, отмеченный небольшим черным блоком, предназначен для использования шин данных, о которых мы поговорим в гл. 8.

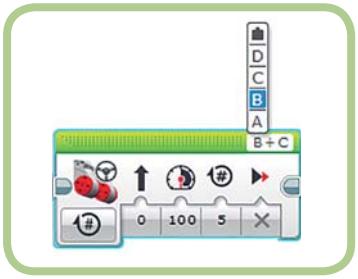


*Рис. 4.10. Параметр **Порт** (Port)*

По умолчанию блок **Рулевое управление** (Move Steering) выбирает порт B для левого мотора и порт C — для правого. Если ты выберешь неправильные порты и поменяешь местами левый и правый моторы, параметр **Рулевое управление** (Steering) заставит робота двигаться в противоположном направлении.

Программное обеспечение EV3 предусматривает функцию Auto-ID, которая сообщает программе о том, какие моторы и датчики подключены к модулю EV3. Кроме того, она может изменить порт, который используется по умолчанию. Например, при добавлении в программу блока **Средний мотор** (Medium Motor) блок **Рулевое управление** (Move Steering) обычно выбирает порт A. Однако, если средний мотор подключен к другому порту при добавлении блока **Средний мотор** (Medium Motor), блок будет использовать этот порт по умолчанию (пока модуль EV3 подключен к программному обеспечению).

Представление порта

При использовании режимов **Включить на количество оборотов** (On for Rotations) или **Включить на количество градусов** (On for Degrees) блока **Рулевое управление** (Move Steering) установка параметра **Продолжительность** (Duration) и определение подходящего значения занимает очень много времени. Вкладка **Представление порта** (Port View) (рис. 4.11) может оказаться довольно полезной, так как в ней показано, сколько оборотов совершил каждый из моторов. Чтобы открыть вкладку **Представление порта** (Port View), выбери среднюю вкладку на странице аппаратных средств. Для того чтобы в этой вкладке отобразилось что-нибудь полезное, среда MINDSTORMS должна быть подключена к модулю EV3.

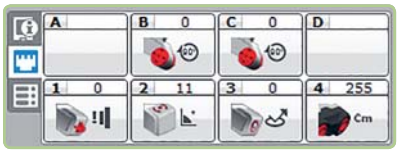


*Рис. 4.11. Вкладка **Представление порта** (Port View)*

Во вкладке **Представление порта** (Port View) указано, к каким портам подключены моторы и датчики, а также тип мотора или датчика. Значение над изображением мотора поступает от его датчика вращения и соответствует общему количеству совершенных им оборотов. Например, если мотор совершит оборот на 360° вперед, а затем на 360° назад, то это значение будет равно 0°. По умолчанию для мотора указано значение в градусах. Щелкни по мотору, чтобы отобразить раскрывающийся список, в котором можно выбрать значение в оборотах. Подробнее вкладка **Представление порта** (Port View) рассмотрена в гл. 5, посвященной датчикам.

Если модуль EV3 не выполняет программу, ты можешь вручную вращать мотор, чтобы узнать, на сколько градусов он повернулся. Это особенно полезный способ для определения того, насколько следует переместить рычаг или захват. Мотор можно вращать вручную, чтобы определить нужное расстояние, а затем ввести полученное значение в один из блоков **Рулевое управление** (Move Steering). Чтобы обнулить значение, щелкни по букве порта (A, B, C или D).

При запуске программы значения обнуляются, а затем обновляются по мере выполнения программы. Ты можешь добавить блоки **Ожидание** (Wait), чтобы приостановить программу и проверить значения на разных этапах ее выполнения.

Приложение Port View модуля EV3

У вкладки **Представление порта** (Port View) есть два недостатка: увидеть ее можно только на экране компьютера; кроме того, модуль EV3 должен быть все время подключен к компьютеру. Вместо нее для определения количества оборотов мотора можно использовать меню модуля EV3. Для этого щелкни по третьей основной вкладке (вверху экрана), а затем выбери пункт **Port View**. Выбери один из портов, и при вращении подключенного к этому порту мотора на экране модуля EV3 отобразится количество совершенных им оборотов.

Программа ThereAndBack

Создадим пару программ с использованием блока **Рулевое управление** (Move Steering). Программа *ThereAndBack* заставит робота TriBot передвинуться вперед на один метр, развернуться, а затем вернуться в исходное положение. Для решения этой задачи необходимо произвести некоторые измерения.

В этой программе используются три блока **Рулевое управление** (Move Steering): один для движения вперед, один для разворота и один для возвращения робота в исходное положение.

Движение вперед

Первый блок должен переместить робота вперед на один метр, однако среди вариантов параметра **Продолжительность** (Duration) метров нет. Как узнать, какому количеству градусов или оборотов соответствует один метр? Начнем с оборотов.

Один из способов заключается в написании программы, которая перемещает робота на большое расстояние, скажем, на 10 оборотов мотора. Перед запуском программы отметь начальную позицию своего робота на полу (например, с помощью кусочка скотча), запусти программу и измерь пройденное роботом расстояние в сантиметрах. Раздели это расстояние на количество оборотов, чтобы определить, насколько передвигается робот за один оборот мотора. Например, мой робот перемещается на 130 см за 10 оборотов или на 13 см за один оборот. Роботу нужно передвинуться на 100 см, поэтому я делю 100 на 13, чтобы найти значение параметра **Продолжительность** (Duration), необходимое для перемещения робота на один метр. Начни с этого расчетного значения и постепенно корректируй его, пока не получишь то, что нужно.

Имей в виду, что это значение будет слегка варьироваться при изменении типа поверхности, по которой движется робот, используемых колес, а также параметра **Мощность** (Power). Мне подошло значение продолжительности в 7,7 оборота при значении параметра **Мощность** (Power), равном 50.

ПРИМЕЧАНИЕ **В образовательной версии конструктора шины больше, чем в домашней, поэтому они перемещают робота на большее расстояние за один оборот мотора. При использовании этих шин и параметра Продолжительность (Duration), равного 5,9, робот переместится на 100 см.**

После определения значения параметра **Продолжительность** (Duration) можно начинать писать программу. Для этого сначала:

1. Создай новую программу под названием *ThereAndBack*.
2. Добавь блок **Рулевое управление** (Move Steering) рядом с блоком **Начало** (Start). По умолчанию будет выбран режим **Включить на количество оборотов** (On for Rotations).
3. Задай вычисленное значение для параметра **Обороты** (Rotations).

На рис. 4.12 показано, как выглядит программа на данном этапе.

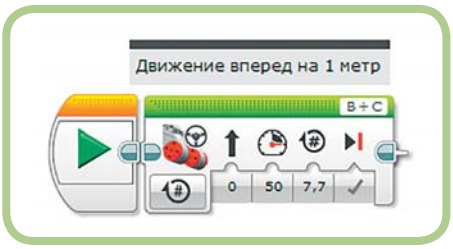


Рис. 4.12. *Шаг 1 этапа создания программы ThereAndBack*

Протестируй эти параметры, запустив программу несколько раз, чтобы робот прошел отмеренное расстояние. Робот должен каждый раз останавливаться очень близко к одному и тому же месту. Если это не так, попробуй уменьшить значение параметра **Мощность** (Power). Скорректируй значение параметра **Продолжительность** (Duration), если робот останавливается в одном и том же месте, но не перемещается на нужное расстояние.

Разворот

Второй блок **Рулевое управление** (Move Steering) развернет робота для движения в обратном направлении. Передвинь ползунковый регулятор **Рулевое управление** (Steering) до конца в одну сторону, чтобы заставить робота TriBot развернуться. Ты можешь выбрать любую сторону — направление не имеет значения.

Сложность настройки этого блока заключается в определении значения продолжительности. После нескольких попыток обнаружилось, что подходит значение 425° (325° при использовании шин из образовательной версии конструктора). Для более точного выполнения разворота пришлось уменьшить значение параметра **Мощность** (Power) до 40, поскольку при значении 50 робот в половине случаев разворачивался на чуть большее количество градусов, чем было необходимо. Это типичный компромисс между скоростью и точностью.

Далее перечислены шаги для реализации этой части программы:

1. Перетащи блок **Рулевое управление** (Move Steering) и помести его после существующего блока **Рулевое управление** (Move Steering).

2. Выбери режим **Включить на количество градусов** (On for Degrees).

3. Перетащи ползунковый регулятор **Рулевое управление** (Steering) в одну сторону до конца.

4. Задай значение **425°** для параметра **Градусы** (Degrees) и значение **40** для параметра **Мощность** (Power). Эти исходные значения можно будет корректировать по мере необходимости при проведении тестирования.

На рис. 4.13 показано, как должна выглядеть программа после добавления нового блока.

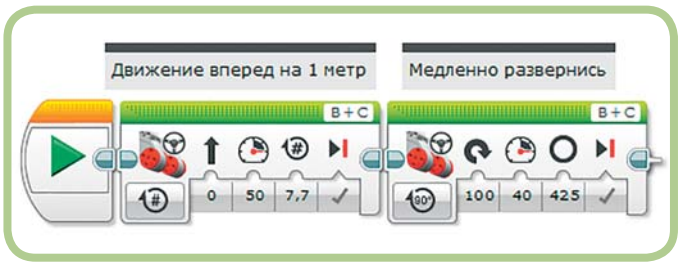


Рис. 4.13. Программа ThereAndBack после добавления второго блока ***Рулевое управление*** *(Move Steering)*

Тестирование отдельного блока

Для точной настройки значений **Градусы** (Degrees) и **Мощность** (Power) легче несколько раз протестировать блок сам по себе и не ждать, пока робот проедет один метр, чтобы посмотреть, правильно ли он разворачивается. Кнопка **Запустить выбранное** (Run Selected) (рис. 4.14) позволяет выполнить отдельный блок или группу блоков. Если ты выберешь второй блок **Рулевое управление** (Move Steering) и затем нажмешь кнопку **Запустить выбранное** (Run Selected), то робот должен будет только развернуться (вместо того, чтобы передвинуться на один метр). Корректируй значения параметров **Градусы** (Degrees) и **Мощность** (Power) по мере необходимости до тех пор, пока робот не будет разворачиваться так, как нужно.

Рис. 4.14. *Кнопка* **Запустить выбранное** *(Run Selected)*

Возвращение в исходное положение

Для возвращения робота TriBot в исходное положение добавь третий блок **Рулевое управление** (Move Steering), перемещающий робота на такое же расстояние, как и первый. Вот заключительные этапы написания программы:

1. Помести блок **Рулевое управление** (Move Steering) в конце программы.

2. Для параметров **Обороты** (Rotations) и **Мощность** (Power) задай такие же значения, как и для первого блока **Рулевое управление** (Move Steering).

На рис. 4.15 показана итоговая версия программа.

Рис. 4.15. Итоговая версия программы ThereAndBack

Протестируй готовую программу. Небольшая ошибка в значении продолжительности для второго блока может появиться после того, как робот TriBot преодолеет один метр в противоположном направлении. По этой причине тебе может потребоваться скорректировать значение параметра **Мощность** (Power) или **Градусы** (Degrees) для второго блока **Рулевое управление** (Move Steering). Когда все будет работать хорошо, попробуй увеличить скорость для определения значения, при котором результат выполнения программы начинает ухудшаться. При слишком большой скорости ты заметишь, что робот начинает поворачиваться слишком далеко или не может вернуться в исходное положение. Кроме того, при слишком быстром движении колеса, как правило, сильнее проскальзывают, а такие проблемы, как несбалансированность конструкции робота, могут вызывать более очевидные ошибки.

Программа AroundTheBlock

Программа *AroundTheBlock* заставит робота TriBot описать квадрат и вернуться в исходное положение. В этом примере мы будем использовать квадрат, длина каждой из сторон которого соответствует трем оборотам мотора. На углу робот может двигаться по кривой, чтобы выполнить плавный поворот вместо вращения на месте.

Для описания квадрата робот должен двигаться вдоль его стороны и повернуть на углу, затем двигаться вдоль следующей стороны и повернуть на углу, и так вдоль всех четырех сторон.

Первая сторона и угол квадрата

В первой части программы используются два блока **Рулевое управление** (Move Steering) для того, чтобы заставить робота TriBot двигаться прямо и совершить первый поворот. Для перемещения робота вдоль стороны квадрата просто установи значение продолжительности равным трем оборотам мотора. Чтобы повернуть на первом углу по кривой, установи значение 25 для параметра **Рулевое управление** (Steering).

Теперь необходимо определить значение параметра **Продолжительность** (Duration), которое позволит выполнить точный поворот на углу. Мне подошло значение 2,4 оборота (или 1,8 оборота при использовании шин из образовательной версии конструктора). Твое значение может отличаться из-за таких факторов, как параметр **Рулевое управление** (Steering) и поверхность, по которой перемещается твой робот. Вот что нужно сделать для создания этой части программы:

1. Создай новую программу под названием *AroundTheBlock*.
2. Добавь блок **Рулевое управление** (Move Steering) после блока **Начало** (Start).
3. Задай значение **3** для параметра **Обороты** (Rotations).
4. Добавь программу второй блок **Рулевое управление** (Move Steering).
5. Задай значение **2,4** для параметра **Обороты** (Rotations).
6. Задай значение **25** для параметра **Рулевое управление** (Steering).

На рис. 4.16 показана программа на данном этапе.

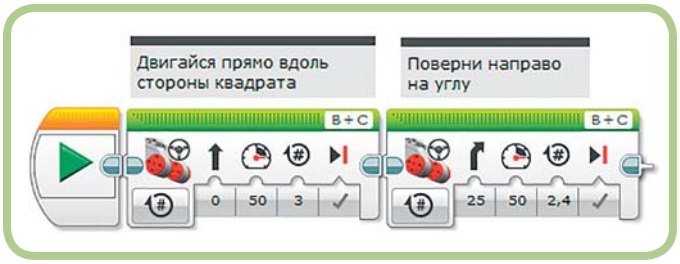


Рис. 4.16. Программа AroundTheBlock заставляет робота на протяжении трех оборотов мотора двигаться прямо и повернуть на углу

Остальные стороны и углы квадрата

*Рис. 4.17. Блок **Цикл** (Loop)*

Теперь расширим программу, чтобы обойти весь квадрат. Для этого можно было бы добавить еще шесть блоков **Рулевое управление** (Move Steering) и использовать те же параметры для остальных трех сторон и углов, но это было бы слишком утомительно. Представь, что роботу требуется описать квадрат 10 раз — в этом случае тебе пришлось бы добавить еще 78 блоков! Более простым способом является использование блока **Цикл** (Loop).

Блок **Цикл** (Loop) (рис. 4.17) позволяет выполнить группу блоков несколько раз. Этот блок находится на вкладке с блоками управления операторами рядом с блоком **Ожидание** (Wait). Ты можешь запустить два блока **Рулевое управление** (Move Steering) четыре раза (по одному разу для каждой из сторон квадрата), поместив их внутри блока **Цикл** (Loop). Подробнее этот блок рассмотрен в гл. 6.

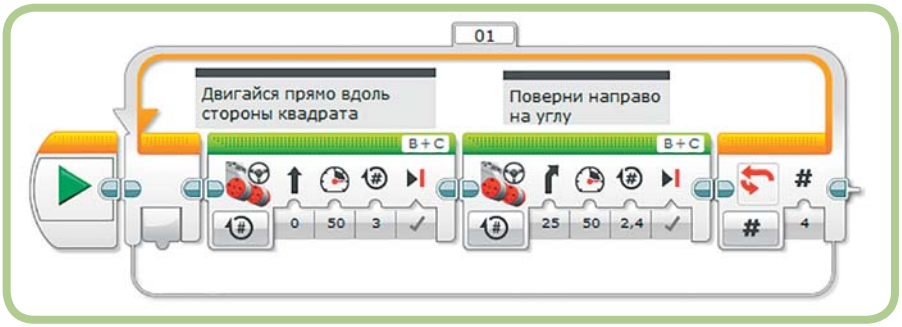
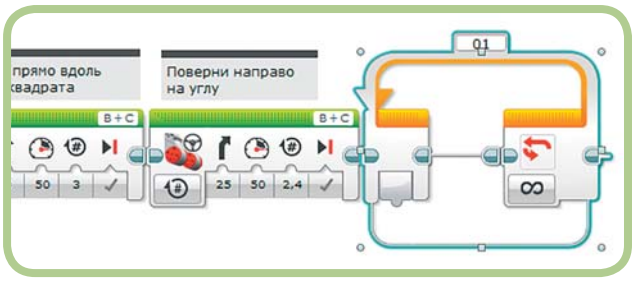
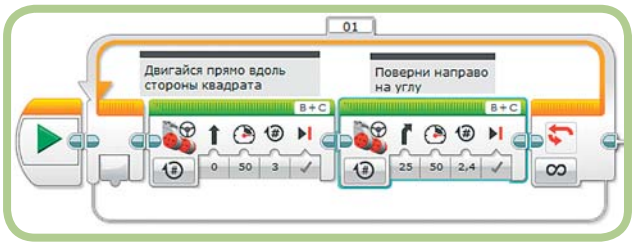


Рис. 4.19. Итоговая версия программы AroundTheBlock

7. Перетащи блок **Цикл** (Loop) в конец программы. Твоя программа должна выглядеть так:



8. Перетащи два блока **Рулевое управление** (Move Steering) в середину блока **Цикл** (Loop). По мере добавления блоков **Рулевое управление** (Move Steering) размер блока **Цикл** (Loop) будет увеличиваться. Теперь программа должна выглядеть так:



ПРИМЕЧАНИЕ **Если над двумя блоками Рулевое управление (Move Steering) были добавлены комментарии, обязательно перемести их так, чтобы они оставались над блоками, к которым они относятся. Комментарии не перемещаются вместе с блоками автоматически, поэтому лучше добавлять их уже после редактирования программы.**

Блок **Цикл** (Loop) предусматривает множество различных режимов, включая один или несколько для каждого типа датчиков. По умолчанию блок **Цикл** (Loop) добавляется в программу с выбранным режимом **Неограниченный** (Unlimited), который обозначен символом бесконечности в правом нижнем углу блока (рис. 4.18). В этой программе нужно изменить режим на **Подсчет** (Count), позволяющий настроить блок на четырехкратное повторение.

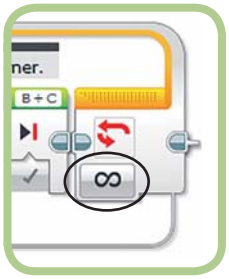


*Рис. 4.18. Раскрывающийся список выбора режима блока **Цикл** (Loop)*

Выполни следующие действия, чтобы цикл выполнялся четыре раза:

9. Щелкни по раскрывающемуся списку выбора режимов и выбери режим **Подсчет** (Count). Справа от меню режимов появится поле для ввода количества повторений цикла.

10. Введи значение **4** для параметра **Подсчет** (Count).

На рис. 4.19 показана итоговая версия программы.

Тестирование программы

После запуска программы робот TriBot должен описать полный квадрат. Если по окончании он не окажется в исходном положении, откорректируй значение продолжительности для второго блока **Рулевое управление** (Move Steering). Этот «поворотный» блок выполняется четыре раза, поэтому по мере движения робота по квадратной траектории ошибки будут накапливаться. Цель состоит в минимизации погрешности, чтобы она не оказывала слишком большого влияния на программу, в данном случае это означает, что робот должен подойти достаточно близко к исходной позиции.

Блок «Независимое управление моторами»

Блок **Независимое управление моторами** (Move Tank) (рис. 4.20) похож на блок **Рулевое управление** (Move Steering) за исключением того, что он предусматривает параметр **Мощность** (Power) для каждого мотора.

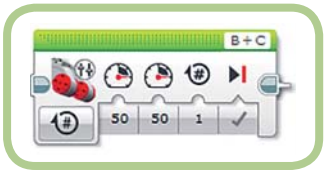


*Рис. 4.20. Блок **Независимое управление моторами** (Move Tank)*

Направление движения робота зависит от значений двух параметров **Мощность** (Power). Если эти значения одинаковые, робот будет двигаться прямо. Если значения различаются, робот повернется, а резкость поворота будет зависеть от разницы значений. Если значения противоположны друг другу (одно положительное, а другое отрицательное), робот будет вращаться на месте.

Как было показано ранее, аналогичного результата можно достичь, используя блок **Рулевое управление** (Move Steering). Однако благодаря явной установке значений параметра **Мощность** (Power) для каждого мотора блок **Независимое управление моторами** (Move Tank) обеспечивает несколько больший контроль. Например, при использовании блока **Рулевое управление** (Move Steering) со значением параметра **Мощность** (Power), равным 50 для поворота за угол, внешний мотор продолжает вращаться с мощностью, равной 50, а вращение другого мотора замедляется. Однако ты можешь использовать блок **Независимое управление моторами** (Move Tank), чтобы ускорить вращение мотора на внешней стороне кривой или замедлить вращение одного мотора и ускорить вращение второго.

ПРИМЕЧАНИЕ **Принцип работы блока Рулевое управление (Move Steering) основан на управлении автомобилем: параметр Мощность (Power) — это акселератор, а параметр Рулевое управление (Steering) — рулевое колесо. Принцип работы блока Независимое управление моторами (Move Tank) основан на управлении транспортными средствами на гусеничном ходу: танков и бульдозеров. Блок Независимое управление моторами (Move Tank) следует применять в программах для роботов, которые вместо колес используют гусеницы.**

Блоки «Большой мотор» и «Средний мотор»

Используй блок **Большой мотор** (Large Motor) или **Средний мотор** (Medium Motor) (рис. 4.21 и 4.22), когда тебе требуется обеспечить работу только одного мотора. Эти два блока отличаются типом мотора, которым они управляют, а также портом, выбранным по умолчанию. Для блока **Большой мотор** (Large Motor) по умолчанию выбран порт D, а для блока **Средний мотор** (Medium Motor) — порт A. Режимы и другие параметры этих блоков идентичны режимам и параметрам блока **Рулевое управление** (Move Steering) за исключением того, что эти блоки не предусматривают параметра **Рулевое управление** (Steering), поскольку управляют только одним мотором.

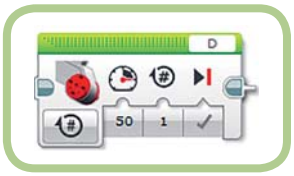


*Рис. 4.21. Блок **Большой мотор** (Large Motor)*

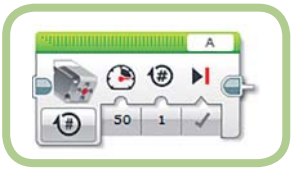


*Рис. 4.22. Блок **Средний мотор** (Medium Motor)*

Подъемный рычаг

Давай поэкспериментируем с блоком **Средний мотор** (Medium Motor), используя подъемный рычаг (рис. 4.23). С помощью этого подъемного рычага робот может поднимать балки или блоки, а если перевернуть рычаг, то робот сможет ловить мяч. Немного изменив конструкцию, собранную из зубчатых колес, рычаг можно превратить в маленькую катапульту.

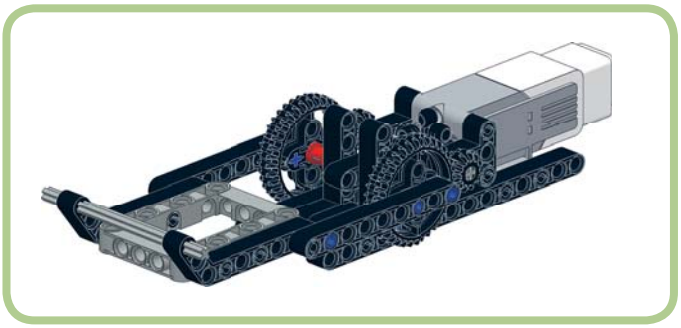


Рис. 4.23. Подъемный рычаг

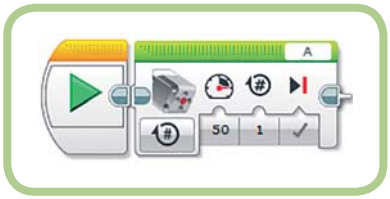
Начни с горизонтально направленного рычага, как показано на рис. 4.23. Чтобы переместить рычаг вручную, осторожно поверни шестерню, прикрепленную к мотору. Если попытаться переместить рычаг напрямую, сопротивление мотора может привести к отделению деталей или проскальзыванию зубчатых колес.

Сначала разберемся с поведением блока **Средний мотор** (Medium Motor) при использовании значений по умолчанию.

1. Создай новую программу под названием *LiftArm*.

2. Подключи средний мотор к порту A модуля EV3, используя длинный кабель (50 см).

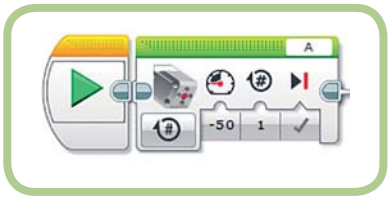
3. Добавь в программу блок **Средний мотор** (Medium Motor). Программа должна выглядеть так:



После запуска программы ты увидишь, что крюк, подсоединенный к крупным шестерням, опускается вниз и прижимается к находящейся под ним поверхности, поднимая мотор и остальную часть механизма. Исходя из результатов этого

теста, ты можешь сделать один важный вывод: положительное значение параметра **Мощность** (Power) опускает рычаг. Вручную верни рычаг в горизонтальное положение, а затем попробуй запустить программу с отрицательным значением параметра **Мощность** (Power). Что произойдет?

4. Измени значение параметра **Мощность** (Power) на **–50**. Теперь программа должна выглядеть так:

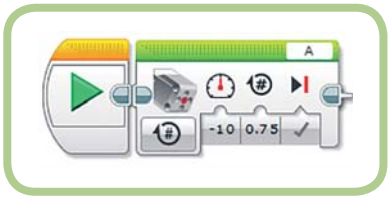


После запуска программы рычаг должен подняться вверх, а затем остановиться чуть дальше точки, в которой он указывает точно вверх. Рычаг должен преодолеть примерно треть круга или 120°.

Блок настроен на один оборот мотора (360°), так почему же рычаг преодолевает только одну треть этого расстояния? Причина заключается в том, что меньшая шестерня имеет 12 зубцов, а большая — 36. Меньшая шестерня совершает три оборота за то время, пока большая совершает один, поэтому ты можешь использовать небольшое значение параметра **Мощность** (Power) в блоке **Средний мотор** (Medium Motor) для обеспечения плавного движения рычага.

Верни рычаг в горизонтальное положение и оцени его движение при небольшом значении параметра **Мощность** (Power).

5. Измени значение параметра **Мощность** (Power) на **–10**, а значение параметра **Обороты** (Rotations) — на **0,75**. Теперь программа должна выглядеть так:



После запуска программы рычаг должен плавно перейти из горизонтального в вертикальное положение.

Блок «Инвертирование мотора»

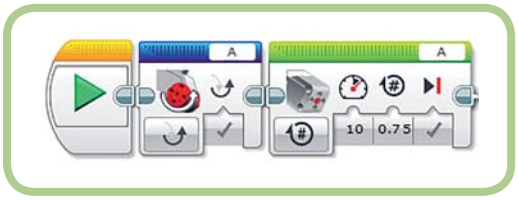
Подъемный рычаг опускается вниз при использовании положительного значения параметра **Мощность** (Power) из-за того, как четырехзубчатая шестерня на моторе соединена с четырехзубчатой шестерней на оси. Интуиция подсказывает, что при использовании положительного значения параметра **Мощность** (Power) рычаг должен подниматься. Подъемный рычаг можно переделать (для этого достаточно переместить шестерню по оси к другой стороне мотора), однако почему бы не воспользоваться блоком для решения этой задачи? Блок **Инвертирование мотора** (Invert Motor) (рис. 4.24) на вкладке с блоками дополнений палитры программирования позволит сделать именно это.

*Рис. 4.24. Блок **Инвертирование мотора** (Invert Motor)*

Блок **Инвертирование мотора** (Invert Motor) изменяет значение параметра **Мощность** (Power) на противоположное для мотора, подключенного к выбранному порту.

6. Добавь блок **Инвертирование мотора** (Invert Motor) между блоком **Начало** (Start) и блоком **Средний мотор** (Medium Motor). Для параметра **Порт** (Port) по умолчанию должно быть задано значение A, поэтому никаких изменений вносить не нужно.

7. Измени значение параметра **Мощность** (Power) в блоке **Средний мотор** (Medium Motor) на **10**. Окончательная версия программы должна выглядеть так:



После запуска программы робот должен вести себя точно так же, как и при запуске предыдущей версии за исключением того, что теперь в ней используется положительное значение параметра **Мощность** (Power).

Проблема при движении накатом

Все блоки управления мотором предусматривают параметр **Тормозить в конце** (Brake at End) с вариантами **Тормозить** (Brake) и **Двигаться накатом** (Coast), которые мы рассмотрели ранее. Когда вариант **Двигаться накатом** (Coast) используется в режиме **Включить на количество оборотов** (On for Rotations) или **Включить на количество градусов** (On for Degrees), эти блоки могут вести себя неожиданным образом. После остановки в результате движения накатом следующее преодоленное роботом расстояние оказывается немного короче или длиннее по сравнению с заданным значением.

Рис. 4.25. Первый этап создания программы CoastTest

При выполнении блока **Рулевое управление** (Move Steering) с такими параметрами *прошивка* отслеживает фактическое количество оборотов мотора (прошивка — это программа, работающая на модуле EV3 и выполняющая написанную программу). При движении накатом мотор останавливается постепенно, поэтому он вращается чуть дольше по сравнению с заданным значением продолжительности, при этом значение продолжительности следующего блока **Рулевое управление** (Move Steering) корректируется с учетом дополнительно пройденного расстояния.

Следующая программа *CoastTest* позволит продемонстрировать этот эффект. Ты будешь использовать два блока **Рулевое управление** (Move Steering) для вращения моторов и вкладку **Представление порта** (Port View), чтобы увидеть количество выполненных ими оборотов. А вот так можно предотвратить такое поведение, если оно создает проблемы для твоей программы.

1. Создай новую программу под названием *CoastTest*.

2. Добавь блок **Рулевое управление** (Move Steering) и оставь все значения по умолчанию.

3. Добавь блок **Ожидание** (Wait) после блока **Рулевое управление** (Move Steering). Задай значение **5** для параметра **Секунды** (Seconds).

4. Добавь еще один блок **Рулевое управление** (Move Steering) в конец программы.

5. Добавь еще один блок **Ожидание** (Wait) в конец программы. Задай значение **5** для параметра **Секунды** (Seconds).

Программа должна выглядеть так, как показано на рис. 4.25.

На данном этапе оба блока **Рулевое управление** (Move Steering) настроены на торможение в конце. Запусти программу и посмотри показания датчиков вращения моторов на вкладке **Представление порта** (Port View). На рис. 4.26 показана вкладка **Представление порта** (Port View) после выполнения первого блока **Рулевое управление** (Move Steering), а на рис. 4.27 — после выполнения второго. Обрати внимание на то, что оба блока сработали так, как ожидалось: каждый из них обеспечил вращение моторов практически на 360°.

*Рис. 4.26. Показания датчиков вращения моторов после выполнения первого блока **Рулевое управление** (Move Steering)*

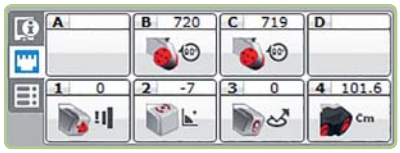
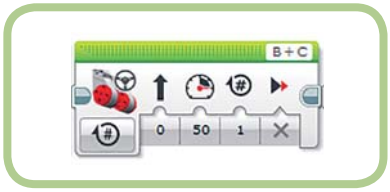


*Рис. 4.27. Показания датчиков вращения моторов после выполнения второго блока **Рулевое управление** (Move Steering)*

ПРИМЕЧАНИЕ Во время работы программы в верхней части исполняемого блока на цветной полосе отображается движущийся узор из белых диагональных полос.

Теперь измени программу, выбрав в первом блоке **Рулевое управление** (Move Steering) вариант **Двигаться накатом** (Coast).

6. Измени значение параметра **Тормозить в конце** (Brake at End) для первого блока **Рулевое управление** (Move Steering) на вариант **Ложь** (False). Теперь блок должен выглядеть следующим образом:



Снова запусти программу и посмотри вкладку **Представление порта** (Port View). На рис. 4.28 и 4.29 показана вкладка **Представление порта** (Port View) после выполнения первого и второго блоков **Рулевое управление** (Move Steering) соответственно.

*Рис. 4.28. После выполнения первого блока **Рулевое управление** (Move Steering) с выбранным вариантом **Двигаться накатом** (Coast)*

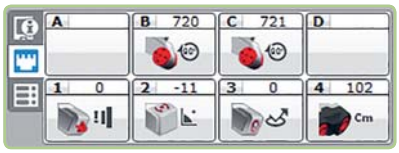


*Рис. 4.29. После выполнения второго блока **Рулевое управление** (Move Steering) с выбранным вариантом **Двигаться накатом** (Coast)*

Первый блок **Рулевое управление** (Move Steering) обеспечил один оборот моторов, после которого робот некоторое время двигался накатом (в данном случае моторы совершили дополнительный оборот на 15–35°). Затем вместо вращения моторов на дополнительные 360°, второй блок **Рулевое управление** (Move Steering) заставил их повернуться так, чтобы общее количество оборотов стало равно двум или 720° (с точностью до одного градуса). Является ли это поведение «правильным» или нет, зависит от того, какая задача поставлена перед программой. Если требуется переместить робота на расстояние, соответствующее 720°, то программа работает нормально. Но что если надо, чтобы второй блок переместил робота на дополнительные 360° независимо от расстояния, на которое уже переместился робот?

Проблема заключается в том, что прошивка отслеживает расстояние, которое робот преодолевает накатом после завершения хода, а затем использует полученную информацию для следующего хода. Для предотвращения этого используй блок **Рулевое управление** (Move Steering) в режиме **Остановка** (Stop) с выбранным для параметра **Тормозить в конце** (Brake at End) вариантом **Тормозить** (Brake).

Помести этот блок непосредственно перед вторым блоком **Рулевое управление** (Move Steering) в программе *CoastTest*. Поскольку робот TriBot не двигается на данном этапе программы, новый блок должен просто устранить смещение, вызванное предыдущим движением накатом.

7. Добавь блок **Рулевое управление** (Move Steering) между первым блоком **Ожидание** (Wait) и вторым блоком **Рулевое управление** (Move Steering).

8. Выбери для нового блока режим **Выключить** (Off).

Теперь программа должна выглядеть так, как показано на рис. 4.30. Запусти программу и посмотри на вкладку **Представление порта** (Port View).

На рис. 4.31 и 4.32 показана вкладка **Представление порта** (Port View) после выполнения первого и второго блоков **Рулевое управление** (Move Steering) соответственно. На этот раз первый блок **Рулевое управление** (Move Steering) заставил моторы вращаться на 360°, после чего робот еще некоторое время двигался накатом. Второй блок **Рулевое управление** (Move Steering) повернул моторы еще на 360°. Добавление нового блока изменило поведение робота так, что последний блок **Рулевое управление** (Move Steering) перемещает его на все указанное расстояние.

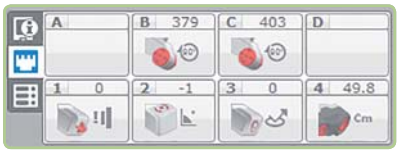


*Рис. 4.31. Показания датчиков вращения моторов после выполнения первого блока **Рулевое управление** (Move Steering)*

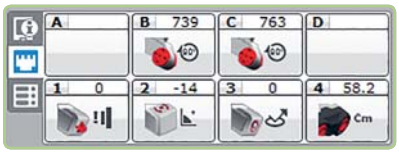


*Рис. 4.32. Показания датчиков вращения моторов после выполнения второго блока **Рулевое управление** (Move Steering)*

Дальнейшее исследование

Попробуй выполнить следующие действия, чтобы еще лучше разобраться с применением представленных в этой главе блоков:

1. Используй вариант **Двигаться накатом** (Coast) в программе *ThereAndBack*. Удалось ли тебе заметить какие-либо изменения в промежуточных передвижениях? Как это влияет на точность возвращения робота TriBot в исходное положение?

2. Измени программу *AroundTheBlock*, используя в ней блоки **Независимое управление моторами** (Move Tank) вместо блоков **Рулевое управление** (Move Steering). Сначала попробуй воспроизвести существующее поворотное движение, а затем поэкспериментируй с различными комбинациями параметров **Мощность** (Power) для двух моторов.

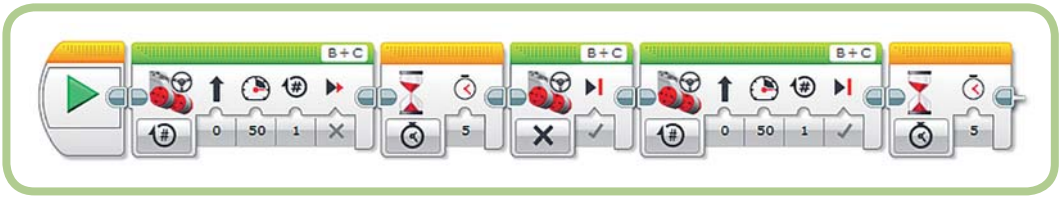


Рис. 4.30. Итоговая версия программы CoastTest

3. Напиши новую программу, использующую тот же метод, что и программа *AroundTheBlock*, для перемещения по треугольнику вместо квадрата.

4. Создай простую полосу препятствий и запрограммируй робота TriBot на ее преодоление, используя последовательность блоков **Рулевое управление** (Move Steering) или **Независимое управление моторами** (Move Tank). Обрати внимание на то, что заставить робота повторять одни и те же действия становится сложнее по мере добавления дополнительных шагов.

Заключение

В набор EV3 входят три мотора, позволяющие легко создавать разнообразных роботов. Программное обеспечение EV3 предусматривает несколько блоков для управления работой моторов, что обеспечивает большую гибкость при принятии решений относительно движения робота. Блок **Рулевое управление** (Move Steering) применяется чаще всего благодаря простоте использования и способности синхронизировать работу двух моторов, что позволяет легко программировать двухколесного робота. Блок **Независимое управление моторами** (Move Tank) предоставляет немного больше контроля над каждым мотором. Блоки **Большой мотор** (Large Motor) или **Средний мотор** (Medium Motor) позволяют управлять одним мотором.

На примерах программ мы изучили несколько разных способов использования этих блоков, в том числе для перемещения робота TriBot по квадратной траектории. Теперь ты можешь запрограммировать своего робота на следование любым курсом, комбинируя блоки. В последующих главах ты еще попрактикуешься в использовании моторов.

5

Датчики

Из этой главы ты узнаешь, как с помощью датчиков EV3 научить робота реагировать на то, что происходит вокруг него. Люди узнают об окружающем мире с помощью пяти чувств (осязание, зрение, слух, обоняние и вкус). Робот использует датчики аналогичным образом для сбора информации об окружающей его среде. С помощью датчиков EV3 робота можно запрограммировать так, чтобы он обходил препятствия, двигался вдоль линии на полу, реагировал на свет, распознавал объекты разных цветов и совершал многие другие действия!

В большинстве случаев программа будет использовать показания датчиков для принятия решений относительно дальнейших действий, однако она также может применять датчики для сбора данных в рамках эксперимента. В этой главе объясняется, как управлять датчиками и использовать их для принятия решений относительно дальнейших действий робота.

ПРИМЕЧАНИЕ **Программное обеспечение EV3 Education поддерживает два типа проектов: программы и эксперименты. Проект эксперимента предназначен для сбора и представления данных. Несмотря на то что этот инструмент отлично подходит для применения в классе, его обсуждение выходит за рамки данной книги. Отмечу, что здесь под словом «*эксперимент*» подразумевается его обычный смысл, а не ссылка на проект эксперимента, предусмотренный в программном обеспечении EV3 Education.**

Использование датчиков

Встроенную поддержку датчиков имеют три блока программирования: **Ожидание** (Wait), **Цикл** (Loop) и **Переключатель** (Switch). Эти блоки можно использовать для того, чтобы заставить программу дождаться наступления какого-либо события, многократно выполнять группу блоков до наступления какого-то события или выбрать подлежащий выполнению блок, исходя из показаний датчика. В этой главе используем все три блока.

На рис. 5.1 изображен раскрывающийся список выбора режимов для блока **Ожидание** (Wait), который предусматривает режимы для всех датчиков EV3, причем некоторые из них могут использоваться более чем одним способом. Например, датчик цвета может измерять либо количество обнаруженного света, либо цвет объекта. В первой программе этой главы (*BumperBot*) блок **Ожидание** (Wait) применяется с датчиком касания для демонстрации принципа его работы.

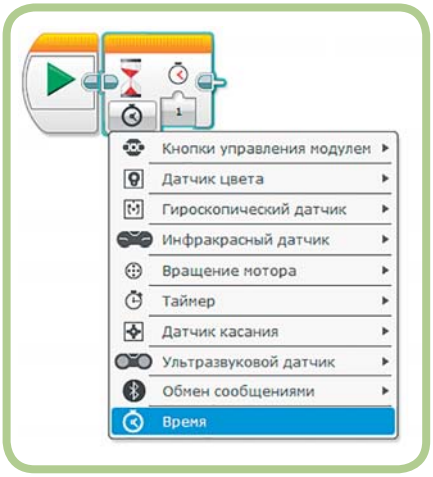


Рис. 5.1. *Раскрывающийся список выбора режимов блока* **Ожидание** *(Wait)*

ПРИМЕЧАНИЕ **Домашняя версия программного обеспечения EV3 не предусматривает параметры для гироскопического или ультразвукового датчика. Если ты приобрел эти датчики отдельно, — загрузи блоки программирования с сайта: www.lego.com.**

Датчик касания

Датчик касания (рис. 5.2) имеет спереди небольшую кнопку. Блоки **Ожидание** (Wait), **Цикл** (Loop) и **Переключатель** (Switch) используют показания этого датчика для определения, нажата ли эта кнопка, отпущена или щелкнута (нажата,

а затем сразу отпущена). Датчик касания часто используется для управления программой или для обнаружения препятствия, с которым столкнулся робот. Например, можно заставить робота дождаться нажатия кнопки датчика касания, прежде чем начинать движение.

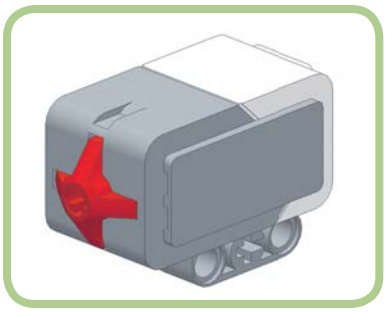


Рис. 5.2. Датчик касания

Датчик касания можно использовать вместе с блоком **Ожидание** (Wait) двумя способами. В режиме **Сравнение** (Compare) ты указываешь состояние кнопки датчика (нажата, отпущена или щелкнута), и блок **Ожидание** (Wait) приостанавливает выполнение программы до тех пор, пока кнопка датчика касания не окажется в указанном состоянии. В режиме **Изменить** (Change) блок **Ожидание** (Wait) проверяет состояние кнопки датчика при запуске блока, а затем ожидает его изменения либо с «нажата» на «отпущена», либо с «отпущена» на «нажата». На рис. 5.3 показаны эти два варианта в раскрывающемся списке выбора режима блока **Ожидание** (Wait).

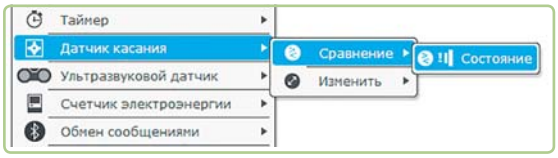


*Рис. 5.3. Режимы датчика касания для блока **Ожидание** (Wait)*

На рис. 5.4 показан блок **Ожидание** (Wait) режимом **Сравнение** (Compare) и параметрами выбора порта для датчика касания.

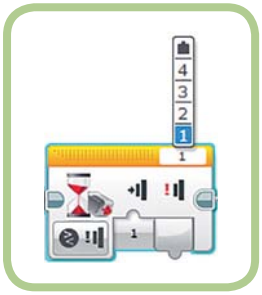


Рис. 5.4. Выбор порта

Для каждого датчика предусмотрен порт по умолчанию. Если модуль EV3 подключен к программному обеспечению, функция Auto-ID предупредит тебя, если выбранный тобой порт не будет совпадать с портом, к которому подключен датчик. В этом случае на экране отобразится специальный значок (рис. 5.5). По возможности следует использовать порты по умолчанию.

Рис. 5.5. Функция Auto-ID предупредит тебя в случае выбора неправильного порта

Другой элемент конфигурации позволяет указать ожидаемое состояние кнопки: «отпущена», «нажата» или «щелкнута», как показано на рис. 5.6.

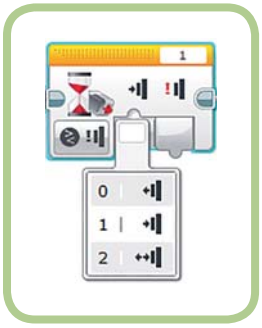


Рис. 5.6. Выбор состояния кнопки датчика касания

Цифры слева от каждого состояния могут быть полезны при использовании шин данных. Значение 0 обозначает, что кнопка «отпущена», 1 — «нажата» и 2 — «щелкнута».

Программа BumperBot

В этом разделе ты создашь программу *BumperBot*, которая будет использовать датчик касания и передний бампер TriBot для того, чтобы заставить робота бродить по комнате. При столкновении робота с препятствием кнопка датчика касания будет нажата, и программа заставит его отъехать назад, развернуться и снова двинуться вперед. Робот будет продолжать движение, пока ты не остановишь его, нажав кнопку **Назад** (Back) на модуле EV3.

Движение вперед

Первая часть программы заставляет робота двигаться прямо, пока он не столкнется с препятствием. Блок **Рулевое управление** (Move Steering) в режиме **Включить** (On) заставляет робота двигаться вперед, а блок **Ожидание** (Wait) использует

датчик касания для сообщения роботу о том, что он с чем-то столкнулся. При нажатии кнопки датчика касания программа должна остановить моторы и заставить робота немного откатиться назад, повернуть в другом направлении, а затем снова двигаться вперед до очередного препятствия. Мы поместим всю программу в блок **Цикл** (Loop), чтобы робот продолжал двигаться до тех пор, пока ты его не остановишь. Выполни следующие действия для создания первой части программы:

1. Создай новый проект под названием *Chapter5*.

2. Создай новую программу под названием *BumperBot*.

3. Перетащи в программу блок **Цикл** (Loop) с вкладки палитры программирования, содержащей блоки управления операторами. Этот блок заставит программу выполняться до тех пор, пока ты ее не остановишь. (По умолчанию для блока **Цикл** (Loop) выбран режим **Неограниченный** (Unlimited), поэтому тебе не нужно ничего менять.)

4. Перетащи в программу блок **Рулевое управление** (Move Steering) и помести его в блок **Цикл** (Loop). Блок **Цикл** (Loop) увеличится, чтобы вместить блок **Рулевое управление** (Move Steering).

5. Выбери для блока режим **Включить** (On) и задай значение **25** для параметра **Мощность** (Power).

Мы устанавливаем для параметра **Мощность** (Power) умеренное значение **25**, чтобы обеспечить работоспособность программы, прежде чем ускорить ее выполнение. Конечно, позже ты сможешь увеличить скорость.

На рис. 5.7 показана программа с блоком **Рулевое управление** (Move Steering), добавленным в блок **Цикл** (Loop).

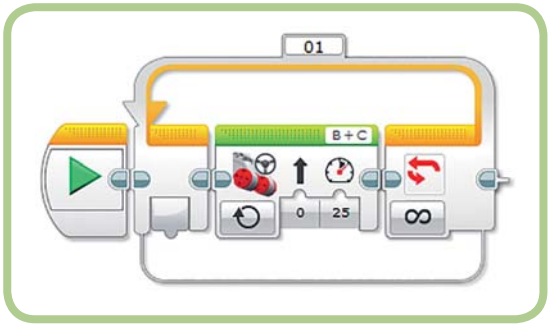


*Рис. 5.7. Программа BumperBot с блоком **Рулевое управление** (Move Steering), добавленным в блок **Цикл** (Loop)*

Обнаружение препятствия

В следующей части программы показания датчика касания используются для остановки моторов при столкновении робота с препятствием (рис. 5.8).

6. Перетащи блок **Ожидание** (Wait) в блок **Цикл** (Loop) и помести его справа от блока **Рулевое управление** (Move Steering).

7. Щелкни по раскрывающемуся списку с режимами и выбери режим **Датчик касания** (Touch Sensor) ⇒ **Сравнение** (Compare). По умолчанию установлен вариант «нажата», поэтому данную настройку менять не нужно.

8. Перетащи еще один блок **Рулевое управление** (Move Steering) в блок **Цикл** (Loop). Выбери для него режим **Выключить** (Off).

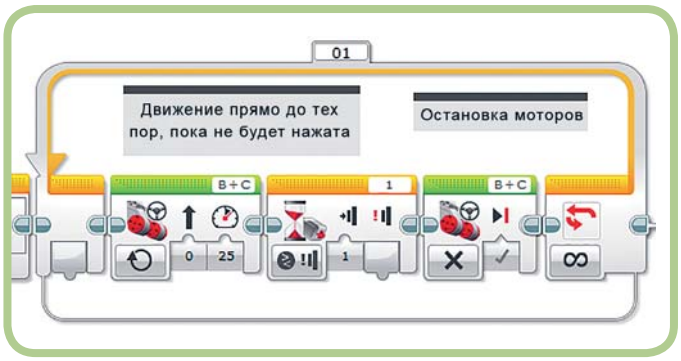


Рис. 5.8. Ожидание нажатия кнопки датчика касания и остановка моторов

Откат и поворот

Теперь мы заставим робота TriBot откатиться назад и повернуть в другом направлении (рис. 5.9).

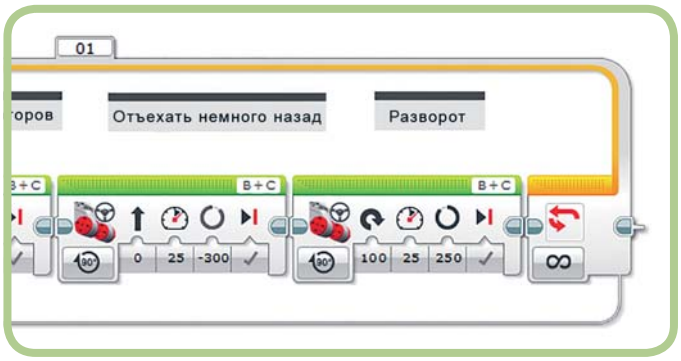


Рис. 5.9. Откат и поворот

9. Добавь еще один блок **Рулевое управление** (Move Steering) в блок **Цикл** (Loop). Выбери для него режим **Включить на количество градусов** (On for Degrees).

10. Задай значение **25** для параметра **Мощность** (Power).

11. Задай для параметра **Продолжительность** (Duration) значение **–300**, чтобы заставить робота откатиться назад.

12. Добавь еще один блок **Рулевое управление** (Move Steering) в блок **Цикл** (Loop). Перетащи ползунковый регулятор до конца вправо или введи значение **100**, чтобы заставить робота повернуться.

13. Выбери режим **Включить на количество градусов** (On for Degrees) и задай значение **25** для параметра **Мощность** (Power).

14. Задай значение **250** для параметра **Продолжительность** (Duration). Ты можешь поэкспериментировать с разными значениями, чтобы посмотреть, как поворот влияет на программу. Вероятно, робот должен повернуться, по крайней мере, на 90°, чтобы отойти от стены с первой попытки.

ПРАКТИКУМ 5.1

Убедившись в том, что программа работает, попробуй увеличить значение параметра **Мощность** (Power) в блоках **Рулевое управление** (Move Steering). Как быстро ты сможешь заставить робота перемещаться по комнате, прежде чем он начнет терять равновесие? Выходит ли его поворот из-под контроля? В какой-то момент? Вероятность возникновения этой проблемы зависит от поверхности, по которой передвигается робот.

ПРАКТИКУМ 5.2

Измени программу *BumperBot*, научив робота TriBot играть в мяч.

Возьми небольшой мягкий мяч и подкати его к бамперу робота. Программа должна дождаться касания мячом бампера, а затем быстро переместить робота вперед на небольшое расстояние, чтобы он оттолкнул мяч обратно к тебе. Затем робот должен вернуться в исходное положение и ожидать твоего следующего броска. Используй мяч, который достаточно большой, чтобы задействовать датчик касания, и при этом достаточно мягкий, чтобы не повредить модуль EV3. Мяч для пинг-понга слишком маленький, а мяч для бейсбола — слишком твердый. Хорошо подойдет теннисный мяч.

Тестирование

Теперь загрузи готовую программу в модуль EV3 и протестируй ее. Помни, что при выполнении программы робот должен двигаться вперед по прямой линии, пока не натолкнется на препятствие, после чего он должен откатиться назад, развернуться и снова начать двигаться вперед. Программа должна продолжать работать, пока ты не остановишь ее, нажав кнопку **Назад** (Back) на модуле EV3. Для нахождения работающей комбинации поэкспериментируй со значением параметра **Продолжительность** (Duration) двух последних блоков **Рулевое управление** (Move Steering), чтобы изменить расстояние, на которое робот откатывается назад и поворачивается.

Датчик цвета

Датчик цвета (рис. 5.10) определяет цвет или яркость света, отраженного от поверхности и попадающего в небольшое окошко на его передней части. Этот датчик можно использовать в трех разных режимах: **Цвет** (Color), **Яркость отраженного света** (Reflected Light Intensity) и **Яркость внешнего освещения** (Ambient Light Intensity). Изучив, как работает каждый режим, мы создадим программы для определения цвета и распознавания линии, чтобы увидеть датчик цвета в действии.

Рис. 5.10. Датчик цвета

Режим «Цвет»

В режиме **Цвет** (Color) этот датчик может определить цвет объекта, находящегося перед ним. Датчик распознает черный, синий, зеленый, желтый, красный, белый и коричневый цвета. Если датчик не может определить цвет находящегося перед ним объекта, он использует значение **Нет цвета** (No Color) или выбирает цвет, наиболее близкий к тому, который он распознал. Например, если поместить перед датчиком оранжевый предмет, он может выдать результат **Красный** (Red), **Желтый** (Yellow) или **Нет цвета** (No Color) в зависимости от оттенка оранжевого. Для точного определения цвета датчик следует помещать очень близко к объекту, но не касаясь его, для уменьшения влияния других источников света.

Как показано на рис. 5.11, блок **Ожидание** (Wait) может использовать датчик цвета в режиме **Сравнение** (Compare) и **Изменить** (Change). В режиме **Изменить** (Change) блок определяет цвет объекта перед датчиком при запуске блока, а затем ожидает изменения цвета.

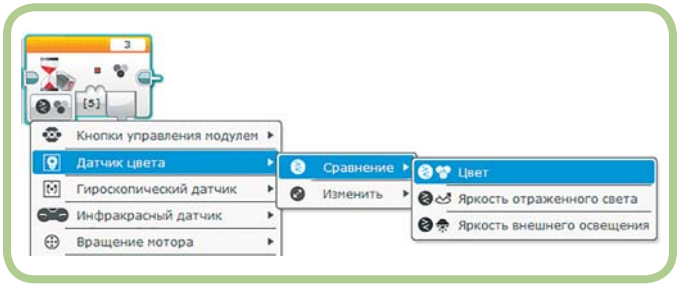


Рис. 5.11. Выбор датчика цвета

В режиме **Сравнение** (Compare) блок ждет, пока датчик не обнаружит цвет, выбранный из списка, как показано на рис. 5.12. Белый квадрат, перечеркнутый красной линией,

соответствует варианту **Нет цвета** (No Color). Квадраты, расположенные над меню с вариантами цветов, показывают, какие цвета ожидает обнаружить блок. Например, блок, изображенный на рис. 5.13, ожидает, пока датчик обнаружит зеленый, синий или красный цвет. (Ты будешь использовать датчик в этом режиме при создании программы *IsItBlue*, далее в этой главе.)

Рис. 5.12. Выбор ожидаемого цвета или цветов

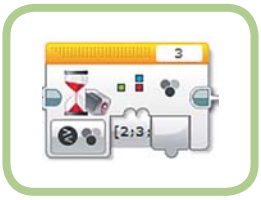


Рис. 5.13. Ожидание зеленого, синего или красного цвета

Режим «Яркость отраженного света»

В режиме **Яркость отраженного света** (Reflected Light Intensity) (рис. 5.14) датчик включает свой красный индикатор и измеряет количество света, отраженного от объекта. Значения варьируются от 0 (очень темно) до 100 (очень яркий свет). Этот режим полезен для следования вдоль линии. Как и при использовании режима **Цвет** (Color) датчик следует расположить близко к объекту, чтобы другие источники света не помешали его работе. В этом режиме нужно настроить такие параметры, как **Тип сравнения** (Compare Type) и **Пороговое значение** (Threshold Value). Параметр **Тип сравнения** (Compare Type) сообщает блоку, как следует сравнивать показания датчика с пороговым значением (показание, вызывающее определенное поведение робота). Для данного параметра предусмотрены следующие варианты: **Равно** (Equal To), **Не равно** (Not Equal To), **Больше чем** (Greater Than), **Больше или равно** (Greater Than or Equal To), **Меньше** (Less Than) и **Меньше или равно** (Less Than or Equal To). Блок, изображенный на рис. 5.14, ожидает, пока показание датчика не станет меньше 50.

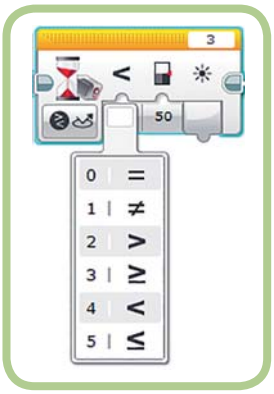


*Рис. 5.14. Настройка параметра **Тип сравнения** (Compare Type)*

Режим **Изменить** (Change) ⇒ **Яркость отраженного света** (Reflected Light Intensity) позволяет настроить степень и направление ожидаемого изменения. Ты можешь заставить программу дождаться увеличения, уменьшения значения или изменения направления. Например, блок, изображенный на рис. 5.15, будет ожидать увеличения или уменьшения значения интенсивности света на 10. Если в момент запуска блока **Ожидание** (Wait) показание датчика было равно 55, выполнение программы будет приостановлено до тех пор, пока значение датчика не превысит 65 или не станет равным или меньшим 45.

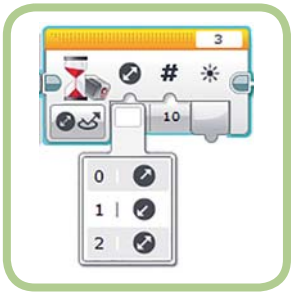


Рис. 5.15. Выбор направления изменения

ПРИМЕЧАНИЕ Может показаться, что для каждого датчика предусмотрено огромное количество параметров, однако многие их режимы (особенно те, которые предполагают измерение числовых значений) настраиваются одинаковым образом.

Режим «Яркость внешнего освещения»

Третий способ использования датчика цвета заключается в измерении *внешнего освещения*, т. е. освещения окружающей робота среды. На рис. 5.16 изображен блок **Ожидание** (Wait) в режиме **Датчик цвета** (Color Sensor) ⇒ **Сравнение** (Compare) ⇒ **Яркость внешнего освещения** (Ambient Light Intensity). При измерении внешнего освещения датчик предусматривает режимы **Сравнение** (Compare) и **Изменить** (Change) с уже рассмотренными ранее настройками.

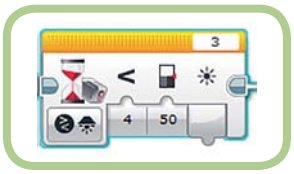


*Рис. 5.16. Режим **Датчик цвета** (Color Sensor) ⇒ **Сравнение** (Compare) ⇒ **Яркость внешнего освещения** (Ambient Light Intensity)*

ПРИМЕЧАНИЕ Определить режим, в котором используется датчик цвета, можно по индикатору на его передней части. В режиме **Цвет** (Color) он будет испускать яркий, разноцветный свет; в режиме **Яркость отраженного света** (Reflected Light Intensity) — ярко-красный свет; а в режиме **Яркость внешнего освещения** (Ambient Light Intensity) — тусклый синий свет. Цвет индикатора датчика позволяет легко выявить такую распространенную ошибку, как выбор неправильного режима.

Вкладка «Представление порта»

Вкладка **Представление порта** (Port View) может отобразить показание датчика, используя любой из поддерживаемых им режимов. Щелкни по изображению датчика на вкладке **Представление порта** (Port View), чтобы выбрать режим. Например, на рис. 5.17 показан раскрывающийся список, появляющийся при щелчке по изображению датчика цвета.

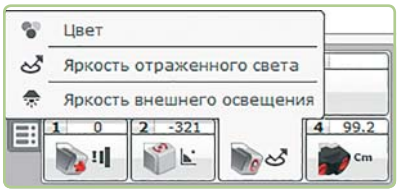


Рис. 5.17. Выбор режима датчика цвета

Значение, полученное от датчика, всегда отображается в виде числа. Это хорошо при использовании датчика цвета в режиме **Яркость отраженного света** (Reflected Light Intensity), но не очень удобно при использовании режима **Цвет** (Color). На рис. 5.18 показана вкладка **Представление порта** (Port View) с выбранным режимом **Цвет** (Color). Обрати внимание на то, что показанием датчика является значение **6**, а не **Белый** (White). Посмотреть, какое число соответствует каждому из цветов, можно в списке выбора цвета блока **Ожидание** (Wait) (см. рис. 5.12).

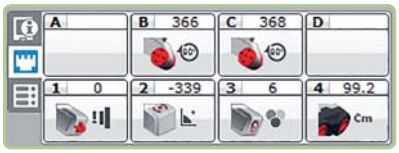


*Рис. 5.18. Вкладка **Представление порта** (Port View) с выбранным режимом **Цвет** (Color)*

Программа IsItBlue

В этом разделе ты создашь программу *IsItBlue*, в которой используется датчик цвета для распознавания синих объектов. После запуска программы робот скажет: “Yes” («Да») или “No” («Нет») в зависимости от цвета объекта, находящегося перед датчиком.

Блок «Переключатель»

Необходимо сделать так, чтобы программа могла выбрать вариант ответа в зависимости от показаний датчика цвета. Для этого мы будем использовать блок **Переключатель** (Switch), который дает программе команду запустить один набор блоков, если датчик распознал синий объект, и другой набор блоков, если не распознал.

Для создания этой программы:

1. Создай новую программу под названием *IsItBlue*.
2. Перетащи блок **Переключатель** (Switch), находящийся на оранжевой вкладке с блоками управления операторами, в область программирования.
3. Выбери режим **Датчик цвета** (Color Sensor) ⇒ **Измерение** (Measure) ⇒ **Цвет** (Color).

ПРАКТИКУМ 5.3

Используй вкладку Представление порта (Port View), чтобы поэкспериментировать с различными режимами датчика цвета. Попытайся почувствовать, как цвет и покрытие поверхности влияют на количество отраженного от нее света. Используй режим Цвет (Color), чтобы увидеть цвета, которые датчик распознает хорошо, и те, с которыми возникают сложности.

На данном этапе блок **Переключатель** (Switch) должен выглядеть так, как показано на рис. 5.19. Черный квадрат в верхней части блока означает, что программа будет запускать блоки в верхней части, если датчик выдаст значение **Черный** (Black). Белый квадрат, перечеркнутый красной линией, — символ того, что блоки в нижней части будут выполнены, если датчик выдаст значение **Нет цвета** (No Color). Если значение датчика будет отличаться от вариантов **Черный** (Black) или **Нет цвета** (No Color), выполнятся блоки в верхней части. При значении Блок **Переключатель** (Switch) по умолчанию будет выбрана группа блоков, отмеченная точкой.

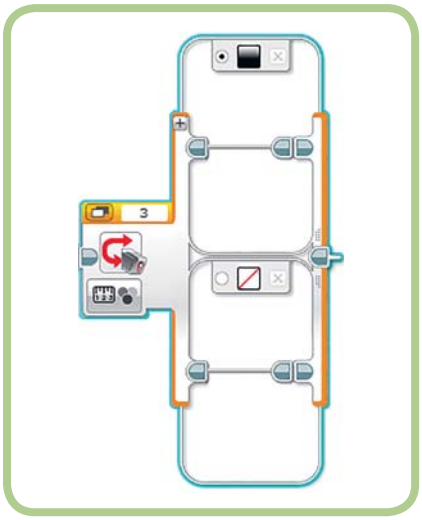


*Рис. 5.19. Режим **Датчик цвета** (Color Sensor) ⇒ **Измерение** (Measure) ⇒ **Цвет** (Color)*

4. Щелкни по квадрату в верхней части нижнего раздела и выбери вариант **Синий** (Blue), как показано на рис. 5.20.

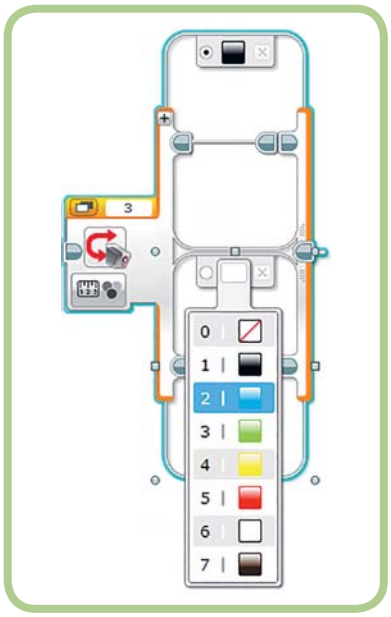


Рис. 5.20. Выбор цвета

5. Перетащи блок **Звук** (Sound) в верхнюю область внутри блока **Переключатель** (Switch).

6. Выбери вариант "**No**" в списке **Звуковые файлы LEGO** (LEGO Sound Files) ⇒ **Связь** (Communications).

7. Перетащи еще один блок **Звук** (Sound) в нижнюю область внутри блока **Переключатель** (Switch).

8. Выбери вариант "Yes" в списке Звуковые файлы LEGO (LEGO Sound Files) ⇒ **Связь** (Communications) для этого блока. Программа должна выглядеть как на рис. 5.21.

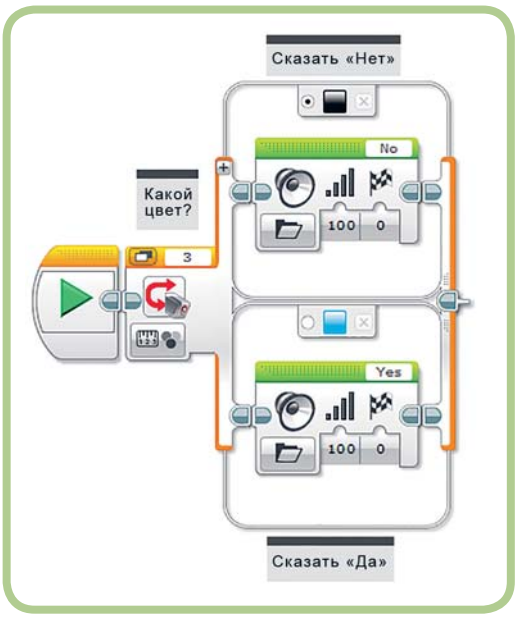


Рис. 5.21. Итоговая версия программы IsItBlue

Перед запуском программы помести тестовый объект перед датчиком цвета. После запуска программы робот должен сообщить, является ли объект синим. Выдав ответ "Yes" или "No", программа завершится, поэтому для тестирования на другом объекте ее нужно будет запустить снова.

Усовершенствование программы

Программа *IsItBlue* работает и в стандартном варианте, однако она не очень удобна в использовании. Ты можешь улучшить ее, добавив способ, позволяющий сообщить роботу о том, что объект, цвет которого требуется определить, находится на месте. Кроме того, можно сделать так, чтобы программа работала непрерывно до тех пор, пока ты ее не остановишь.

Использование датчика касания

Сначала мы добавим блок, использующий датчик касания для того, чтобы сообщить программе, когда придет время для использования датчика цвета.

1. Перетащи блок **Ожидание** (Wait) в программу и помести его слева от блока **Переключатель** (Switch).

2. Выбери режим **Датчик касания** (Touch Sensor) ⇒ **Сравнение** (Compare) ⇒ **Состояние** (State).

3. Выбери состояние **Щелчок** (Bumped), как показано на рис. 5.22.

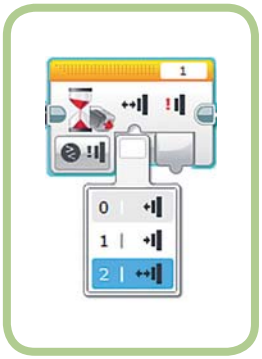


Рис. 5.22. Выбор состояния кнопки датчика касания

Теперь программа должна выглядеть, как показано на рис. 5.23. После запуска программа должна находиться в режиме ожидания до тех пор, пока ты не нажмешь и отпустишь кнопку датчика касания, прежде чем выдать ответ "Yes" или "No", и завершиться после определения цвета одного объекта.

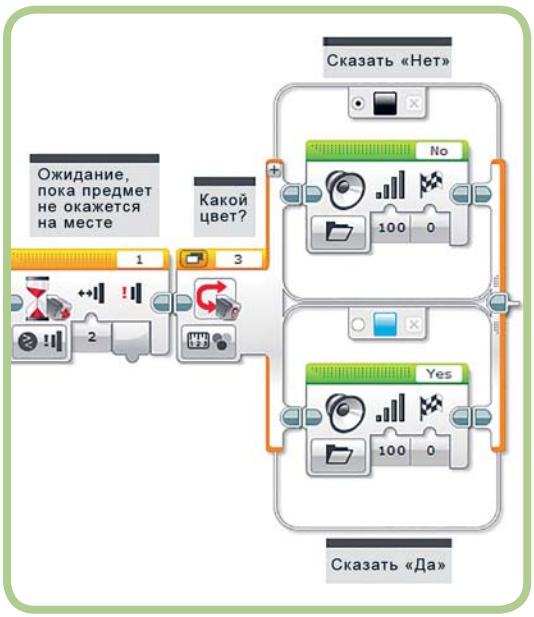


Рис. 5.23. Программа ожидает, пока объект не окажется на месте

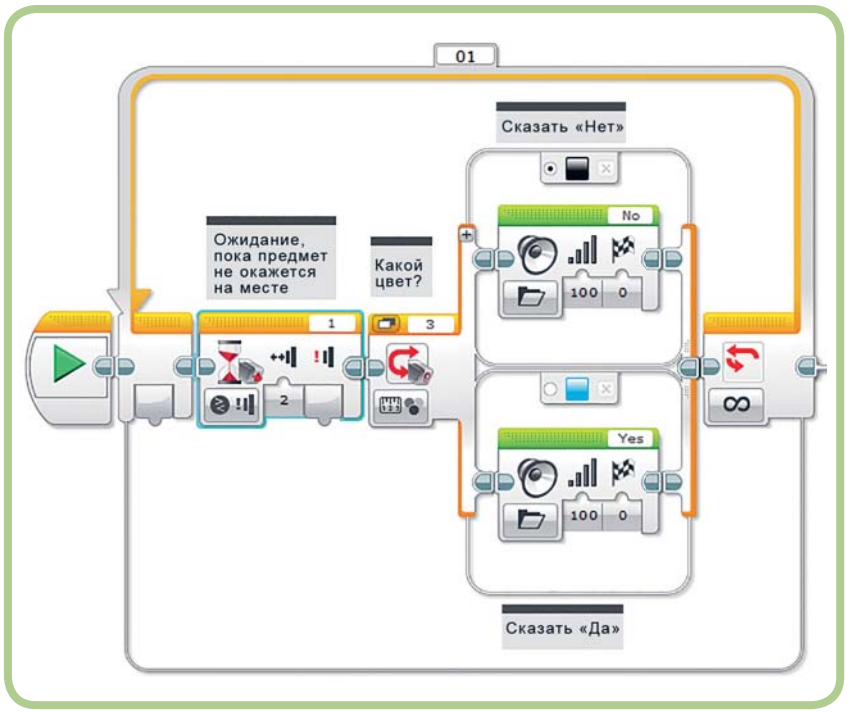


Рис. 5.24. Улучшенная версия программы IsItBlue

Добавление цикла

Для того чтобы программа продолжала работать, добавь блок **Цикл** (Loop), а затем перемести в него существующие блоки.

4. Добавь блок **Цикл** (Loop) в конец программы.

5. Выбери блоки **Ожидание** (Wait) и **Переключатель** (Switch) и перетащи их в блок **Цикл** (Loop).

На рис. 5.24 показана новая версия программы. После запуска она должна находиться в режиме ожидания касания бампера, выдать ответ “Yes” или “No”, а затем снова перейти в режим ожидания. Нажми кнопку «Выход» на модуле EV3, чтобы завершить программу.

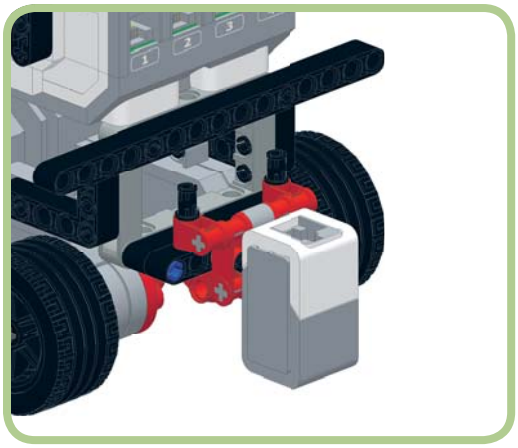


Рис. 5.25. Положение датчика цвета для программы LineFinder

Программа LineFinder

В этом разделе ты создашь программу, которая заставит робота TriBot двигаться вперед до тех пор, пока он не обнаружит линию на полу с помощью датчика цвета в режиме **Яркость отраженного света** (Reflected Light Intensity). Программа *LineFinder* заставляет робота двигаться вперед и останавливает его при обнаружении темной линии. Сначала открепи бампер датчика касания от передней части робота и установи датчик цвета так, чтобы он был направлен вниз, как показано на рис. 5.25.

ПРИМЕЧАНИЕ Для тестирования этой программы тебе потребуется светлая поверхность с темной линией. Ты можешь использовать лист белого картона и маркер или черную изоленту для создания линии.

Основная идея этой программы аналогична первой части программы *BumperBot*. Один блок **Рулевое управление** (Move Steering) в режиме **Включить** (On) заставляет робота TriBot двигаться вперед, а другой в режиме **Остановка** (Stop) позволяет его остановить. Между этими двумя блоками мы поместим блок **Ожидание** (Wait) в режиме **Датчик цвета** (Color Sensor) ⇒ **Яркость отраженного света** (Reflected Light Intensity). Единственная новая задача заключается в определении порогового значения для блока **Ожидание** (Wait).

Использование вкладки «Представление порта» для определения порогового значения

Используй вкладку **Представление порта** (Port View) (рис. 5.26), чтобы определить разумное пороговое значение для остановки робота. Убедись, что модуль EV3 подключен к программному обеспечению, а на вкладке **Представление порта** (Port View) отображается датчик цвета в режиме **Яркость отраженного света** (Reflected Light Intensity). На рис. 5.26 видно, что этот датчик подключен к порту 3. Помести робота TriBot на белую поверхность и обрати внимание на показание датчика. Теперь перемести робота так, чтобы датчик оказался над линией и считал новое значение. Для примера были выбраны значения датчика 74 и 6 соответственно.

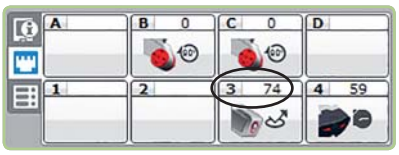


Рис. 5.26. Показание датчика цвета на светлой поверхности

Если датчик зафиксирует линию на белой поверхности, его показание будет находиться между двумя крайними значениями (в данном случае между 6 и 74). Но тогда какого порогового значения должен дождаться робот, чтобы остановиться? Если мы установим значение 6 для параметра **Пороговое значение** (Threshold value), робот не остановится, пока датчик не будет полностью находиться над черной поверхностью. Поскольку мы хотим, чтобы робот останавливался, добравшись до линии, надо установить пороговое значение выше 6.

Простой эффективный способ определения разумного порогового значения — это выбор средней величины между показаниями датчика, соответствующими светлому и темному значениям. Таким образом, робот остановится вскоре после обнаружения темной линии вместо того, чтобы ждать, пока датчик полностью окажется над линией и станет распознавать только черный цвет. Разделив сумму 74 и 6 на 2, получим значение 40, которое и будем использовать в качестве порогового значения в приведенном здесь коде. Не забудь вычислить собственное значение, исходя из показаний своего датчика цвета.

Теперь мы создадим программу, показанную на рис. 5.27.

1. Создай новую программу под названием *LineFinder*.

2. Добавь в программу блок **Рулевое управление** (Move Steering) и выбери для него режим **Включить** (On).

3. Задай значение **25** для параметра **Мощность** (Power).

4. Добавь блок **Ожидание** (Wait) и выбери режим **Датчик цвета** (Color Sensor) ⇒ **Сравнение** (Compare) ⇒ **Яркость отраженного света** (Reflected Light Intensity).

5. Для параметра **Тип сравнения** (Compare Type) уже будет выбран вариант **Меньше** (Less Than). Задай определенное ранее значение для параметра **Пороговое значение** (Threshold value).

6. Добавь блок **Рулевое управление** (Move Steering) после блока **Ожидание** (Wait). Выбери для него режим **Выключить** (Off).

7. Добавь блок **Ожидание** (Wait). Задай значение **5** для параметра **Секунды** (Seconds).

Рис. 5.27. Программа LineFinder

Второй блок **Рулевое управление** (Move Steering) и блок **Ожидание** (Wait) приводят к полной остановке робота до завершения программы, чтобы тот по инерции не проехал мимо линии. Запусти программу и при необходимости скорректируй пороговое значение в первом блоке **Ожидание** (Wait). Попробуй увеличить значение параметра **Мощность** (Power) в первом блоке **Рулевое управление** (Move Steering) для определения максимальной скорости робота, при которой он все еще может остановиться точно у линии.

ПРАКТИКУМ 5.4

Программа *AroundTheBlock* (см. подробнее в гл. 4) перемещает робота вперед на три оборота моторов, заставляя его двигаться по квадратной траектории. Измени количество повторений блока **Цикл** (Loop) с 4 на 40 (чтобы робот TriBot описал квадрат 10 раз), и ты увидишь, что робот все больше и больше отклоняется от курса из-за ошибок, возникающих при каждом движении, например из-за проскальзывания колес или неточного поворота. Эти ошибки накапливаются и становятся заметными. Для решения этой проблемы в программе *AroundTheBlock* применим блоки из программы *LineFinder*, чтобы робот двигался вперед до обнаружения линии, а затем поворачивался. Создай тестовую область с четырьмя темными линиями, по одной в конце каждой стороны квадрата. Программа должна заставить робота TriBot двигаться вперед до обнаружения первой линии, повернуть на 90°, двигаться вперед до обнаружения второй линии и т. д. Это не сделает отдельные движения более точными, но предотвратит накопление ошибок.

ПРАКТИКУМ 5.5

Из фрагментов программ *AroundTheBlock*, *LineFinder* и *RedAndBlue* можно создать простую программу для следования по некоторой траектории, в которой используется несколько цветных линий, указывающих роботу, как ему переходить от одной линии к другой. Используй блоки из программы *LineFinder*, чтобы заставить робота двигаться вперед до обнаружения линии. Затем измени блок **Переключатель** (Switch) из программы *RedAndBlue* так, чтобы робот поворачивал влево или вправо в зависимости от цвета линии. Для выполнения поворота можно использовать блок **Рулевое управление** (Move Steering) из программы *AroundTheBlock* с соответствующим параметром **Рулевое управление** (Steering).

Инфракрасный датчик и удаленный инфракрасный маяк

На рис. 5.28 изображен инфракрасный датчик и удаленный инфракрасный маяк. С помощью *инфракрасного датчика* можно измерять расстояние до объекта, а также направление и расстояние до удаленного инфракрасного маяка. Стоит отметить, что инфракрасный датчик не входит в образовательную версию конструктора. В большинстве случаев для измерения расстояния вместо инфракрасного датчика можно использовать ультразвуковой датчик (в одном из режимов **Расстояние** (Distance)).

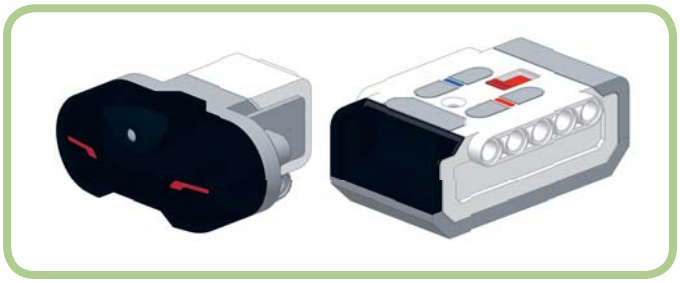


Рис. 5.28. Инфракрасный датчик и удаленный инфракрасный маяк

Режим «Приближение»

Используй режим **Приближение** (Proximity) для измерения расстояния до объекта, который находится перед датчиком. В этом режиме датчик излучает инфракрасный сигнал, а затем измеряет силу возвращенного сигнала, на основании этого программа определяет примерное расстояние до ближайшего объекта. Значение датчика находится в диапазоне между 0 (когда датчик находится вблизи объекта) и 100 (когда датчик находится далеко от объекта или когда объект не обнаружен). Способность объекта отражать инфракрасный сигнал зависит от многих факторов, включая его цвет и твердость. Например, такой мягкий круглый объект, как теннисный мяч, может обеспечить большее значение датчика, чем такой твердый и плоский объект, как книга, даже если они находятся на одинаковом расстоянии от датчика.

На рис. 5.29 изображен блок **Ожидание** (Wait) в режиме **Инфракрасный датчик** (Infrared Sensor) ⇒ **Сравнение** (Compare) ⇒ **Приближение** (Proximity), благодаря которому программа находится в режиме ожидания до тех пор, пока объект не будет обнаружен на определенном расстоянии. Благодаря режиму **Изменить** (Change) ⇒ **Приближение** (Proximity), программа ждет изменения расстояния до ближайшего объекта на определенную величину.

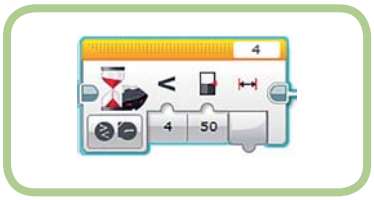


*Рис. 5.29. Блок **Ожидание** (Wait) в режиме **Инфракрасный датчик** (Infrared Sensor) ⇒ **Сравнение** (Compare) ⇒ **Приближение** (Proximity)*

Режимы «Направление маяка» и «Приближение маяка»

В режимах **Направление маяка** (Beacon Heading) и **Приближение маяка** (Beacon Proximity) инфракрасный датчик указывает направление или расстояние до удаленного инфракрасного маяка. Для использования этих режимов активируй режим **Маяк** (Beacon), нажав на маяке одноименную кнопку (рис. 5.30).

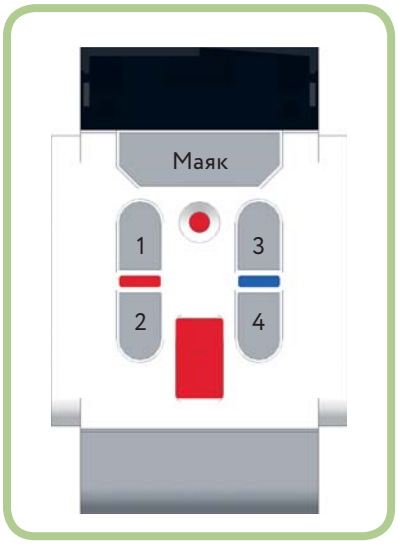


Рис. 5.30. *Кнопки удаленного инфракрасного маяка, включая кнопку* **Маяк** *(Beacon)*

На рис. 5.31 изображен блок **Ожидание** (Wait) в режиме **Сравнение** (Compare) ⇒ **Направление маяка** (Beacon Heading). Удаленный инфракрасный маяк может работать в четырех каналах, в которых задействованы разные инфракрасные сигналы для связи с датчиком. По умолчанию выбран канал 1, однако тебе может потребоваться другой, чтобы избежать помех от пульта телевизора, стереосистемы или другого инфракрасного маяка EV3. Если твой робот не реагирует на сигнал маяка EV3 или реагирует на пульт телевизора, попробуй использовать другой канал, отрегулировав красный ползунковый регулятор на маяке EV3, а затем соответствующим образом изменив канал в блоке программирования.

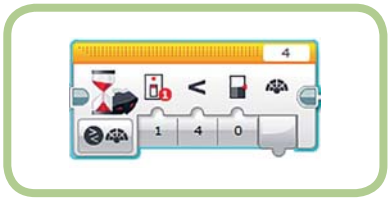


*Рис. 5.31. Блок **Ожидание** (Wait) в режиме **Сравнение** (Compare) ⇒ **Направление маяка** (Beacon Heading)*

В режиме **Направление маяка** (Beacon Heading) значения инфракрасного датчика находятся в диапазоне от –25 до 25. Значение 0 означает, что маяк находится непосредственно перед датчиком. Положительные значения указывают на то, что маяк находится справа от робота, отрицательные — что маяк находится слева от робота. Точного значения в градусах получить нельзя.

В режиме **Приближение маяка** (Beacon Proximity) датчик измеряет относительное расстояние до маяка, при этом значения находятся в диапазоне от 0 (близко) до 100 (далеко).

ПРИМЕЧАНИЕ **Инфракрасный сигнал, используемый для общения датчика с маяком, передается в пределах прямой видимости, т. е. для произведения замера между маяком и датчиком не должно быть препятствий.**

Режим «Удаленный»

Режим **Удаленный** (Remote) позволяет использовать кнопки маяка для управления программой. Ты выбираешь канал маяка и кнопку или кнопки, нажатия которых требуется дождаться, как показано на рис. 5.32.

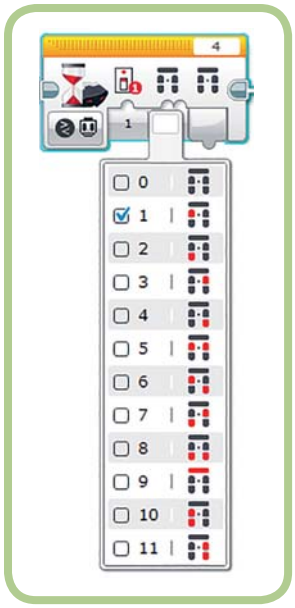


*Рис. 5.32. Блок **Ожидание** (Wait) в режиме **Инфракрасный датчик** (Infrared Sensor) ⇒ **Сравнение** (Compare) ⇒ **Удаленный** (Remote)*

Ты можешь выбрать одну кнопку, две кнопки или вообще ни одной. Также можно выбрать несколько вариантов, чтобы блок ожидал нажатия любой комбинации из пяти кнопок. Если ты выберешь несколько вариантов из списка, блок выйдет из режима ожидания, как только будет выполнено одно из заданных условий. Мы будем использовать этот блок в следующем разделе для управления программой *BumperBot*.

ПРАКТИКУМ 5.6

Для того чтобы увидеть, как работает удаленный маяк в режиме Маяк (Beacon), используй вкладку Представление порта (Port View) для отображения значений Направление маяка (Beacon Heading) и Приближение маяка (Beacon Proximity) по одному за раз. Приблизь маяк к датчику, а затем отодвинь от него. Обрати внимание, что показания стабильны, только когда маяк находится в пределах пары метров от датчика и примерно в зоне прямой видимости.

Программа BumperBot-WithButtons

В этом подразделе мы добавим инфракрасный датчик и удаленный маяк в программу *BumperBot*, чтобы робот начал движение только после нажатия кнопки на маяке. Если ты используешь образовательную версию конструктора, в которую не входит инфракрасный датчик, пропусти этот подраздел. Выполни следующие действия:

1. Открой программу *BumperBot*.
2. Добавь блок **Ожидание** (Wait) в начало программы между блоками **Начало** (Start) и **Цикл** (Loop).
3. Выбери для блока **Ожидание** (Wait) режим **Инфракрасный датчик** (Infrared Sensor) ⇒ **Сравнение** (Compare) ⇒ **Удаленный** (Remote).

После запуска программы (рис. 5.33) робот TriBot не должен начать движение, пока не будет нажата кнопка 1 на маяке. Важно, чтобы маяк находился перед роботом и указывал в направлении датчика.

Рис. 5.33. Ожидание нажатия кнопки в режиме ***Удаленный*** *(Remote)*

Ультразвуковой датчик

Ультразвуковой датчик (рис. 5.34) определяет расстояние до объекта, измеряя время, за которое высокочастотные звуковые волны отражаются от целевого объекта, что напоминает принцип работы сонара. Этот датчик входит только в образовательную версию конструктора. В большинстве программ, использующих его для измерения расстояния, также можно задействовать инфракрасный датчик в режиме **Приближение** (Proximity).

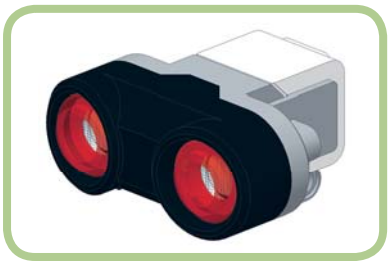


Рис. 5.34. Ультразвуковой датчик

Форма и текстура объекта оказывают значительное влияние на качество его обнаружения ультразвуковым датчиком. Некоторые поверхности отражают звуковые волны лучше других, поэтому их легче обнаружить. Плоские твердые поверхности найти легче всего, поскольку они отражают большую часть звуковых волн, в то время как изогнутые поверхности отражают некоторые звуковые волны, но рассеивают другие, а мягкие объекты скорее поглощают звуковые волны, нежели отражают их. Таким образом, датчик способен обнаружить твердые и плоские объекты на бóльшем расстоянии, чем мягкие и круглые.

Режимы «Расстояние в дюймах» и «Расстояние в сантиметрах»

Ультразвуковой датчик обычно используется в режиме **Расстояние в дюймах** (Distance Inches) или **Расстояние в сантиметрах** (Distance Centimeters). Эти режимы отличаются только единицами измерения. На рис. 5.35 изображен блок **Ожидание** (Wait) в режиме **Ультразвуковой датчик** (Ultrasonic Sensor) ⇒ **Сравнение** (Compare) ⇒ **Расстояние в дюймах** (Distance Inches).

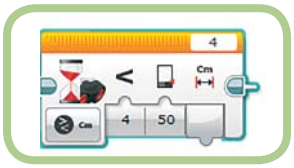


Рис. 5.35. Блок ***Ожидание*** *(Wait) в режиме* ***Ультразвуковой датчик*** *(Ultrasonic Sensor) ⇒* ***Сравнение*** *(Compare) ⇒* ***Расстояние в дюймах*** *(Distance Inches)*

Элементы конфигурации такие же, что и для датчика цвета за исключением того, что параметр **Пороговое значение** (Threshold value) представляет собой расстояние, а не интенсивность света. Далее мы будем использовать ультразвуковой датчик для создания программы *DoorChime*.

Режим «Присутствие/слушать»

Ультразвуковой датчик способен обнаружить присутствие другого ультразвукового датчика в режиме **Присутствие/слушать** (Presence/Listen), что может оказаться полезным при проведении игр или конкурсов с участием нескольких роботов. Единственным параметром, требующим настройки в блоке **Ожидание** (Wait) в режиме **Ультразвуковой датчик** (Ultrasonic Sensor) ⇒ **Сравнение** (Compare) ⇒ **Присутствие/слушать** (Presence/Listen), является порт, к которому подключен датчик (рис. 5.36). Блок находится в режиме ожидания до тех пор, пока не будет обнаружен другой ультразвуковой датчик.

Рис. 5.36. Блок ***Ожидание*** *(Wait) в режиме* ***Ультразвуковой датчик*** *(Ultrasonic Sensor) ⇒* ***Сравнение*** *(Compare) ⇒* ***Присутствие/слушать*** *(Presence/Listen)*

Программа DoorChime

Теперь мы создадим простой дверной звонок. Помести робота TriBot в дверной проем так, чтобы ультразвуковой или инфракрасный датчик был направлен поперек дверного проема (рис. 5.37). Робот будет издавать звуковой сигнал, когда

кто-то будет входить в дверь. Для того чтобы программа *DoorChime* запускалась автоматически снова и готовилась к приветствию следующего человека, помести основную часть программы *DoorChime* в блок **Цикл** (Loop).

Рис. 5.37. Расположение робота TriBot в дверном проеме для выполнения программы DoorChime

Обнаружение человека

В этой программе ты можешь использовать ультразвуковой или инфракрасный датчик для обнаружения человека, проходящего мимо робота, поскольку они оба могут измерять расстояние до объекта. При первом запуске программы датчик должен измерить ширину дверного проема. Когда человек пройдет мимо датчика, тот определит расстояние между роботом и вошедшим. Для определения порогового значения используй вкладку **Представление порта** (Port View), чтобы увидеть показание датчика после расположения робота в дверном проеме. В проведенных автором этой книги экспериментах указана ширина дверного проема, равная 80 см, а инфракрасный датчик в режиме **Приближение** (Proximity) считывает значение 61. Проходя в дверь, человек окажется ближе к роботу, поэтому используем значение 55 для инфракрасного датчика и 75 см — для ультразвукового датчика. Выполни следующие действия для создания программы:

1. Создай программу под названием *DoorChime*.
2. Добавь в программу блок **Цикл** (Loop). Он должен выполняться вплоть до остановки программы, поэтому тебе не нужно менять значения каких-либо параметров.
3. Добавь блок **Ожидание** (Wait) в блок **Цикл** (Loop).
4. Выбери режим **Ультразвуковой датчик** (Ultrasonic Sensor) ⇒ **Сравнение** (Compare) ⇒ **Расстояние в сантиметрах** (Distance Centimeters) или **Инфракрасный датчик** (Infrared Sensor) ⇒ **Сравнение** (Compare) ⇒ **Приближение** (Proximity) в зависимости от используемого датчика.
5. Для параметра **Пороговое значение** (Threshold value) задай значение, которое подходит для твоего дверного проема.

На рис. 5.38 показана программа, использующая инфракрасный датчик.

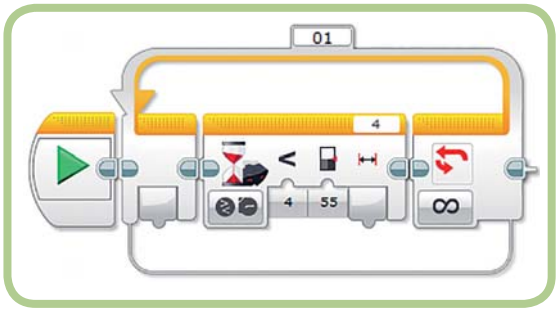


Рис. 5.38. Ожидание человека с использованием инфракрасного датчика

Воспроизведение звукового сигнала

Для воспроизведения звукового сигнала можно использовать два блока **Звук** (Sound) в режиме **Воспроизвести ноту** (Play Note), чтобы воспроизвести любые две ноты. Поэкспериментируй с различными комбинациями или добавь еще больше нот.

6. Добавь блок **Звук** (Sound) в блок **Цикл** (Loop).
7. Выбери для него режим **Воспроизвести ноту** (Play Note).
8. Щелкни по элементу конфигурации **Нота** (Note) и выбери ноту с помощью появившейся клавиатуры.
9. Добавь еще один блок **Звук** (Sound) в блок **Цикл** (Loop).
10. Выбери режим **Воспроизвести ноту** (Play Note) и добавь другую ноту.

На рис. 5.39 показаны два добавленных в программу блока **Звук** (Sound). Загрузи и протестируй программу, а затем поэкспериментируй с разными расстояниями, чтобы определить оптимальное пороговое значение. Попробуй различные параметры продолжительности и громкости для блоков **Звук** (Sound) и максимально творчески подойди к созданию звонка.

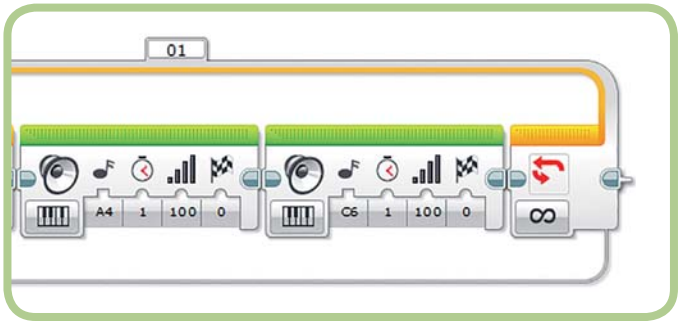


Рис. 5.39. Воспроизведение звонка

Рис. 5.40. Итоговая версия программы DoorChime, использующая инфракрасный датчик

Прекращение воспроизведения звонка

В текущей версии программа начинает воспроизводить звуковой сигнал, когда человек входит в дверной проем, и продолжает делать это, пока он не уйдет. Это может начать раздражать! Как сделать так, чтобы сигнал воспроизводился только один раз, когда человек проходит через дверной проем?

Для решения этой задачи мы добавим еще один блок **Ожидание** (Wait) после двух блоков **Звук** (Sound), чтобы приостановить выполнение программы на время, пока человек не выйдет из дверного проема. При использовании блока **Ожидание** (Wait) программа находится в режиме ожидания до того момента, пока показание ультразвукового или инфракрасного датчика не превысит исходное пороговое значение.

11. Перетащи блок **Ожидание** (Wait) в блок **Цикл** (Loop) и помести его справа от блоков **Звук** (Sound).

12. Выбери режим **Ультразвуковой датчик** (Ultrasonic Sensor) ⇒ **Сравнение** (Compare) ⇒ **Расстояние в сантиметрах** (Distance Centimeters) или **Инфракрасный датчик** (Infrared Sensor) ⇒ **Сравнение** (Compare) ⇒ **Приближение** (Proximity) в зависимости от используемого датчика.

13. Измени значение параметра **Пороговое значение** (Threshold value) так, чтобы оно превышало пороговое значение, использованное в первом блоке **Ожидание** (Wait).

14. Для параметра **Тип сравнения** (Compare Type) выбери вариант **Больше чем** (Greater Than).

На рис. 5.40 показана программа, использующая инфракрасный датчик. Протестируй ее, чтобы убедиться в том, что она работает так, как нужно при прохождении человека через дверной проем. Затем поэкспериментируй с разными расстояниями, чтобы посмотреть, как работает программа в случае, если через дверной проем пройдут несколько человек.

Гироскопический датчик

Гироскопический датчик (входит только в образовательную версию конструктора), показанный на рис. 5.41, измеряет вращательное движение. Он может сообщить программе, как быстро датчик вращается в направлении любой из двух изображенных на его корпусе стрелок. Поскольку этот датчик измеряет вращательное движение в градусах в секунду, он также может определить преодоленное расстояние в градусах.

Рис. 5.41. Гироскопический датчик

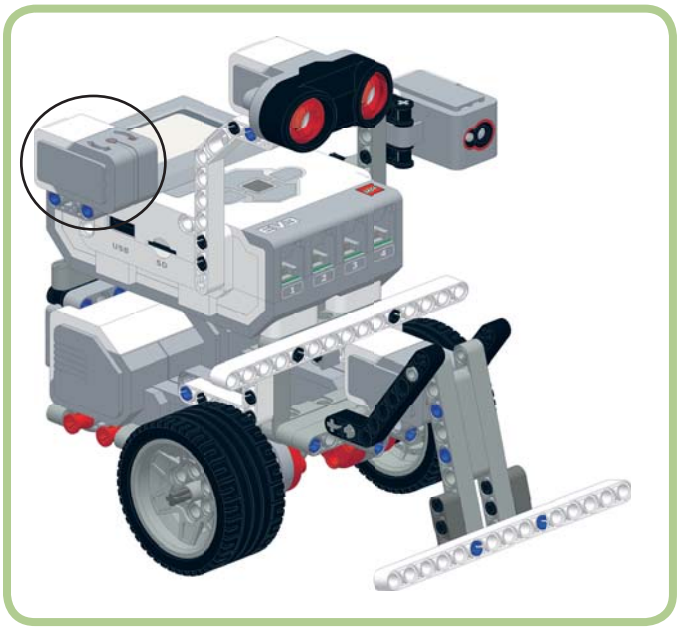


Рис. 5.42. Гироскопический датчик, установленный на роботе TriBot

Гироскопический датчик измеряет вращение только в одной плоскости, поэтому убедись в его правильной установке. Например, будучи установленным так, как показано на рис. 5.42, он позволяет измерить скорость и угол вращения робота TriBot влево и вправо, но не даст знать, если робот опрокинется на бок или кувыркнется вперед или назад.

Режим «Скорость»

В режиме **Скорость** (Rate) гироскопический датчик измеряет скорость вращения в градусах в секунду. Это может быть полезно, когда роботу требуется выполнить поворот с заданной скоростью. Блок **Ожидание** (Wait) в режиме **Гироскопический датчик** (Gyro Sensor) ⇒ **Сравнение** (Compare) ⇒ **Скорость** (Rate) изображен на рис. 5.43. Гироскопический датчик показывает положительное значение при вращении по часовой стрелке и отрицательное — при вращении против часовой стрелки.

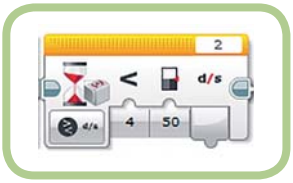


*Рис. 5.43. Блок **Ожидание** (Wait) в режиме **Гироскопический датчик** (Gyro Sensor) ⇒ **Сравнение** (Compare) ⇒ **Скорость** (Rate)*

Ты можешь использовать этот датчик для обнаружения изменений в параметрах вращения, свидетельствующих об опрокидывании робота, а также для проведения экспериментов, связанных с движением. Представь, что у тебя есть возможность записывать показания датчика, когда робот преодолевает препятствие или путешествует по лабиринту. Полученные данные можно изучить, чтобы выяснить, как движение робота изменяется со временем.

Режим «Угол»

В режиме **Угол** (Angle) (рис. 5.44) датчик определяет, на сколько градусов повернулся робот с момента последнего сброса показаний датчика.

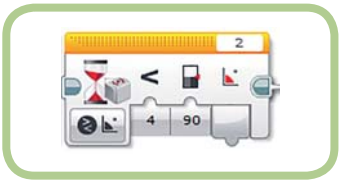


*Рис. 5.44. Блок **Ожидание** (Wait) в режиме **Гироскопический датчик** (Gyro Sensor) ⇒ **Сравнение** (Compare) ⇒ **Угол** (Angle)*

Используя режим **Изменить** (Change), можно заставить робота, например, выполнить поворот на 90° из любого текущего положения, независимо от начальной точки. Это особенно полезно для поворота за угол.

ПРИМЕЧАНИЕ **Гироскопический датчик измеряет скорость вращения, а значение угла выводит из скорости движения и истекшего времени. Слишком большая скорость вращения датчика может нарушить этот процесс. Обрати внимание, что значение угла со временем становится менее точным из-за накопления погрешности измерения. После подключения гироскопического датчика к модулю EV3 его следует удерживать в неподвижном состоянии еще несколько секунд для выполнения калибровки. Нам удалось получить хороший результат, отключив датчик, подождав несколько секунд и подключив его обратно к роботу, который все это время остается неподвижным.**

Сброс угла

После запуска программы значение угла гироскопического датчика сбрасывается на 0. Для обнуления показания во время выполнения программы используй блок **Гироскопический датчик** (Gyro Sensor) в режиме **Сброс** (Reset), как показано на рис. 5.45. Этот блок находится на желтой вкладке палитры программирования с блоками датчиков.

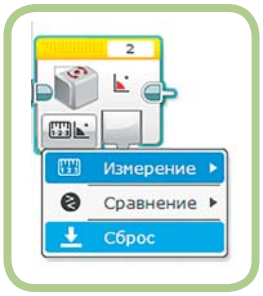


Рис. 5.45. Сброс показаний гироскопического датчика

Программа GyroTurn

Гироскопический датчик особенно полезен для контроля над движением робота TriBot и обеспечения точности выполняемых им поворотов.

Цель программы *GyroTurn* — заставить робота совершить поворот на 90°, используя гироскопический датчик для определения момента, когда следует останавливать моторы. Для создания программы сначала:

1. Создай новую программу под названием *GyroTurn*.
2. Добавь в программу блок **Рулевое управление** (Move Steering).
3. Выбери режим **Включить** (On) и задай значение **30** для параметра **Мощность** (Power).
4. Для параметра **Рулевое управление** (Steering) задай значение **25**.

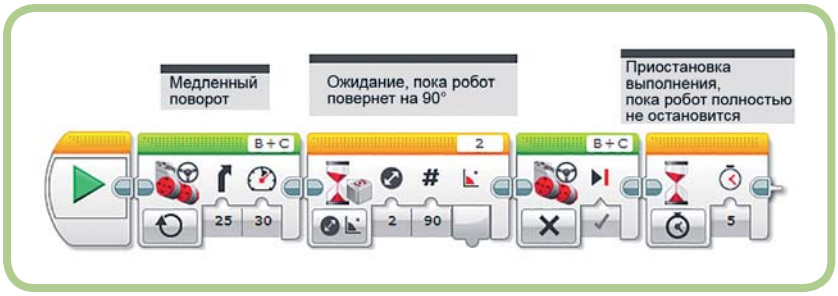


Рис. 5.46. Программа GyroTurn

5. Добавь в программу блок **Ожидание** (Wait). Выбери режим **Гироскопический датчик** (Gyro Sensor) ⇒ **Изменить** (Change) ⇒ **Угол** (Angle).

6. Для параметра **Пороговое значение** (Threshold value) задай значение **90**.

7. Добавь в конец программы блок **Рулевое управление** (Move Steering). Выбери для него режим **Выключить** (Off).

8. Добавь блок **Ожидание** (Wait). Задай значение **5** для параметра **Секунды** (Seconds).

На рис. 5.46 показана готовая программа. Первый блок **Ожидание** (Wait) будет ждать, пока угол гироскопического датчика изменится на 90°. Второй блок **Ожидание** (Wait) гарантирует полную остановку робота до завершения программы. Пяти секунд более чем достаточно для полной остановки робота. Обычно я использую пять секунд, когда считаю, что одной секунды может оказаться недостаточно.

Запусти программу и открой вкладку **Представление порта** (Port View), чтобы посмотреть, насколько итоговый угол приблизился к значению 90°. Если для параметра **Мощность** (Power) блока **Рулевое управление** (Move Steering) задано значение 30, поворот выполняется довольно точно — обычно с точностью до 1°. Теперь попробуй увеличить значение параметра **Мощность** (Power) до 50, 70 и 90. С увеличением скорости робота точность снижается, и робот все чаще заезжает слишком далеко. Это вызвано небольшой задержкой между моментом, когда программа понимает, что значение датчика достигло порогового значения, и моментом остановки моторов. При слишком большой скорости движения моторов робот TriBot может проехать на несколько градусов дальше цели, прежде чем остановиться. О том, как решить эту проблему, поговорим в гл. 13, после того, как ты освоишь другие методы программирования EV3.

Датчик вращения мотора

В каждый мотор EV3 встроен *датчик вращения мотора*, который измеряет количество градусов и оборотов мотора. Этот датчик можно использовать для контроля над пройденным роботом расстоянием и считывания текущего значения параметра **Мощность** (Power), что может быть полезно при проведении экспериментов с блоками **Рулевое управление** (Move Steering) и **Независимое управление моторами** (Move Tank) или при измерении скорости вращения мотора, вызванного внешней силой (например, ветряком или беговой дорожкой). На рис. 5.47 изображен блок **Ожидание** (Wait) в режиме **Вращение мотора** (Motor Rotation) ⇒ **Сравнение** (Compare) ⇒ **Градусы** (Degrees).

Ты можешь в любое время считать показание или сбросить его с помощью блока **Вращение мотора** (Motor Rotation) в режиме **Сброс** (Reset), как показано на рис. 5.48. Часто с датчиком вращения мотора ты будешь использовать два блока: один для считывания значения и один для сброса.

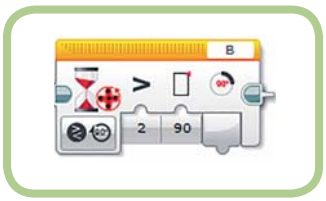


Рис. 5.47. *Блок* **Ожидание** *(Wait) в режиме* **Вращение мотора** *(Motor Rotation)* ⇒ **Сравнение** *(Compare)* ⇒ **Градусы** *(Degrees)*

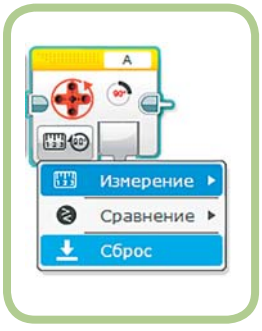


Рис. 5.48. Сброс показаний датчика вращения мотора

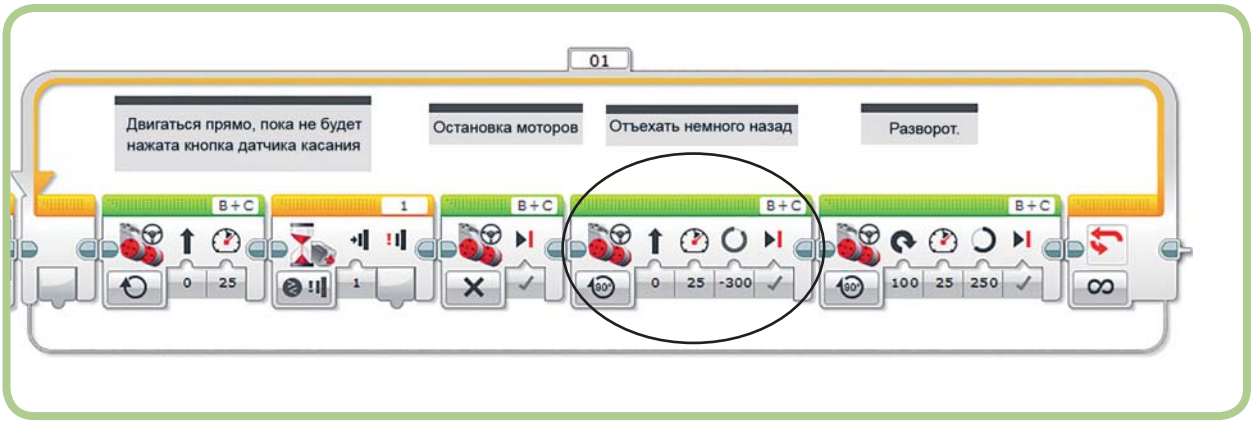


*Рис. 5.49. Блок **Рулевое управление** (Move Steering), заставляющий робота откатываться назад*

ПРАКТИКУМ 5.7

Измени программу *AroundTheBlock*, добавив в нее блоки из программы *GyroTurn*, чтобы на каждом углу робот поворачивал на 90°. Протестируй обе версии программы, используя разные параметры скорости и поверхности. Применение этого датчика должно повысить точность поворотов при изменении условий.

Программа *BumperBot2*

В исходной версии программы *BumperBot* блок **Рулевое управление** (Move Steering) с параметром **Продолжительность** (Duration), установленным на значение –300°, заставлял робота TriBot откатываться назад при столкновении с препятствием (рис. 5.49). Что если сделать так, чтобы при откате назад робот издавал звуковой сигнал, как это делает школьный автобус или крупный грузовик?

Ты уже знаешь, как использовать блок **Звук** (Sound), чтобы заставить робота издавать звуковой сигнал. В данном случае задача заключается в том, чтобы звуковой сигнал издавался во время движения и чтобы робот отъехал на правильное расстояние. Один из способов решения этой задачи — запустить робота с помощью блока **Рулевое управление** (Move Steering) в режиме **Включить** (On), а затем использовать блок **Цикл** (Loop) для остановки робота после того, как он переместится как минимум на –300°. Внутри блока **Цикл** (Loop) можно применять блок **Звук** (Sound), для того чтобы заставить робота издавать звуковой сигнал. Скопируй и вставь блоки программы *BumperBot*, создав ее дубликат, а затем щелкни по названию вновь созданной программы *BumperBot2* в списке программ, чтобы открыть и отредактировать ее. Далее мы заменим блок **Рулевое управление** (Move Steering) (обведенный на рис. 5.49) блоками, которые сбросят показание датчика вращения мотора, заставят робота TriBot начать движение, а затем будут издавать звуковой сигнал, пока значение датчика вращения не станет меньше –300°.

1. Добавь блок **Вращение мотора** (Motor Rotation) слева от блока **Рулевое управление** (Move Steering), обведенного на рис. 5.49.

2. Выбери режим **Сброс** (Reset) и убедись в том, что для параметра **Порт** (Port) выбран вариант **B**.

3. Измени режим блока **Рулевое управление** (Move Steering), обведенного на рис. 5.49, на **Включить** (On) и задай значение **–15** для параметра **Мощность** (Power). В результате робот будет медленно откатываться назад до тех пор, пока ты его не остановишь.

Параметр **Продолжительность** (Duration) отсутствует, поэтому мы будем использовать отрицательное значение параметра **Мощность** (Power), чтобы заставить робота откатиться назад. Теперь эта часть программы должна выглядеть, как показано на рис. 5.50.

Рис. 5.50. Сброс показаний датчика вращения моторов и медленный откат назад

4. Добавь блок **Цикл** (Loop) справа от блока **Рулевое управление** (Move Steering) (режим которого был только что изменен на **Включить** (On)).

5. Выбери для блока **Цикл** (Loop) режим **Вращение мотора** (Motor Rotation) ⇒ **Сравнение** (Compare) ⇒ **Градусы** (Degrees).

6. Для параметра **Тип сравнения** (Compare Type) выбери вариант **Меньше** (Less Than), а для параметра **Пороговое значение** (Threshold value) задай значение **–300**.

7. Перетащи блок **Звук** (Sound) в блок **Цикл** (Loop) и выбери для него режим **Воспроизвести тон** (Play Tone).

8. Для параметра **Продолжительность** (Duration) блока **Звук** (Sound) задай значение **0.5**.

9. Помести блок **Ожидание** (Wait) справа от блока **Звук** (Sound) внутри блока **Цикл** (Loop). Задай период ожидания в **0.25** секунды для добавления паузы между звуковыми сигналами.

Теперь блок **Цикл** (Loop) должен выглядеть так, как показано на рис. 5.51. Запусти программу. После столкновения с препятствием робот TriBot должен издавать звуковой сигнал при откате назад. Теперь робот будет откатываться дальше, чем раньше (чуть более чем на –300°), поскольку он будет проверять значение датчика вращения мотора только после воспроизведения тона и ожидания в течение четверти секунды. Это не проблема, потому что расстояние, на которое откатывается робот, не является критическим. Ему требуется отъехать достаточно далеко, чтобы суметь повернуться, не задев препятствие. Поэкспериментируй с тоном и громкостью, чтобы найти подходящую комбинацию.

Рис. 5.51. Подача звукового сигнала при откате назад

Дальнейшее исследование

Далее перечислены некоторые идеи для дальнейшего исследования возможностей использования датчиков.

1. Используй вкладку **Представление порта** (Port View), чтобы поэкспериментировать с режимами датчиков и найти подходящие сочетания.
   - Какие цвета и оттенки хорошо работают с датчиком цвета?
   - Как текстура объекта влияет на показания ультразвукового или инфракрасного датчика? Поэкспериментируй, сравни теннисный мяч и бейсбольный. Как расстояние влияет на надежность показаний датчиков?
   - Как ведет себя гироскопический датчик при очень быстром повороте? Что происходит, когда он поворачивается, но не в той плоскости, в которой он производит измерения, например, когда робот TriBot опрокидывается на бок?
   - Как цвет объекта влияет на датчик цвета в режиме **Яркость отраженного света** (Reflected Light Intensity)? Как расстояние до объекта влияет на значение яркости отраженного света? На каком расстоянии показания этого режима становятся ненадежными? Подумай, как это влияет на место расположения датчика для выполнения программы обнаружения линии или движения вдоль линии.

2. Используй кнопки на удаленном инфракрасном маяке для управления подъемным рычагом. Напиши программу, которая находится в режиме ожидания, пока ты не нажмешь кнопку, а затем начинает перемещать рычаг и останавливает его при очередном нажатии кнопки. Используй небольшое значение скорости, чтобы иметь возможность остановить рычаг, пока он не передвинулся слишком далеко. Затем добавь блок **Переключатель** (Switch), который перемещает рычаг вверх или вниз в зависимости от нажатой кнопки.

3. Используй инфракрасный или ультразвуковой датчик для создания детектора движения. Блок **Ожидание** (Wait) в режиме **Изменить** (Change) хорошо подходит для обнаружения приближающегося или удаляющегося от датчика предмета. Реши, что должен делать робот при обнаружении незваного гостя.

Заключение

Датчики EV3 позволяют создать робота, который взаимодействует с окружающим его пространством. Датчик цвета, касания, ультразвуковой, инфракрасный, гироскопический датчик, а также датчик вращения мотора позволяют роботу по-разному воспринимать сигналы из окружающей среды. В связи с этим можно создавать роботов, которые демонстрируют разнообразное и интересное поведение. В примерах, представленных в этой главе, было показано, как использовать датчики в сочетании с блоками **Ожидание** (Wait), **Цикл** (Loop) и **Переключатель** (Switch) для создания интересных программ EV3, начиная от простого дверного звонка и заканчивая более сложной программой *BumperBot*.

6

Процесс выполнения программы

Процесс выполнения программы — это порядок запуска блоков программирования. Как правило, блоки выполняются слева направо, однако этим процессом можно управлять, заставляя блоки ждать, повторять или выбирать действие в соответствии с определенными условиями. Есть три основных блока, используемые для управления процессом выполнения программы: **Ожидание** (Wait), **Переключатель** (Switch) и **Цикл** (Loop).

Ты уже знаешь, как работает блок **Ожидание** (Wait). В этой главе подробно описано о блоках **Переключатель** (Switch) и **Цикл** (Loop), а также о блоке **Прерывание цикла** (Loop Interrupt), который используется для управления блоками **Цикл** (Loop).

ПРИМЕЧАНИЕ Блоки **Переключатель** (Switch) и **Цикл** (Loop) также имеют некоторые функции, которые используются только с шинами данных (подробно о шинах данных — см. гл. 9 и 10).

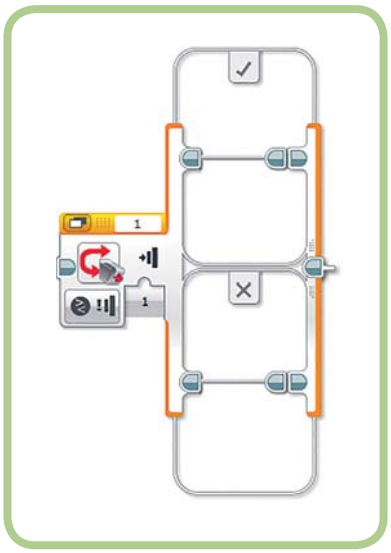


*Рис. 6.1. Блок **Переключатель** (Switch)*

Блок «Переключатель»

С помощью блока **Переключатель** (Switch) (рис. 6.1) программа принимает решение о том, какие блоки следует выполнять. Такая структура называется *условной*, поскольку процесс выполнения программы изменяется в зависимости от того, выполняется условие или нет. Блок **Переключатель** (Switch) проверяет условие, чтобы выбрать из двух или более групп блоков — *случаев*. Благодаря этому программа принимает решение и реагирует на данные, полученные от датчиков робота. Например, программа *RedOrBlue*, о которой пойдет речь далее в этой главе, использует показания датчика цвета, чтобы решить, какой из блоков **Звук** (Sound) следует выполнить.

Задание условия

Для того чтобы задать условие в блоке **Переключатель** (Switch), сначала выбери датчик и режим из соответствующего раскрывающегося списка. Затем введи значение параметра **Пороговое значение** (Threshold value) и настрой любые другие дополнительные параметры, например укажи порт датчика.

Подумай о том, как настроить режим в виде вопроса. Блок, показанный на рис. 6.1, спрашивает: «Нажата ли кнопка датчика касания?» Этот вопрос подразумевает ответ «Да» или «Нет», значит у блока **Переключатель** (Switch) может быть только две ситуации: «Истина» и «Ложь».

Другие вопросы допускают более двух ответов. Например, вопрос «Какой цвет обнаружил датчик цвета?» предполагает восемь возможных ответов, поскольку датчик цвета может выдавать восемь значений (семь цветов и значение **Нет цвета** (No Color)). Для этого вопроса блок **Переключатель** (Switch) может предусматривать до восьми случаев.

ИЗМЕНЕНИЕ РАЗМЕРА БЛОКА

Размер блоков **Переключатель** (Switch) и **Цикл** (Loop) автоматически изменяется при добавлении в них других блоков. Ты можешь самостоятельно настроить размер блока **Переключатель** (Switch) или **Цикл** (Loop), чтобы уменьшить его после удаления некоторых блоков, отобразить дополнительные вкладки блока **Переключатель** (Switch) или освободить место для комментариев.

После щелчка по блоку **Цикл** (Loop) или **Переключатель** (Switch) отобразятся специальные маркеры (рис. 6.2). Щелкни и перетащи один из маркеров для изменения размера блока.

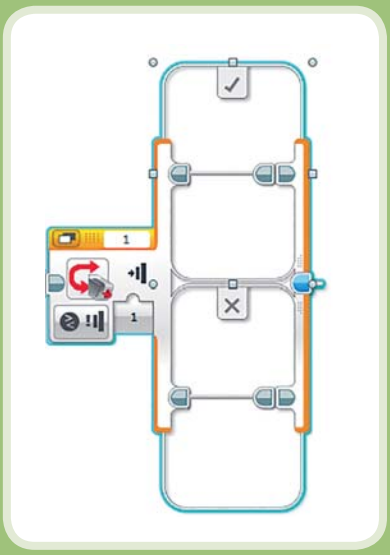


Рис. 6.2. Маркеры для изменения размера блока

Программа LineFollower

LineFollower предназначена для того, чтобы запрограммировать робота следовать вдоль линии. В ней блок **Переключатель** (Switch) применяется роботом для определения направления движения. Робот TriBot использует датчик цвета в режиме **Яркость отраженного света** (Reflected Light Intensity), чтобы следовать вдоль кромки линии, корректируя параметр **Рулевое управление** (Steering) на основе показания датчика.

Для этой программы тебе нужно открепить от робота бампер с датчиком касания. Вместо него установить датчик цвета так, чтобы он был направлен вниз (рис. 6.3).

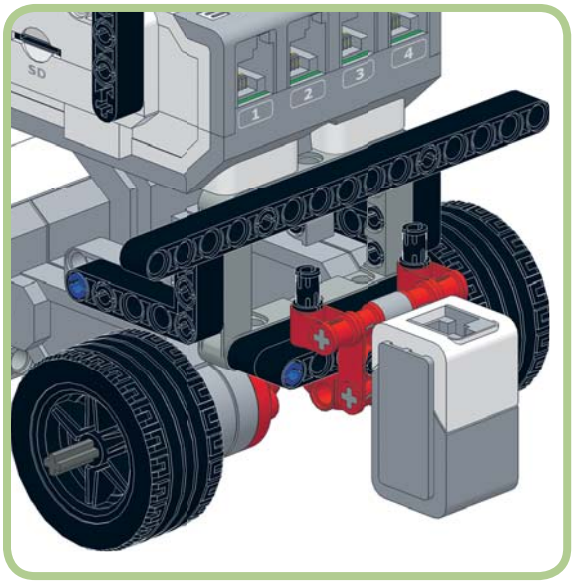


Рис. 6.3. Положение датчика цвета для следования вдоль линии

Для тестирования этой программы тебе понадобится темная линия. Ты можешь использовать черную изоленту или маркер, чтобы нарисовать овал на белом листе картона (рис. 6.4). Ширина линии должна быть не менее 3 см, а цвет линии — намного темнее фона.

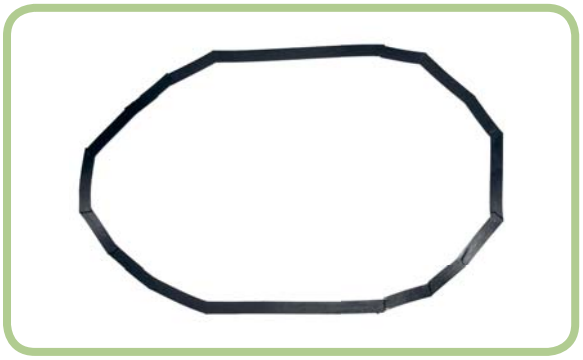


Рис. 6.4. Тестовая линия, отмеченная с помощью изоленты на листе картона

ПРИМЕЧАНИЕ **Эта программа лучше всего работает с плавными поворотами. При резких поворотах могут возникать проблемы. Однако окончательный вариант программы, представленный в гл. 19, отлично подходит для выполнения резких поворотов.**

Основная часть программы

Эта программа заставляет робота следовать вдоль линии, постоянно корректируя направление движения так, чтобы датчик цвета все время находился над краем линии (рис. 6.5). Датчик цвета в режиме **Яркость отраженного света** (Reflected Light Intensity) считывает, сколько света

отражается от небольшой круглой области под ним. Когда датчик находится над белой поверхностью, он выдает большое значение, поскольку от нее отражается больше света. И наоборот, полностью находясь над темной линией, он выдает низкое значение в связи с малым количеством отраженного от нее света. Между двумя этими крайними значениями показание датчика варьируется в зависимости от того, какая часть линии находится под ним. При движении вперед робот будет использовать показание датчика цвета, чтобы сообщить свое положение относительно края линии. Если робот окажется слишком далеко на линии, он повернет влево, чтобы переместиться обратно к ее краю. Если робот удалится от линии, он повернет вправо, чтобы вернуться к краю линии.

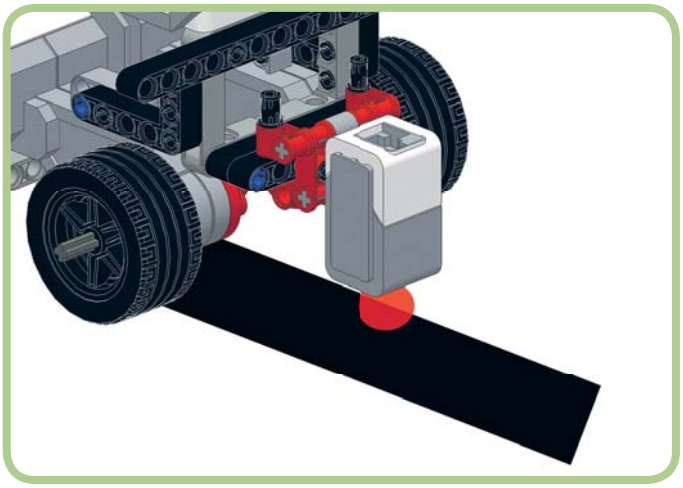


Рис. 6.5. Датчик цвета, находящийся над краем линии

В данной программе используется блок **Переключатель** (Switch), чтобы выбрать из двух блоков **Рулевое управление** (Move Steering), один из которых направляет робота влево, а другой — вправо. На рис. 6.6 показана итоговая версия программы. Вся программа заключена в блок **Цикл** (Loop), обеспечивающий работу программы вплоть до ее остановки. Блок **Переключатель** (Switch) считывает показание датчика цвета и решает, какой блок **Рулевое управление** (Move Steering) следует выполнить. Блок в верхней части (случай «Истина») направляет робота TriBot влево, а блок в нижней части (случай «Ложь») направляет его вправо.

Теперь давай создадим программу:

1. Создай новый проект под названием *Chapter6*.
2. Создай новую программу под названием *LineFollower*.
3. Добавь блок **Цикл** (Loop).
4. Перетащи блок **Переключатель** (Switch) в блок **Цикл** (Loop) (размер блока **Цикл** (Loop) увеличится, чтобы вместить блок **Переключатель** (Switch)).
5. Добавь блок **Рулевое управление** (Move Steering) для каждого случая блока **Переключатель** (Switch).
6. Выбери для блока **Переключатель** (Switch) режим **Датчик цвета** (Color Sensor) ⇒ **Сравнение** (Compare) ⇒ **Яркость отраженного света** (Reflected Light Intensity).

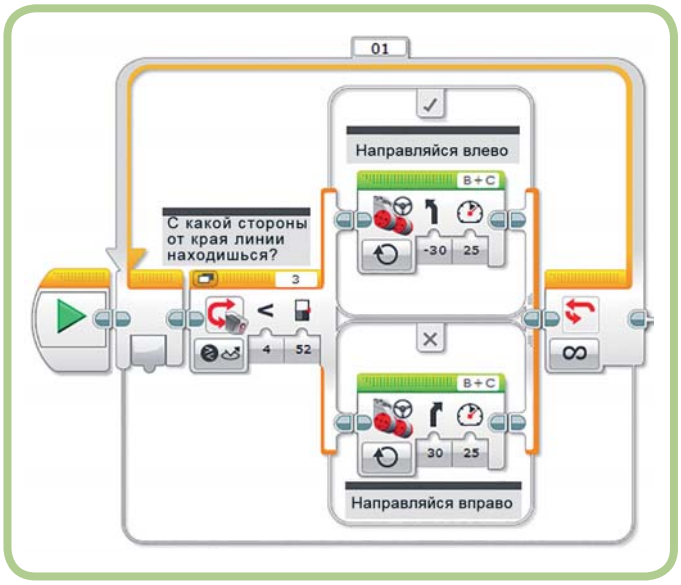


Рис. 6.6. Программа LineFollower

Теперь осталось только настроить параметры каждого блока. Давай обсудим способы определения их значений.

Выбор порогового значения для датчика цвета

Как определить значение параметра **Пороговое значение** (Threshold value) для блока **Переключатель** (Switch)? Чтобы заставить робота TriBot следовать вдоль края линии, надо найти значение, которое считывает датчик цвета, когда он находится на краю линии. На рис. 6.7 показаны показания датчика цвета, полученные в пяти разных положениях робота при его перемещении по линии. Простой надежный способ выбора разумного порогового значения — определение средней величины между высоким (когда датчик находится полностью вне линии) и низким (когда датчик находится полностью над линией) значениями. При использовании значений, показанных на рис. 6.6, получается пороговое значение (92 + 13) / 2, которое надо округлить до 52 и задать для параметра **Пороговое значение** (Threshold value) в блоке **Переключатель** (Switch). Обрати внимание на то, что оно близко к значению, которое было получено, когда робота поместили над краем линии. Твое значение может немного отличаться в зависимости от датчика, тестовой поверхности и освещения.

7. Задай значение для параметра **Пороговое значение** (Threshold value) блока **Переключатель** (Switch). Теперь этот блок должен выглядеть так:

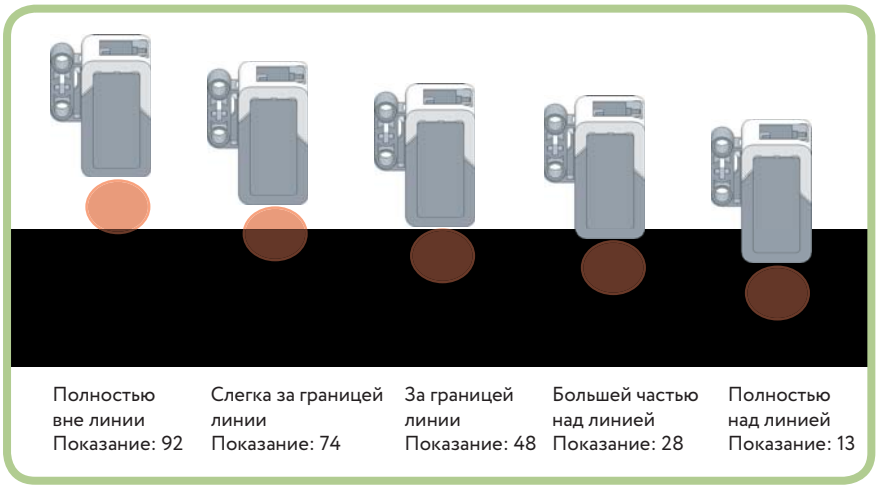


Рис. 6.7. Показания датчика цвета при различных положениях робота

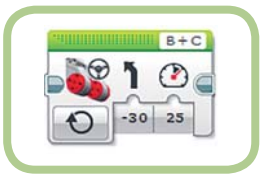
Настройка блоков «Рулевое управление»

Два блока **Рулевое управление** (Move Steering) имеют похожие параметры, однако они заставляют робота двигаться в противоположных направлениях. Скорость моторов и значение параметра **Рулевое управление** (Steering) оказывают существенное влияние на точность, с которой робот TriBot движется вдоль линии. При слишком низком значении параметра **Рулевое управление** (Steering) робот не сможет поворачивать достаточно быстро, чтобы следовать вдоль кривой линии. Если значение параметра **Рулевое управление** (Steering) будет слишком высоким, робот будет слишком часто отклоняться из стороны в сторону, переходя от одного крайнего значения к другому. При слишком быстром движении робот хуже реагирует на изменения в направлении линии. Начни со значений 30 и –30 для параметра **Рулевое управление** (Steering) и значения 25 для параметра **Мощность** (Power):

8. Выбери блок **Рулевое управление** (Move Steering) в верхней последовательности.

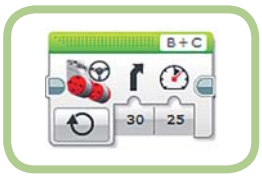
9. Выбери для него режим **Включить** (On).

10. Задай значение **25** для параметра **Мощность** (Power), а для параметра **Рулевое управление** (Steering) — значение **–30**. Блок должен выглядеть следующим образом:



11. Выбери блок **Рулевое управление** (Move Steering) в нижней последовательности и выбери для него режим **Включить** (On).

12. Задай значение **25** для параметра **Мощность** (Power), а для параметра **Рулевое управление** (Steering) — значение **30**. Блок должен выглядеть так:



Тестирование программы

Теперь загрузи и запусти программу, чтобы оценить ее работоспособность. Возможно, тебе потребуется скорректировать скорость движения робота и резкость его поворотов. В этом случае убедись, что изменения внесены в оба блока **Рулевое управление** (Move Steering). Удостоверившись в том, что программа работает, постарайся определить максимальную скорость, при которой робот все еще способен довольно точно следовать вдоль линии.

Наличие более двух вариантов

Первая версия программы *LineFollower* заставляет робота отклоняться вправо и влево, следуя вдоль прямой линии, что вызвано постоянной корректировкой параметра **Рулевое управление** (Steering). Ты можешь сделать его движение более плавным. Для этого используй три блока **Рулевое управление** (Move Steering): один для движения влево, второй для движения прямо, а третий для движения вправо.

В режиме **Датчик цвета** (Color Sensor) ⇒ **Сравнение** (Compare) ⇒ **Яркость отраженного света** (Reflected Light Intensity) блок **Переключатель** (Switch) может выбрать только один из двух наборов блоков, исходя из показаний датчика цвета. Чтобы выбрать один из трех вариантов, нужно использовать два блока **Переключатель** (Switch). Первый блок **Переключатель** (Switch) решает, должен ли робот повернуть влево, а второй — должен ли он двигаться прямо или повернуть вправо. Такая структура часто применяется при создании программ для роботов EV3 (и вообще в программировании) для принятия сложных решений.

Начни с внесения следующих изменений в программу *Line Follower*:

13. Помести блок **Переключатель** (Switch) в нижнюю часть существующего блока **Переключатель** (Switch) справа от блока **Рулевое управление** (Move Steering).

14. Выбери для нового блока **Переключатель** (Switch) режим **Датчик цвета** (Color Sensor) ⇒ **Сравнение** (Compare) ⇒ (**Яркость отраженного света** Reflected Light Intensity).

15. Перетащи существующий блок **Рулевое управление** (Move Steering) (который заставляет робота поворачивать вправо) в нижнюю часть нового блока **Переключатель** (Switch).

16. Помести новый блок **Рулевое управление** (Move Steering) в верхнюю часть нового блока **Переключатель** (Switch).

17. Выбери режим **Включить** (On) и задай для параметра **Мощность** (Power) значение **25**.

Теперь блок **Переключатель** (Switch) должен выглядеть так, как показано на рис. 6.8.

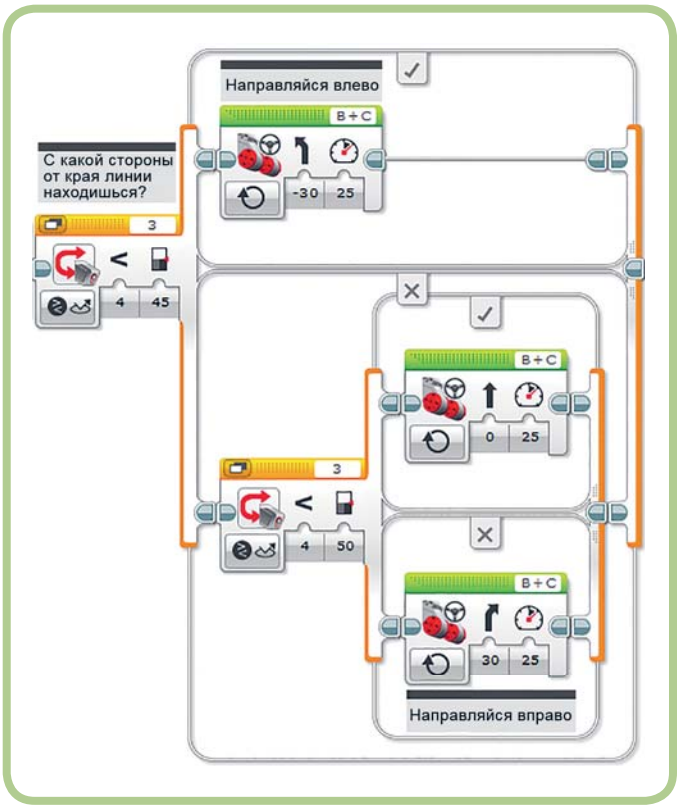


Рис. 6.8. Установка пороговых значений для новых блоков

Надо установить пороговые значения для двух блоков **Переключатель** (Switch), чтобы робот TriBot двигался прямо, находясь на краю линии, и поворачивал при удалении от ее края (обусловленного либо наездом на линию, либо слишком сильным отклонением от нее).

Сначала было использовано только одно пороговое значение — 52, которое находится посередине между двумя предельными значениями 13 и 92. В качестве двух пороговых значений можно использовать показания, находящиеся посередине между этим средним и двумя предельными значениями. Это 32 и 72. Если еще раз посмотрим на рис. 6.6, то увидим, что эти значения имеют смысл, поскольку они близки к показаниям, которые были получены, когда датчик находился большей частью на линии и большей частью вне линии. Эти пороговые значения были заданы для того, чтобы робот двигался прямо, когда показание датчика находится между 32 и 72, и поворачивал влево или вправо, когда показание находится вне этого диапазона. В табл. 6.1 показано, как должен вести себя робот в зависимости от показания датчика цвета. Не забудь рассчитать собственные значения, поскольку они будут отличаться в зависимости от освещения и материалов, использованных для создания линии.

Табл. 6.1. Диапазоны показаний датчика цвета и поведение робота

Показание датчика цвета	Поведение робота
0–31	Поворот влево
32–72	Прямолинейное движение
73–100	Поворот вправо

Для завершения программы выполни следующие действия:

18. Выбери внешний блок **Переключатель** (Switch) и установи для параметра **Пороговое значение** (Threshold value) нижнее значение диапазона, при котором робот должен двигаться прямо (здесь я выбрал значение 32).

19. Выбери внутренний блок **Переключатель** (Switch) и установи для параметра **Пороговое значение** (Threshold value) нижнее значение диапазона, при котором робот должен повернуть вправо (здесь я указал значение 73).

Программа должна выглядеть так, как показано на рис. 6.9.

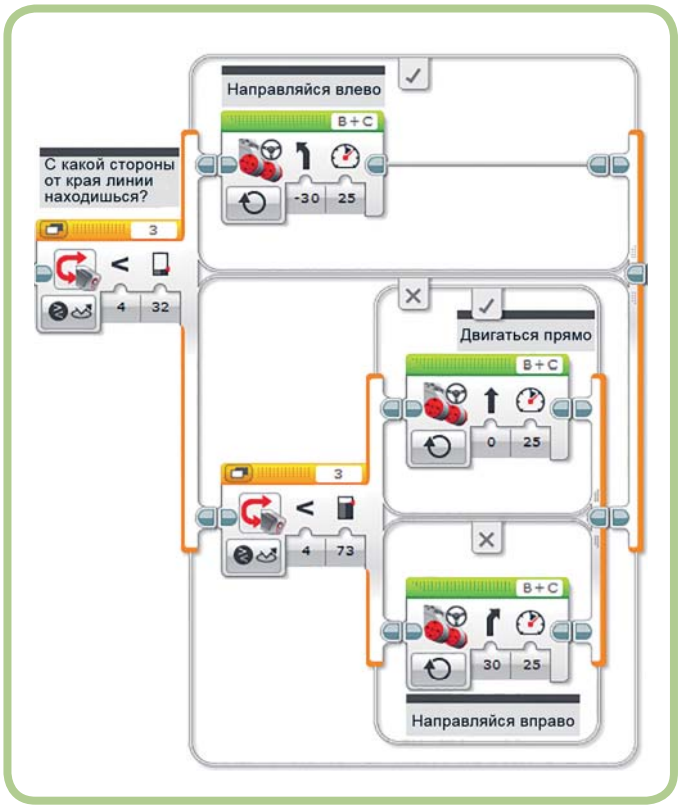


Рис. 6.9. Установка пороговых значений

ПРИМЕЧАНИЕ В этой программе каждый блок **Переключатель** (Switch) считывает показание датчика цвета для принятия решения. Показание датчика считывается дважды, и эти два значения могут оказаться разными. Однако это не должно вызывать проблем, поскольку показание датчика не может сильно измениться за то время, пока модуль EV3 выполняет два блока **Переключатель** (Switch).

Тестирование программы

После запуска программы движение робота по прямой линии станет более ровным. Поэкспериментируй с различными значениями параметров **Пороговое значение** (Threshold value), **Мощность** (Power) и **Рулевое управление** (Steering), чтобы определить максимальную скорость робота TriBot, при которой ему все еще удается не отклоняться от линии.

Использование вида с вкладками

Программа *LineFollower* использует *вложенные* блоки **Переключатель** (Switch), т. е. один блок **Переключатель** (Switch) находится внутри другого. Вложенные блоки **Переключатель** (Switch), как правило, занимают на экране много места, что затрудняет работу с остальной частью программы. Для уменьшения размера блоков **Переключатель** (Switch), выбери **Вид с вкладками** (Tabbed View). Для этого щелкни по кнопке **Вид без вкладок/Вид с вкладками** (Flat/Tabbed View). Теперь программа занимает на экране гораздо меньше места (рис. 6.10).

ПРИМЕЧАНИЕ При использовании блока **Переключатель** (Switch) в режиме **Вид с вкладками** (Tabbed View) лучше помещать все комментарии над самым внешним блоком **Переключатель** (Switch), поскольку ты видишь только одну вкладку за раз, и комментарии внутри блока **Переключатель** (Switch) могут быть скрыты. На рис. 6.10 показан комментарий к коду программы для следования вдоль линии.

ПРАКТИКУМ 6.1

Первая версия программы *LineFollower* при выполнении каждого цикла выбирала одно из двух действий: повернуть налево или повернуть направо. Вторая версия обеспечила более плавное движение робота за счет добавления третьего варианта — двигаться прямо. Усовершенствуй программу, добавив еще два случая, чтобы получилось пять вариантов: резкий поворот влево, плавный поворот влево, движение вперед, плавный поворот вправо и резкий поворот направо. Это можно сделать, заменив блоки **Рулевое управление** (Move Steering), которые заставляют робота поворачивать влево и вправо, на блоки **Переключатель** (Switch), который определяет резкий или плавный поворот в зависимости от показаний датчика цвета.

Программа RedOrBlue

В этом подразделе создадим программу *RedOrBlue*, которая будет распознавать красные и синие объекты. Используем программу *IsItBlue* из гл. 5 в качестве отправной точки (рис. 6.11), поскольку она уже может распознавать синие объекты. Начнем с внесения изменений, позволяющих программе распознавать красные объекты, а затем добавим в нее инструкции на случай, если цвет объекта ни красный и ни синий.

Программа *IsItBlue* была сохранена в проекте *Chapter5*, поэтому первым делом нужно скопировать ее в проект *Chapter6* и переименовать.

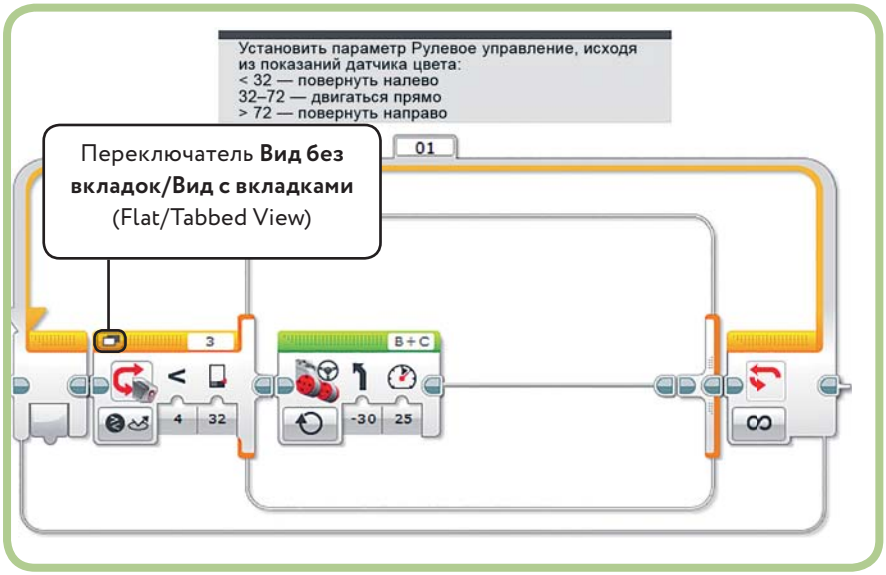


*Рис. 6.10. Отображение блоков **Переключатель** (Switch) в режиме **Вид с вкладками** (Tabbed View)*

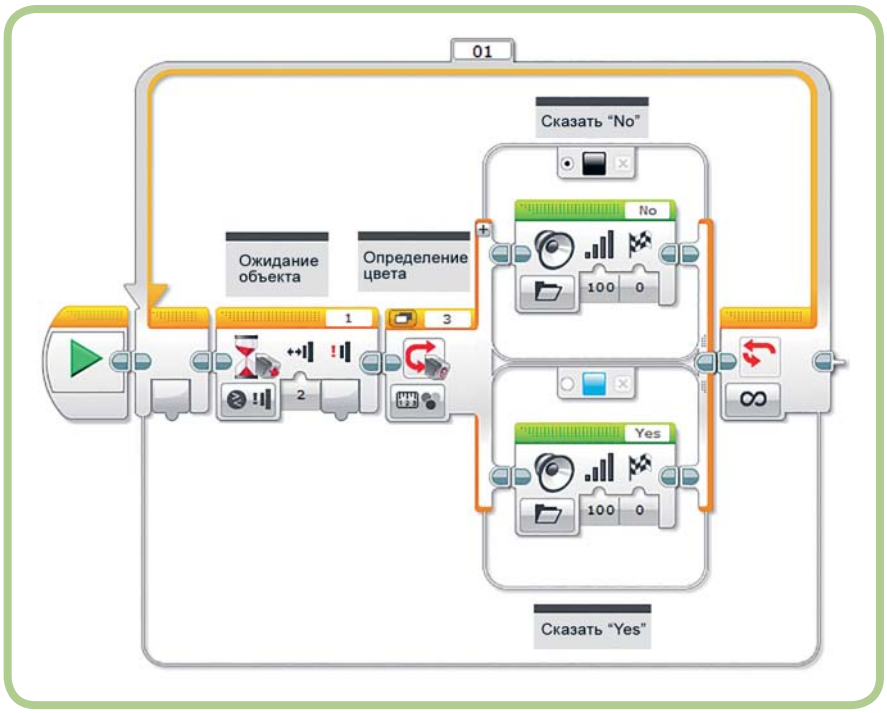


Рис. 6.11. Основа для программы RedOrBlue

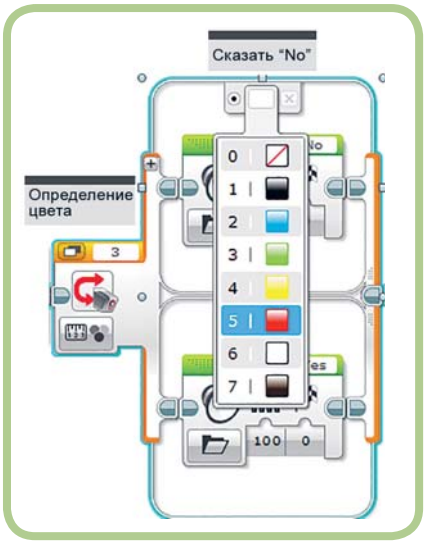


*Рис. 6.12. Выбор варианта **Красный** (Red) для верхнего случая*

1. Открой проект *Chapter6*, если он еще не открыт.
2. Открой проект *Chapter5*.
3. Открой страницу **Свойства проекта** (Project Properties) для проекта *Chapter5*, щелкнув по небольшому значку в виде гаечного ключа в левой части вкладки программы.
4. Выбери *RedOrBlue* в списке программ.
5. Щелкни по кнопке **Копировать** (Copy) в нижней части окна.
6. Выбери проект *Chapter6*.
7. Открой страницу **Свойства проекта** (Project Properties) для проекта *Chapter6*.
8. Щелкни по кнопке **Вставить** (Paste) — и программа *IsItBlue* будет добавлена в проект.
9. Открой программу *IsItBlue* и переименуй ее в *RedOrBlue*.
10. Закрой проект *Chapter5*.

Распознавание красных объектов

Нижний случай блока **Переключатель** (Switch) уже соответствует синим объектам, поэтому сделаем так, чтобы верхний случай соответствовал красным объектам.

1. Щелкни по черному квадрату в верхней части блока **Переключатель** (Switch) и выбери вариант **Красный** (Red), как показано на рис. 6.12.

Ответ типа «Да» или «Нет» уместен в программе *IsItBlue*, однако он не совсем подходит для программы, распознающей бóльшее количество цветов. Измени блоки **Звук** (Sound) так, чтобы программа говорила "Red" («Красный») при распознавании красных объектов и "Blue" («Синий») при распознавании синих.

2. Выбери блок **Звук** (Sound) в верхнем случае и измени звуковой файл на *Red*.
3. Выбери блок **Звук** (Sound) в нижнем случае и измени звуковой файл на *Blue*.

Теперь программа должна выглядеть так, как показано на рис. 6.13. После запуска она должна правильно распознавать красные и синие объекты.

Добавление нового случая

Пока эта программа по-настоящему распознает только синие объекты, объявляя объект любого другого цвета красным (поскольку красный выбран в качестве случая по умолчанию). В этом подразделе ты узнаешь, как изменить программу так, чтобы она правильно распознавала красные объекты и говорила "Uh-oh" в случае невозможности определения цвета объекта. В настоящее время блок **Переключатель** (Switch) предусматривает два случая: для красных объектов и для синих. Кнопка **Добавить случай** (Add Case), обведенная на рис. 6.14, позволяет добавить новый случай в блок **Переключатель** (Switch). Если после добавления случая ты решишь его удалить, щелкни по кнопке с маленьким крестиком (x) справа в верхней части случая.

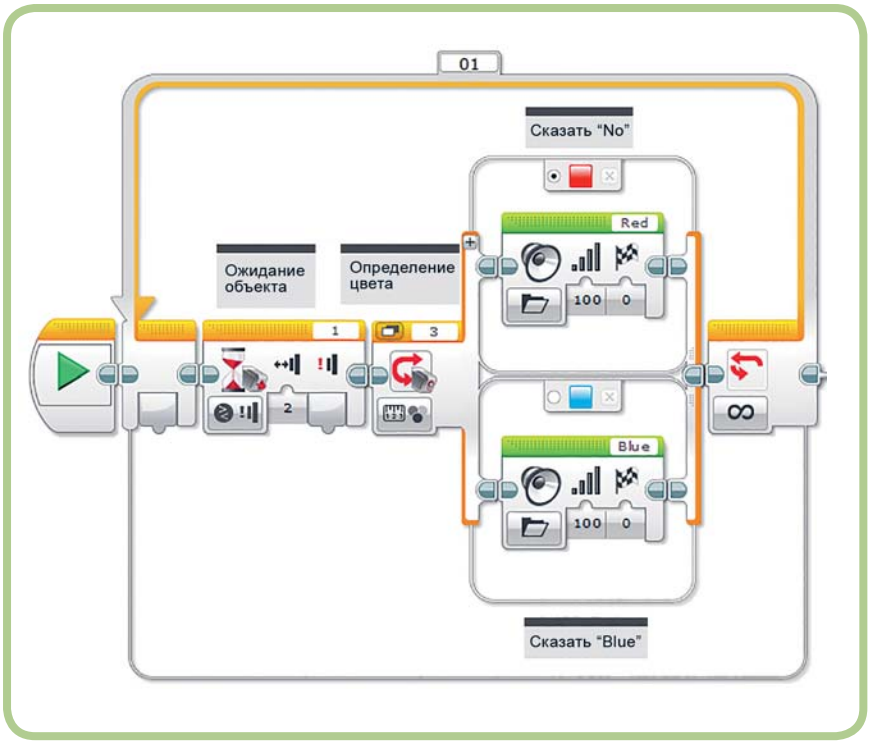


Рис. 6.13. Распознавание красных или синих объектов

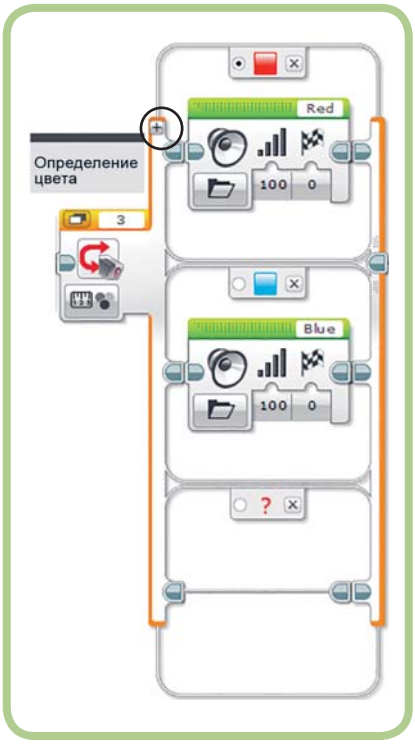


Рис. 6.14. Добавление нового случая

ПРИМЕЧАНИЕ Кнопка **Добавить случай** (Add Case) отображается в блоке **Переключатель** (Switch) только в режимах, предусматривающих более двух возможных случаев.

Теперь вернемся к нашей программе:

4. Щелкни по кнопке **Добавить случай** (Add Case). Блок **Переключатель** (Switch) должен выглядеть, как показано на рис. 6.14.

5. Щелкни по красному знаку вопроса на вкладке в верхней части нового случая, чтобы открыть меню. Выбери вариант **Нет цвета** (No Color).

6. Добавь блок **Звук** (Sound) в случае **Нет цвета** (No Color).

7. Щелкни по полю **Звуковые файлы LEGO** (LEGO Sound Files) и выбери вариант "Uh-oh". Ты найдешь его в папке **Выражения** (Expressions). Теперь программа должна выглядеть так, как показано на рис. 6.15.

Случай по умолчанию

На данный момент случай по умолчанию в нашей программе — это **Красный** (Red). Это означает, что если не выполняется условие ни одного из случаев, программа выполнит блоки, предусмотренные для случая с распознаванием красных объектов, и программа скажет "Red" («Красный»), когда показание датчика цвета будет отличаться от вариантов **Красный** (Red), **Синий** (Blue) и **Нет цвета** (No Color) (например, при распознавании желтого или зеленого объекта). В качестве случая по умолчанию логичнее использовать вариант **Нет цвета** (No Color). Тогда программа будет говорить "Uh-oh" при распознавании любого цвета, кроме красного или синего.

8. Щелкни по кнопке **Случай по умолчанию** (Default Case) в случае **Нет цвета** (No Color).

На рис. 6.16 показана окончательная конфигурация блока **Переключатель** (Switch).

ПРАКТИКУМ 6.2

Доработай программу *RedOrBlue* так, чтобы она распознавала все семь цветов, которые может обнаружить датчик цвета.

Блок «Цикл»

Блок **Цикл** (Loop) позволяет многократно выполнять группу блоков. Блоки внутри блока **Цикл** (Loop) называются *телом цикла*. Надо задать условие, определяющее, как часто повторяется тело цикла и когда программа запускает блок, следующий после блока **Цикл** (Loop).

Блок **Цикл** (Loop) предусматривает те же режимы датчиков, что и блок **Переключатель** (Switch), плюс четыре дополнительных режима (рис. 6.17):

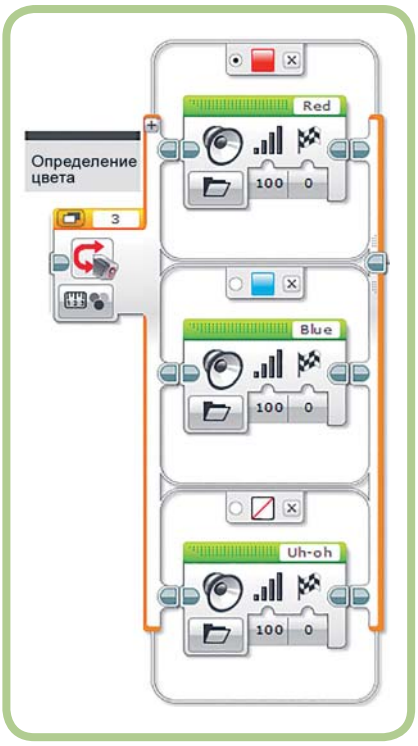


Рис. 6.15. Случай ***Нет цвета*** *(No Color)*

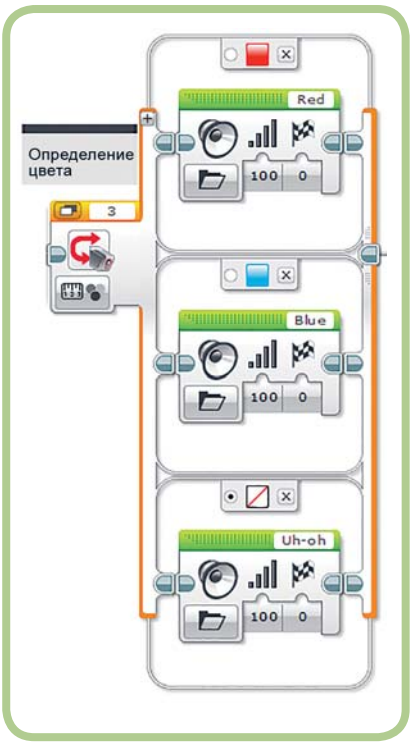


Рис. 6.16. Определение случая ***Нет цвета*** *(No Color) в качестве случая по умолчанию*

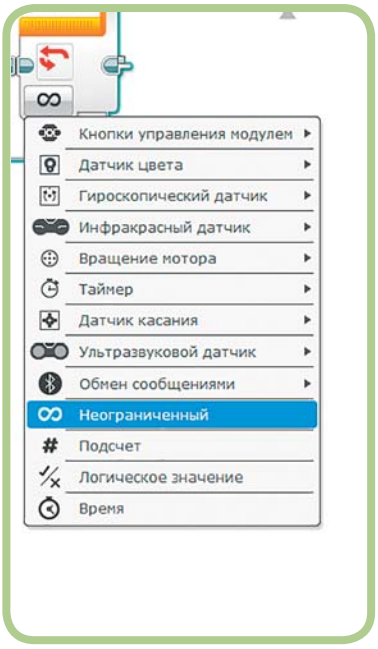


Рис. 6.17. Выбор режима блока ***Цикл*** *(Loop)*

Неограниченный (Unlimited), Цикл повторяется до завершения программы или до выхода из цикла, так как используется блок **Прерывание цикла** (Loop Interrupt) (о ней — далее в этой главе).
Подсчет (Count). Цикл повторяется указанное количество раз.
Логическое значение (Logic). Значение, переданное в блок **Цикл** (Loop) с помощью шины данных, определяет условие выхода из цикла. (с гл. 10 речь пойдет о применении блока **Цикл** (Loop) с шинами данных).
Время (Time). Цикл выполняется в течение указанного количества секунд.

Когда для блока **Цикл** (Loop) выбран режим, предполагающий использование датчика, программа проверяет его показание после выполнения тела цикла (тело цикла всегда выполняется хотя бы один раз). Затем в зависимости от показания датчика программа решает, повторить цикл или выйти из него. Не имеет значения, что показывает датчик во время выполнения тела цикла; на программу влияет только показание датчика, полученное после его выполнения. Если ты не будешь соблюдать осторожность, это может привести к сбою в программе. Поскольку показание датчика цвета проверяется только в конце цикла, очень вероятно, что робот переедет линию, не обнаружив ее, пока выполняется блок **Звук** (Sound) или **Ожидание** (Wait). В качестве примера на рис. 6.18 показана некорректная версия программы *LineFinder*, которая часто не позволяет обнаружить линию.

Рис. 6.18. *Некорректная версия программы* *LineFinder*

Блок «Прерывание цикла»

Блок **Прерывание цикла** (Loop Interrupt), изображенный на рис. 6.19, обеспечивает еще один способ выхода из цикла. Этот блок имеет всего один параметр — имя цикла, из которого требуется выйти. В верхней части каждого блока **Цикл** (Loop) находится вкладка с его именем (рис. 6.20). По умолчанию циклу присваивается имя *01*; для изменения имени щелкни по нему.

*Рис. 6.19. Блок **Прерывание цикла** (Loop Interrupt)*

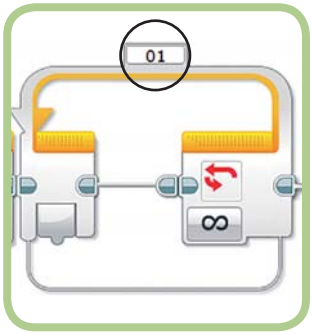


*Рис. 6.20. Имя блока **Цикл** (Loop)*

В блоке **Прерывание цикла** (Loop Interrupt) отображаются только первые три буквы имени блока **Цикл** (Loop), который необходимо прервать. Чтобы не возникло путаницы, для имен циклов лучше использовать числа или сокращения; в противном случае может быть неясно, к какому циклу относится блок **Прерывание цикла** (Loop Interrupt).

Программа BumperBot3

Теперь ты внесешь некоторые изменения в программу *BumperBot2*, чтобы узнать, как работает блок **Прерывание цикла** (Loop Interrupt). Для использования этой программы требуется восстановить исходную конфигурацию робота TriBot, установив на его переднюю часть бампер с датчиком касания и прикрепив сбоку датчик цвета, как показано на рис. 6.21.

Внесенные изменения обеспечат выход из цикла и остановят выполнение программы при выключении света в комнате. На рис. 6.22 показан фрагмент исходной программы, который будет изменен. До сих пор программа заставляла робота двигаться вперед и ждать нажатия кнопки датчика касания. После нажатия кнопки робот откатывался назад, и программа возвращалась к началу.

Теперь ты изменишь программу так, чтобы в процессе ожидания робот проверял показание датчика цвета. Если это значение очень низкое (что говорит о выключенном освещении), блок **Прерывание цикла** (Loop Interrupt) обеспечит выход из цикла, и программа завершится, поскольку после блока **Цикл** (Loop) нет других блоков.

Рис. 6.21. Конфигурация робота TriBot для выполнения программы BumperBot3

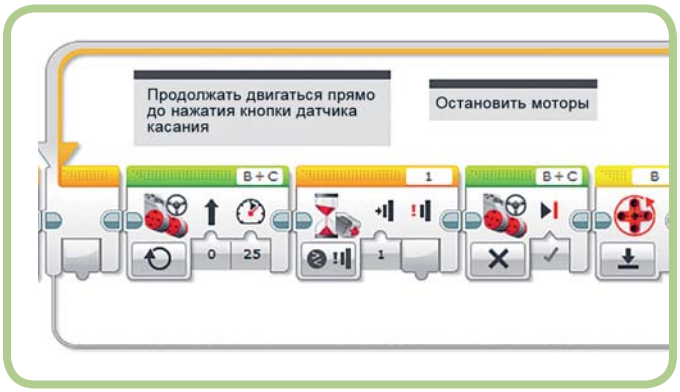


Рис. 6.22. Программа BumperBot2

Для обнаружения столкновения с бампером вместо блока **Ожидание** (Wait) используем блок **Цикл** (Loop), который будет повторяться вплоть до нажатия кнопки датчика касания (рис. 6.23). В теле цикла используем блок **Переключатель** (Switch) в режиме **Яркость внешнего освещения** (Ambient Light Intensity), чтобы проверить уровень освещенности комнаты. Если датчик цвета не обнаружит достаточного количества света, программа остановит моторы, скажет "Goodbye" («До свидания»), а затем выйдет из цикла.

Для создания новой программы *BumperBot3*, показанной на рис. 6.23, выполни следующие действия:

1. Скопируй программу *BumperBot2* из проекта *Chapter5* в проект *Chapter6*.
2. Измени имя программы на *BumperBot3*.
3. Измени имя основного блока **Цикл** (Loop) на *02*.
4. Удали блок **Ожидание** (Wait).

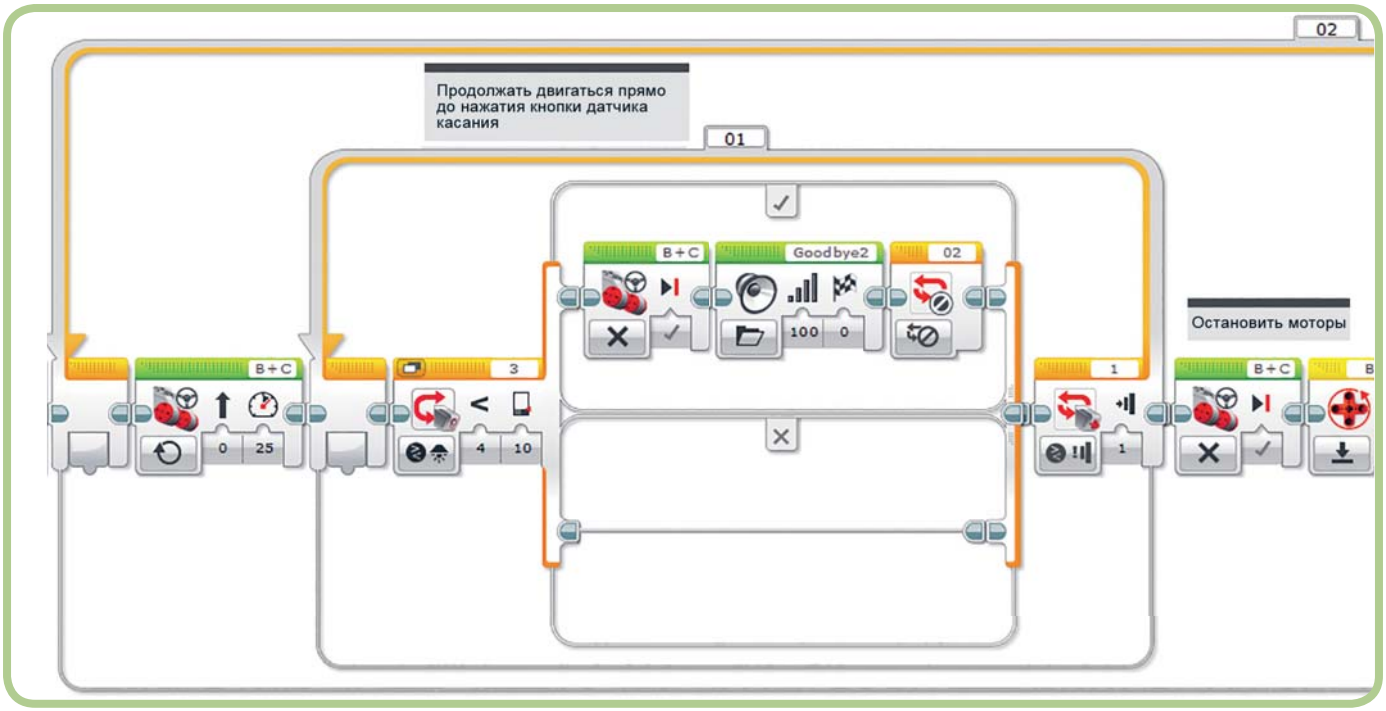


Рис. 6.23. Программа BumperBot3

5. Добавь блок **Цикл** (Loop) на место удаленного блока **Ожидание** (Wait).

6. Для блока **Цикл** (Loop) выбери режим **Датчик касания** (Touch Sensor) ⇒ **Сравнение** (Compare) ⇒ **Состояние** (State).

Эта часть программы теперь должна выглядеть, как на рис. 6.24.

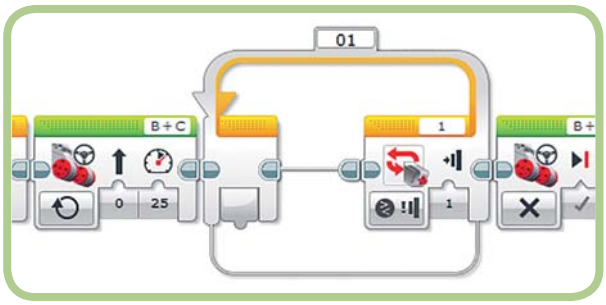


Рис. 6.24. *Замена блока* **Ожидание** *(Wait) блоком* **Цикл** *(Loop)*

На данном этапе пустой блок **Цикл** (Loop) делает то же самое, что и блок **Ожидание** (Wait), место которого он занял: он просто ждет нажатия кнопки датчика касания. Однако теперь в блок **Цикл** (Loop) можно добавить несколько блоков, которые будут выполняться, пока программа находится в режиме ожидания.

Затем добавим блок **Переключатель** (Switch) в блок **Цикл** (Loop) для проверки показания датчика цвета:

7. Перетащи блок **Переключатель** (Switch) в блок **Цикл** (Loop).

8. Выбери для блока **Переключатель** (Switch) режим **Датчик цвета** (Color Sensor) ⇒ **Сравнение** (Compare) ⇒ **Яркость внешнего освещения** (Ambient Light Intensity).

9. Для параметра **Пороговое значение** (Threshold value) задай значение **10**. Это значение можно будет скорректировать после тестирования, если оно окажется слишком высоким или слишком низким. Блок **Цикл** (Loop) должен выглядеть, как на рис. 6.25.

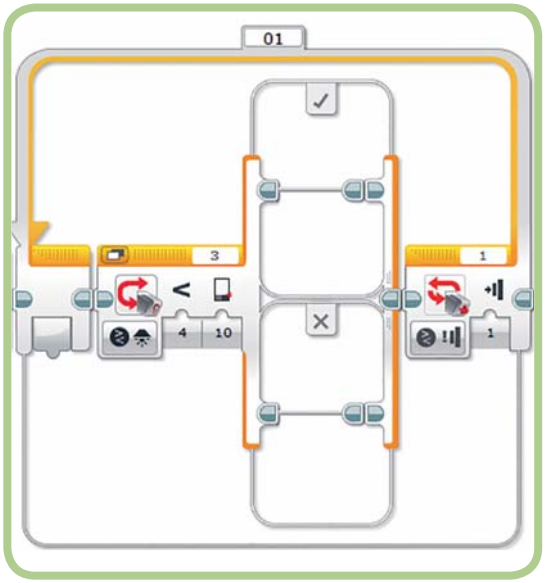


Рис. 6.25. Добавление блока ***Переключатель*** *(Switch)*

После запуска блок **Переключатель** (Switch) проверяет показание датчика цвета, и, если интенсивность света

не превышает 10, выполняется блок, предусмотренный для верхнего случая. Теперь добавь следующие блоки:

10. Перетащи блок **Рулевое управление** (Move Steering) в верхний случай блока **Переключатель** (Switch) и выбери для него режим **Выключить** (Off).

11. Добавь блок **Звук** (Sound) после блока **Рулевое управление** (Move Steering). Выбери вариант *Goodbye* в списке звуковых файлов **Звуковые файлы LEGO** (LEGO Sound Files) ⇒ **Связь** (Communications).

12. Добавь блок **Прерывание цикла** (Loop Interrupt) после блока **Звук** (Sound). В качества имени цикла введи *02*.

На рис. 6.26 показан блок **Цикл** (Loop) с внесенными изменениями.

Теперь после запуска программа должна сказать "Goodbye" и завершиться при выключении света в помещении. Чтобы протестировать программу, ты можешь взять робота TriBot в одну руку, и когда колеса начнут двигаться, закрыть датчик цвета другой рукой. При этом показание датчика цвета уменьшится. После этого программа должна сказать "Goodbye" и завершиться.

Дальнейшее исследование

Изучив нюансы блоков **Переключатель** (Switch) и **Цикл** (Loop), ты можешь попрактиковаться в их использовании:

1. Напиши программу, заставляющую робота следовать за тобой. При этом он должен находиться от тебя на расстоянии 30–60 см. Используй инфракрасный или ультразвуковой датчик и блок **Переключатель** (Switch), чтобы робот двигался вперед или назад, если ты окажешься слишком далеко или слишком близко. Робот не должен двигаться, если ты находишься в пределах допустимого расстояния.

2. Используй удаленный инфракрасный маяк для добавления в программу *BumperBot3* функций паузы и возобновления. Добавь новый блок **Переключатель** (Switch), проверяющий, нажата ли кнопка маяка. Если нажата, программа должна остановить моторы, дождаться нажатия другой кнопки на маяке, а затем снова запустить моторы. Первая кнопка действует как кнопка паузы, а вторая — как кнопка возобновления.

Заключение

В этой главе рассмотрен блок **Переключатель** (Switch) и его применение для принятия решений (это позволяет делать выбор между двумя или более группами блоков). Ты можешь увеличить количество вариантов с помощью вложенных блоков **Переключатель** (Switch), или добавив дополнительных случаев в блок **Переключатель** (Switch), представленный в режиме **Вид с вкладками** (Tabbed View).

Другим важным блоком управления операторами является блок **Цикл** (Loop), позволяющий повторять группу блоков до тех пор, пока не будет выполнено определенное условие. Большая часть возможностей программ EV3 обеспечивается гибкостью, достигаемой за счет настройки блоков. Как правило, при проверке условия показание датчика сравнивается с заданным пороговым значением, однако ты можешь настроить блок **Цикл** (Loop) так, чтобы он выполнялся определенное число раз или в течение заданного периода времени. Блок **Прерывание цикла** (Loop Interrupt) обеспечивает другой способ выхода из блока **Цикл** (Loop), что обеспечивает еще большую гибкость в плане управления работой программы.

Ты уже знаешь, как использовать моторы и датчики EV3, и имеешь представление о блоках программирования, необходимых для создания сложных программ. В гл. 7 ты применишь эти знания для того, чтобы запрограммировать робота TriBot на нахождение выхода из лабиринта.

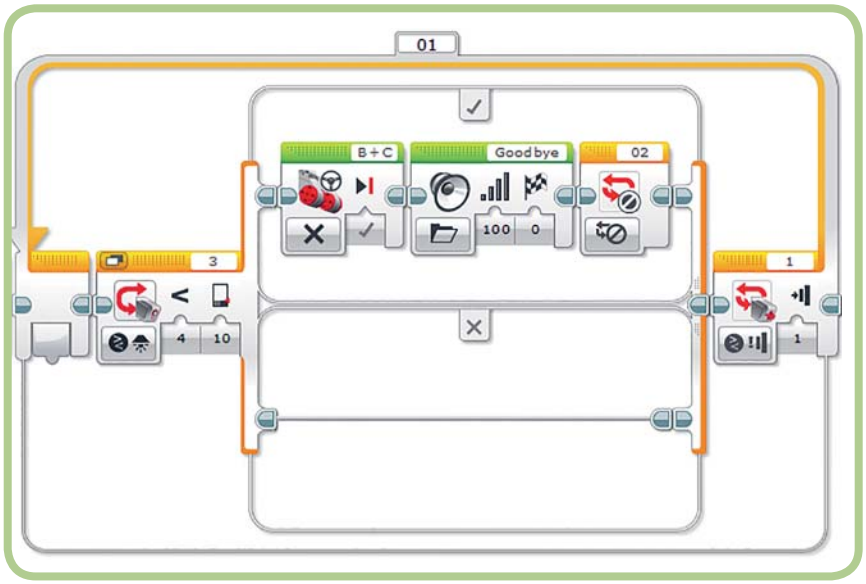


Рис. 6.26. Программа останавливает моторы и говорит "Goodbye"

7

Программа WallFollower: путешествие по лабиринту

В этой главе создадим программу *WallFollower*, которая позволит роботу TriBot найти выход из простого лабиринта. Ты узнаешь все шаги, необходимые для составления плана и написания программы, начиная от разработки и заканчивая окончательным тестированием. К ним относятся: выбор начальных параметров определенных блоков, оценка качества их работы и при необходимости настройка этих параметров, поскольку программы почти никогда не работают правильно при первом запуске. Начнем с изучения того, как читать и писать псевдокод.

Псевдокод

Чем сложнее программа, тем труднее дать короткое и точное ее описание в виде обычных предложений и абзацев. Существуют более эффективные способы описания логики программы, один из наиболее распространенных — псевдокод.

Псевдокод можно использовать для описания наиболее важных аспектов программы и ее логики. Это позволяет легко делиться своими программами с людьми или создавать программы EV3 на основе псевдокода, написанного кем-то другим. Обычно псевдокод напоминает традиционные текстовые языки программирования, вроде Java или C, однако не подразумевает следование строгим правилам.

Например, на рис. 7.1 показана программа *RedOrBlue* из гл. 6. Она находится в ожидании нажатия кнопки датчика касания, а затем использует датчик цвета, чтобы определить, является ли исследуемый объект синим или красным. В листинге 7.1 приведен псевдокод для этой программы.

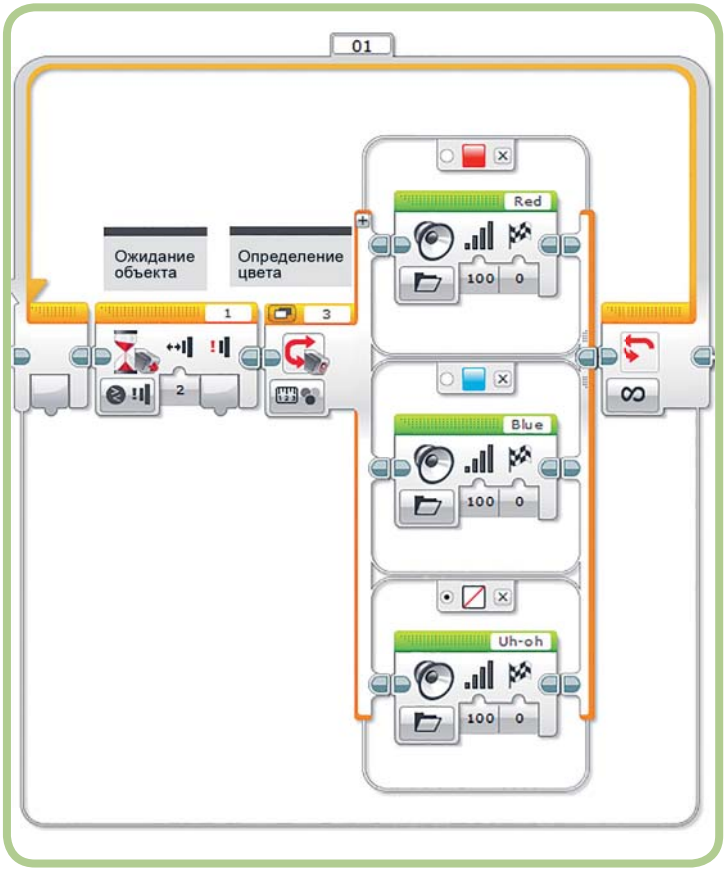


Рис. 7.1. Программа RedOrBlue

Листинг 7.1. Псевдокод для программы RedOrBlue

```
begin loop
   wait for the Touch Sensor to be pressed
   if Color Sensor detects red then
      use a Sound block to say "Red"
   else if the Color Sensor detects blue then
      use a Sound block to say "Blue"
   else
      use a Sound block to say "Uh-oh"
   end if
loop forever
```

Этот псевдокод представляет собой краткое и простое для понимания описание программы. Немного попрактиковавшись, ты быстро привыкнешь к чтению псевдокода и его преобразованию в рабочую программу EV3.

У листинга 7.1 есть особенности:

- В большинстве случаев для каждого блока используется отдельная строка.
- Строки имеют отступ, обозначающий то, что блоки вложены один в другой. Отступы позволяют легче понять, что происходит внутри блока, например, **Цикл** (Loop) или **Переключатель** (Switch).
- Термины `if`, `then` и `else` используются для описания поведения блока **Переключатель** (Switch). Операторы `if/then` применяются во многих языках программирования, и с их помощью можно легко описать логику программы на простом английском языке.
- Строка с отступом под первым оператором `if` содержит описание блоков в верхнем случае блока **Переключатель** (Switch), а строка с отступом под оператором `else if` — описание блоков в среднем случае. Строка под оператором `else` описывает случай по умолчанию блока **Переключатель** (Switch). Случай по умолчанию следует описывать в последнюю очередь вне зависимости от его местоположения в блоке **Переключатель** (Switch). По сути, это означает: «Если первый случай истинен, выполни этот набор действий. Если истинен второй случай, выполни другой набор действий. В противном случае (`else`) выполни последний набор действий».
- Строка `end if` отмечает окончание блока **Переключатель** (Switch), после чего программа переходит к следующей части кода.

В данном примере псевдокод описывает готовую программу со всеми деталями. Однако псевдокод часто используется на этапе планирования и разработки программы, и в этом случае в нем могут отсутствовать некоторые подробности.

Теперь начнем составлять план программы *WallFollower*. После определения того, что должна делать наша программа, можно написать псевдокод, который поможет продумать всю программу.

Нахождение выхода из лабиринта

Существует множество способов нахождения выхода из лабиринта. Для этой программы будем использовать алгоритм *правило правой руки*. *Алгоритм* представляет собой набор инструкций для решения задачи. В данном случае *правило правой руки* подразумевает одну главную инструкцию: всегда следуй вдоль правой стены и входи в любой находящийся справа проход.

Алгоритм, основанный на *правиле правой руки*, подходит для прохождения лабиринтов без туннелей и мостов, когда начальная и конечная точки находятся на границе лабиринта. Однако это правило не всегда подходит для прохождения лабиринта, когда цель заключается в нахождении его центра. Если лабиринт соответствует этим критериям, робот обязательно найдет из него выход.

На рис. 7.2 показан пример лабиринта, путь по которому проложен с помощью *правила правой руки*. Чтобы понять этот алгоритм, представь, что ты идешь по данному лабиринту, все время касаясь рукой правой стены. В этом случае ты будешь следовать по отмеченному на рисунке пути и в конце концов найдешь выход. Этот метод не всегда поможет найти кратчайший путь через лабиринт, но благодаря ему ты найдешь выход из него.

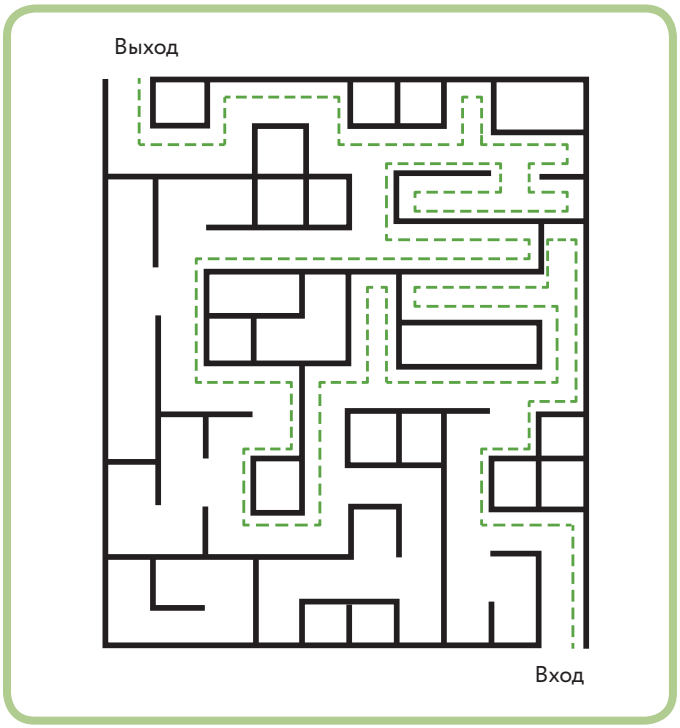


Рис. 7.2. Путь через простой лабиринт, определенный с помощью правила правой руки

Требования к программе

Первым делом надо составить простой список *требований к программе*, которые точно описывают ее действия. В случае соответствия перечисленным в этом списке требованиям наша программа должна успешно справляться с поставленной задачей.

Перед составлением списка требований подумай о различных ситуациях, с которыми предстоит столкнуться роботу, и реши, как программа должна действовать в каждой из них. Например, если стена находится справа от робота, как показано на рис. 7.3, он должен продолжать двигаться прямо. По мере продвижения вперед робот должен поддерживать постоянное расстояние между собой и стеной, не слишком сильно от нее отклоняясь и не врезаясь в нее.

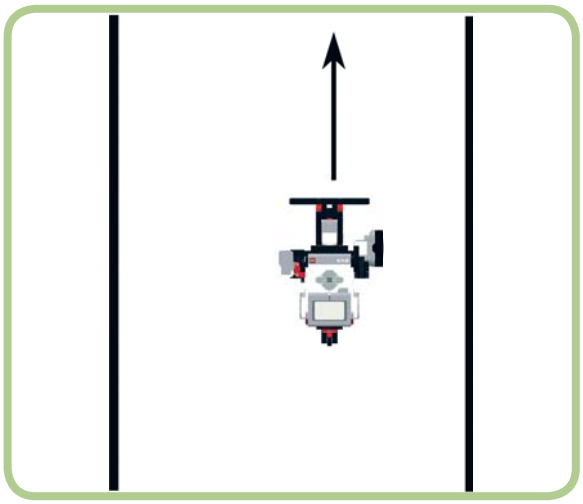


Рис. 7.3. Движение робота параллельно стене справа

Когда робот попадает в угол (т. е. когда справа и спереди оказывается стена), как показано на рис. 7.4, робот должен повернуться влево и продолжить движение вперед.

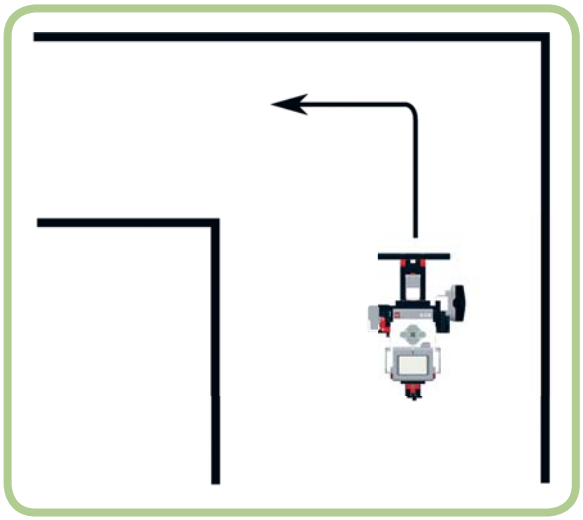


Рис. 7.4. Поворот влево на углу

Когда в стене справа появляется проход (рис. 7.5), робот должен в него завернуть. Следуя *правилу правой руки*, робот *всегда* должен заворачивать в проход справа, даже если он может ехать прямо, как показано на рис. 7.6, или когда он может свернуть влево, как отражено на рис. 7.7.

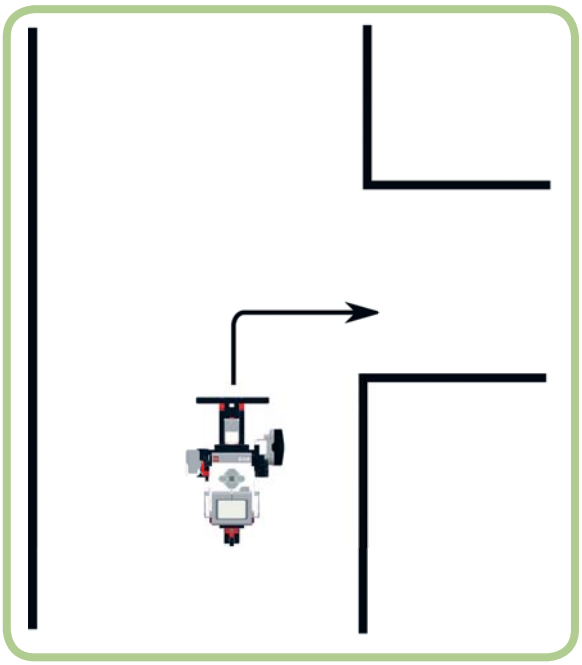


Рис. 7.5. Поворот в проход справа

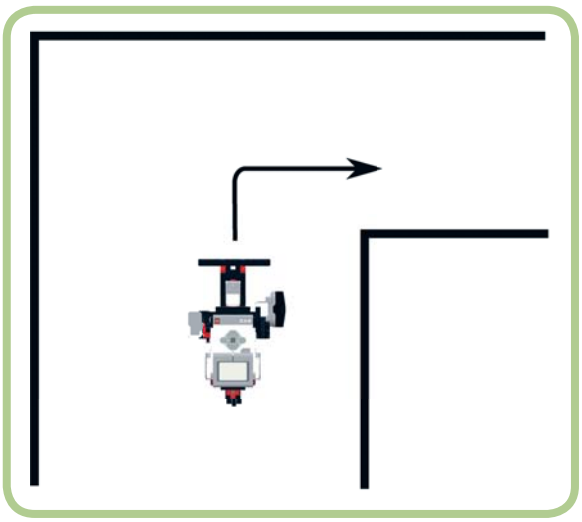


Рис. 7.6. Поворот направо вместо движения по прямой

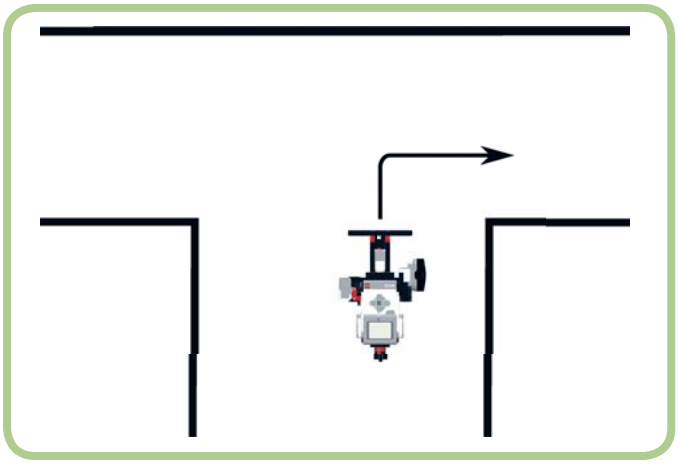


Рис. 7.7. Поворот направо вместо поворота налево

Учитывая эти ситуации, можно составить список требований к программе в виде трех утверждений:

— Робот TriBot должен двигаться вперед вдоль правой стены, держась близко к ней.
— Если впереди и справа от робота находится стена, он должен повернуть налево на 90°, а затем следовать вдоль новой стены.
— Если справа от робота оказывается проход в стене, он должен повернуть направо на 90° и въехать в этот проход.

Допущения

При составлении списка требований к программе также рекомендуется перечислить любые допущения или ограничения. Это поможет решить, какие условия следует протестировать, а какие можно проигнорировать. Относительно этой программы можно сделать четыре допущения:

— Стены лабиринта прямые.
— Все проходы достаточно большие, чтобы робот TriBot мог в них проехать.
— Стены соединяются под прямым углом (это упростит код для поворота за угол).
— В начале программы стена будет находиться справа от робота.

Последний пункт этого списка, в котором описывается расположение робота при запуске программы, называется *начальным условием*. Гораздо проще запустить программу, предварительно поместив робота в нужное место, чем заставлять его бродить в поисках входа в лабиринт.

Обдумывание этих допущений до составления программы позволит определить, какие проблемы предстоит решить, а какие можно игнорировать. Вместе со списком требований это поможет понять, чего ожидать от программы.

Хорошо, если окончательная программа окажется более функциональной, чем планировалось изначально! Например, согласно одному из допущений, стены лабиринта прямые, однако ты можешь создать программу, которая успешно справляется и с изогнутыми стенами.

Начальный этап разработки

Теперь нужно выяснить, как заставить робота решить поставленные перед ним задачи. Первая задача заключается в том, чтобы двигаться вдоль стены, находясь от нее на небольшом расстоянии. Для определения этого расстояния можно использовать инфракрасный или ультразвуковой датчик. Чтобы заставить робота приблизиться или отъехать от стены, можно использовать блоки **Рулевое управление** (Move Steering).

Поскольку стена будет находиться сбоку от робота во время его движения, инфракрасный датчик необходимо установить так, чтобы он был направлен в сторону (а не вперед). Следуй инструкциям в разделе «Альтернативное расположение ультразвукового или инфракрасного датчика» в гл. 3, чтобы установить датчик, как показано на рис. 7.8.

Рис. 7.8. Робот TriBot с инфракрасным датчиком, направленным в сторону

Теперь надо заставить робота правильно реагировать на ситуацию, когда перед ним оказывается стена (см. рис. 7.4, 7.5). Здесь будем использовать датчик касания. При столкновении робота со стеной будет нажата кнопка датчика касания, после чего робот должен будет откатиться назад, совершить поворот влево на 90° и следовать вдоль стены, с которой он только что столкнулся. Вспомни, программа *BumperBot* обеспечивает аналогичное поведение, у тебя должно быть хорошее представление о том, как это будет работать.

Наконец, нам нужно, чтобы наша программа обнаруживала и реагировала на проходы в стене справа от робота. Мы будем использовать инфракрасный датчик, чтобы обнаружить проход, мимо которого проезжает робот. Если в процессе следования вдоль стены показание датчика внезапно увеличится, это будет означать, что он проезжает мимо прохода.

Поместим всю программу в блок **Цикл** (Loop), чтобы робот TriBot продолжал движение до тех пор, пока ты вручную не завершишь выполнение программы.

Рассмотрев задачи высокого уровня, самое время написать псевдокод для описания работы программы (см. листинг 7.2). В следующем листинге приводится краткое изложение шагов, описанных выше и продемонстрированных на рис. 7.5–7.7. Поскольку мы находимся на ранней стадии разработки программы, этот листинг охватывает только основные моменты. В следующих разделах разработаем каждую часть программы EV3.

Листинг 7.2. Первоначальный вариант программы WallFollower в форме псевдокода

```
begin loop
   if too close to the wall (use Infrared Sensor)
      then drive forward, steering away
      from the wall
   else
      drive forward, steering toward the wall
   end if
   if Touch Sensor is pressed then
      back up a little to get room to turn around
      spin one quarter-turn to the left
   end if
   if an opening is detected (by the Infrared
      Sensor) on the right then
      spin one quarter-turn toward the opening
   end if
loop forever
```

Следование вдоль прямой стены

Первый раздел кода EV3 заставит робота TriBot двигаться вдоль стены. Ты будешь использовать блок **Переключатель** (Switch) для выбора одного из двух блоков **Рулевое управление** (Move Steering): заставляющим робота двигаться вперед, приближаясь к стене, и указывающим роботу двигаться вперед, отдаляясь от стены. Этот блок должен сделать так, чтобы робот TriBot двигался вперед, постоянно находясь на расстоянии от стены. Этот подход был применен в версии программы *LineFollower* (см. гл. 6), и ты довольно часто будешь обращаться к нему при использовании датчика для управления мотором.

ИСПОЛЬЗОВАНИЕ ОБРАЗОВАТЕЛЬНОЙ ВЕРСИИ КОНСТРУКТОРА

Инфракрасный датчик (из домашней версии конструктора) и ультразвуковой датчик (из образовательной версии конструктора) могут измерять расстояние от робота до стены, поэтому можно использовать для этой программы любой из них. Единственным отличием будет пороговое значение, которое используется в программе. Представленные в остальной части этой главы значения и инструкции относятся к инфракрасному датчику. Об ультразвуковом датчике будет скаазно, только когда тебе нужно будет сделать с ним что-то другое.

Еще одно существенное различие между домашней и образовательной версиями конструктора заключается в размере колес. Этот параметр влияет на расстояние, заданное в блоках **Рулевое управление** (Move Steering). Далее будем использовать колеса из домашней версии при составлении инструкций, рассмотрим любые изменения, которые необходимо внести для использования колес из образовательной версии.

Написание кода

Для написания этой части кода сначала определим нужное расстояние между роботом TriBot и стеной. Робот должен находиться близко к стене, но при этом иметь достаточно места, чтобы выполнить поворот при езде в угол. Для получения разумного начального значения можно использовать вкладку **Представление порта** (Port View) и выполнить следующие действия:

1. Поставь робота TriBot на пол рядом со стеной так, чтобы инфракрасный датчик был направлен в сторону стены. Убедись, что для совершения роботом полного оборота достаточно места (рис. 7.9).

2. Открой вкладку **Представление порта** (Port View) в программном обеспечении EV3 или на модуле EV3 и обрати внимание на показания инфракрасного датчика.

В моем случае датчик показывает значение 7, которое будет использовано в следующих инструкциях. Помни о том, что инфракрасный датчик измеряет расстояние неточно, поэтому твое значение может отличаться от указанного в этой книге.

Теперь можно приступить к написанию программы. Начнем с объединения блоков в соответствии с приведенными ниже инструкциями, а затем после проведения тестирования уточним значения параметров:

1. Создай новый проект под названием *Chapter7*.

2. Создай новую программу под названием *WallFollower*.

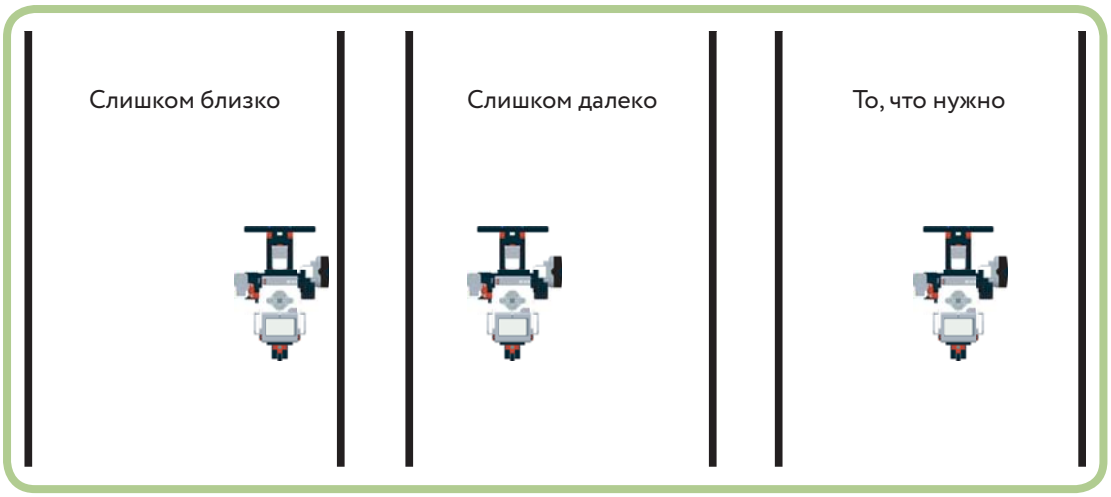


Рис. 7.9. Размещение робота TriBot у стены

3. Перетащи в программу блок **Цикл** (Loop). Используй выбранный по умолчанию режим **Неограниченный** (Unlimited).

4. Перетащи блок **Переключатель** (Switch) в блок **Цикл** (Loop).

5. Выбери режим **Инфракрасный датчик** (Infrared Sensor) ⇒ **Сравнение** (Compare) ⇒ **Приближение** (Proximity).

6. Для параметра **Пороговое значение** (Threshold value) задай значение, которое было определено ранее (я указал 7).

ПРИМЕЧАНИЕ Если ты используешь ультразвуковой датчик, для блока **Переключатель** (Switch) выбери режим **Ультразвуковой датчик** (Ultrasonic Sensor) ⇒ **Сравнение** (Compare) ⇒ **Расстояние в сантиметрах (или дюймах)** (Distance Inches (or Centimeters)). Для параметра **Пороговое значение** (Threshold value) задай 13 см (около 5 дюймов).

Блок **Переключатель** (Switch) должен выглядеть, как показано на рис. 7.10.

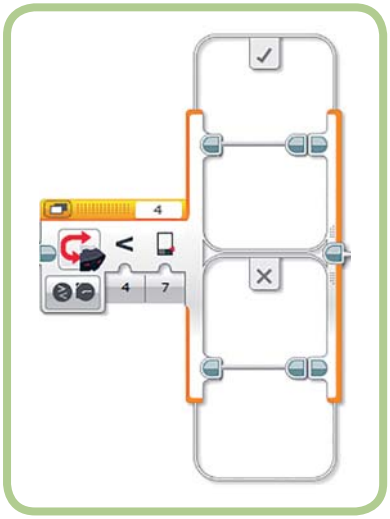


*Рис. 7.10. Блок **Переключатель** (Switch) с учетом расстояния от стены*

7. Перетащи блок **Рулевое управление** (Move Steering) в верхний случай блока **Переключатель** (Switch).

8. Выбери режим **Включить** (On), чтобы моторы вращались во время корректировки параметра **Рулевое управление** (Steering) при каждом выполнении цикла.

Блок **Рулевое управление** (Move Steering) запустится, когда показание инфракрасного датчика будет меньше порогового значения, значит расстояние между роботом и стеной слишком маленькое. В этом случае робот должен отъехать от стены, свернув влево. Для выполнения небольшого поворота начни со значения –10 для параметра **Рулевое управление** (Steering).

9. Задай для параметра **Рулевое управление** (Steering) значение **–10**.

Добавь такой же блок в нижний случай, чтобы приблизить робота к стене. Для этого используй положительное значение параметра **Рулевое управление** (Steering).

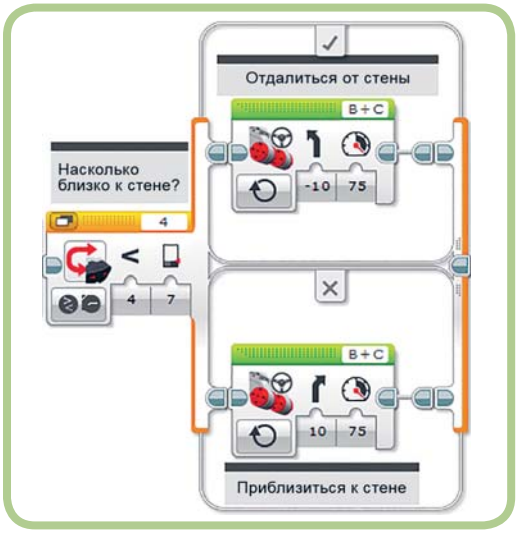


Рис. 7.11. *Следование вдоль стены*

10. Перетащи блок **Рулевое управление** (Move Steering) в нижний случай блока **Переключатель** (Switch).

11. Выбери для него режим **Включить** (On), а для параметра **Рулевое управление** (Steering) задай значение **10**.

Благодаря такому блоку **Переключатель** (Switch) (рис. 7.11) робот TriBot сможет двигаться вдоль стены. Следующим шагом будет тестирование и при необходимости корректировка программы.

Тестирование

Теперь проведем тестирование, чтобы оценить качество работы первой части программы. Затем можно будет внести некоторые корректировки для улучшения программы. С первой попытки трудно правильно настроить параметры, поэтому важно их протестировать и при необходимости скорректировать программу.

Для того чтобы протестировать программу *WallFollower*, тебе потребуется часть стены с углом и проходом, однако ты можешь использовать весь лабиринт (что намного интереснее). Стены лабиринта должны быть достаточно высокими, чтобы их мог обнаружить инфракрасный датчик. Коробки, стопки книг или доски хорошо подойдут для создания стен лабиринта. Его можно собрать даже из деталей конструкторов LEGO!

При тестировании программы проверь, не врезается ли робот в стену и не отходит ли слишком далеко от нее. Такое поведение может говорить об очень высоком значении параметра **Мощность** (Power), из-за которого робот движется слишком быстро и не успевает вовремя скорректировать свой маршрут. Или настройки рулевого управления не позволяют ему выполнить достаточно резкий поворот, чтобы поддерживать нужное расстояние до стены.

Настроим оба эти параметра для повышения надежности данной части программы. Во-первых, нужно замедлить движение робота, чтобы облегчить настройку рулевого управления.

12. Задай значение **25** для параметра **Мощность** (Power) в обоих блоках **Рулевое управление** (Move Steering).

Уменьшение скорости робота TriBot должно значительно улучшить ситуацию, однако можно сделать еще больше, скорректировав значение параметра **Рулевое управление** (Steering), чтобы контролировать резкость поворотов робота. Ты можешь вводить разные значения параметра **Рулевое управление** (Steering), чтобы посмотреть, как они влияют на движение робота, и не забудь изменить значение в обоих блоках **Рулевое управление** (Move Steering). В табл. 7.1 приведены результаты моего тестирования.

Исходя из своих результатов, предлагаю задать для параметра **Рулевое управление** (Steering) значение **20**.

13. Задай значение **–20** для параметра **Рулевое управление** (Steering) в верхнем случае.

14. Задай значение **20** для параметра **Рулевое управление** (Steering) в нижнем случае.

Табл. 7.1. Результаты тестирования значений параметра **Рулевое управление**

Значение параметра «Рулевое управление»	**Результат**
10	Сначала робот TriBot движется нормально, но в конце концов врезается в стену и не корректирует свой маршрут достаточно быстро.
20	Робот TriBot остается на близком расстоянии от стены, не врезаясь в нее. Движение недостаточно плавное, но в целом все хорошо.
30	Робот TriBot остается на близком расстоянии от стены, не врезаясь в нее. Однако он движется очень неровно, часто отклоняясь из стороны в сторону.

На рис. 7.12 показаны внесенные в программу изменения. На этом этапе робот TriBot должен легко двигаться вдоль прямой стены без углов или проходов.

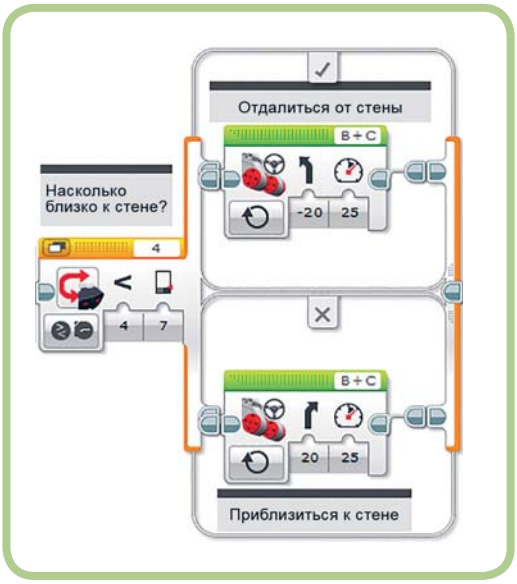


Рис. 7.12. Параметры для следования вдоль стены после тестирования

Поворот за угол

В следующей части программы датчик касания используется для того, чтобы робот мог определить, что доехал до угла, а затем повернуть налево и следовать вдоль новой стены. Это похоже на программу *BumperBot*, благодаря которой робот TriBot может откатываться назад и поворачивать при столкновении с препятствием.

В листинге 7.3 приведен псевдокод для данного раздела программы.

Листинг 7.3. Псевдокод для поворота за угол

```
if the Touch Sensor is pressed then
   stop the motors
   back up far enough to turn the robot
   spin a quarter-turn to the left
end if
```

После отката и поворота робот TriBot должен находиться на правильном расстоянии от стены, чтобы не врезаться в нее и не отдалиться от нее слишком сильно. Для этого тебе нужно определить правильные значения продолжительности для двух блоков **Рулевое управление** (Move Steering). Можешь начать со значений из программы *BumperBot* и при необходимости скорректировать их после некоторого тестирования.

Написание кода

Для создания этого раздела программы выполни следующие действия:

15. Перетащи блок **Переключатель** (Switch) в блок **Цикл** (Loop) и помести его справа от существующего блока **Переключатель** (Switch). Программа должна выглядеть, как показано на рис. 7.13. Установи параметры по умолчанию: режим **Датчик касания** (Touch Sensor) ⇒ **Сравнение** (Compare) ⇒ **Состояние** (State). Блоки в верхнем случае блока **Переключатель** (Switch) будут выполнены, если робот врежется в стену, что приведет к нажатию кнопки датчика касания.

16. Перетащи блок **Рулевое управление** (Move Steering) в верхний случай нового блока **Переключатель** (Switch). Выбери для него режим **Выключить** (Off).

17. Добавь еще один блок **Рулевое управление** (Move Steering) в верхнюю последовательность. Выбери режим **Включить на количество градусов** (On for Degrees) и задай значение **–300** для параметра **Градусы** (Degrees); в результате робот TriBot откатится от угла.

18. Задай значение **25** для параметра **Мощность** (Power), чтобы он соответствовал другим блокам **Рулевое управление** (Move Steering). Движение робота будет более плавным, если во всех блоках **Рулевое управление** (Move Steering) используется одно и то же значение параметра **Мощность** (Power).

19. Добавь еще один блок **Рулевое управление** (Move Steering) в верхний случай. Задай значение **–100** для параметра **Рулевое управление** (Steering), значение **250** для параметра **Градусы** (Degrees) и значение **25** для параметра **Мощность** (Power). В результате робот повернет так, что инфракрасный датчик будет указывать в направлении новой стены.

ПРИМЕЧАНИЕ **Образовательная версия конструктора предусматривает шины бóльшего размера, чем домашняя, поэтому для ее использования надо будет изменить значения некоторых параметров. Используй значение 185 для параметра Градусы (Degrees) вместо 250.**

20. Задай значение **–100** для параметра **Рулевое управление** (Steering), чтобы робот повернул влево.

21. В этом блоке **Переключатель** (Switch) используется только верхний случай, поскольку программе не нужно ничего делать, если кнопка датчика касания не будет нажата. Щелкни по переключателю **Плоский вид/Вид с вкладками** (Flat/Tabbed View) в блоке **Переключатель** (Switch), чтобы использовать вид с вкладками и скрыть пустой нижний случай.

На рис. 7.14 показан готовый блок **Переключатель** (Switch)

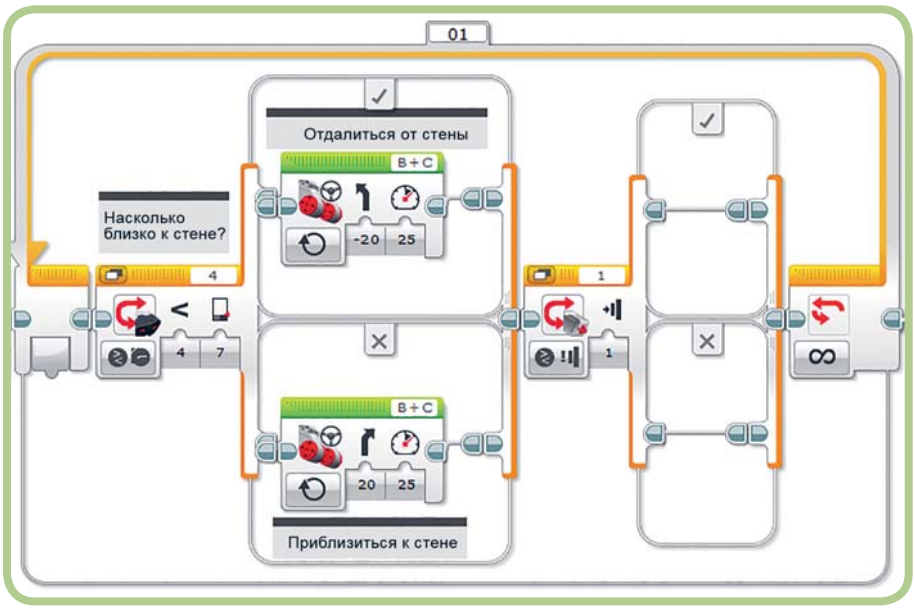


*Рис. 7.13. Добавление еще одного блока **Переключатель** (Switch)*

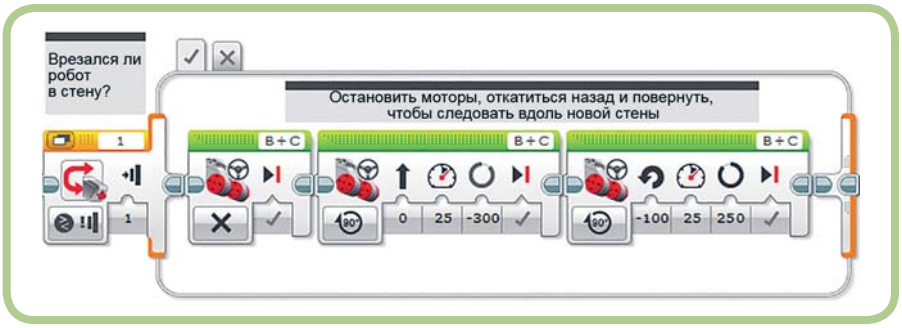


Рис. 7.14. Поворот за угол

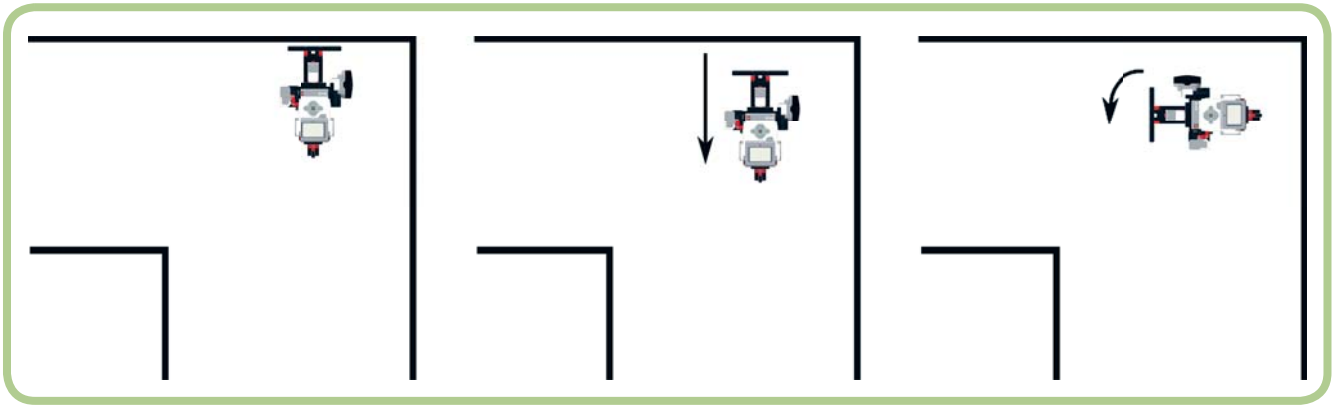


Рис. 7.15. Откат от стены и поворот налево

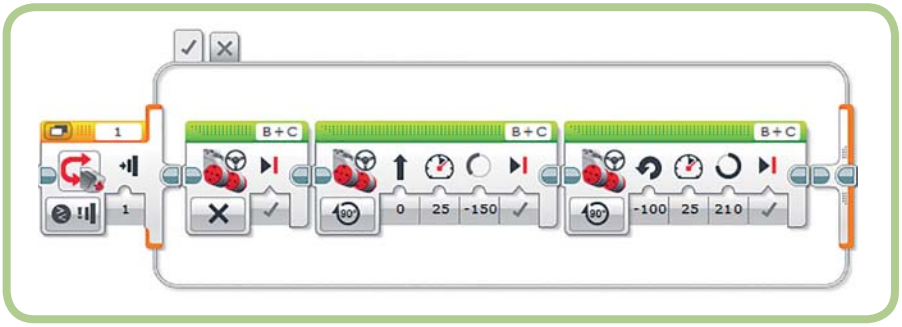


Рис. 7.16. Значения параметров для выполнения отката

Тестирование

Протестируем новый код. Для этого помести робота TriBot близко к углу и посмотри, как он будет действовать, когда врежется в стену. На рис. 7.15 примерно показано, как должен двигаться робот при выполнении отката и поворота.

При первом тестировании были выявлены две проблемы: робот откатывается слишком далеко назад; робот поворачивается на бóльшее количество градусов, чем требуется. Попробовав несколько разных значений, было выбрано 150° для отката и на 210° для поворота. На рис. 7.16 показаны два блока **Рулевое управление** (Move Steering) с новыми значениями.

ПРИМЕЧАНИЕ **В случае использования шин из образовательной версии конструктора задай 110° для отката и 160° для поворота.**

Прежде чем переходить к следующему разделу программы, протестируй ранее приведенный код для следования вдоль стены еще раз, чтобы убедиться в том, что он по-прежнему работает как надо.

ПРИМЕЧАНИЕ **При добавлении новых фрагментов кода рекомендуется тестировать те части программы, которые работали до внесения изменений. Это упрощает и ускоряет поиск ошибок, которые могли возникнуть из-за добавления нового кода.**

Въезд в проход

Когда робот TriBot оказывается рядом с проходом в стене справа, он должен повернуться, въехать в проход и продолжить следовать вдоль правой стены.

Когда робот будет проезжать мимо прохода, показание инфракрасного датчика резко увеличится. Инфракрасный

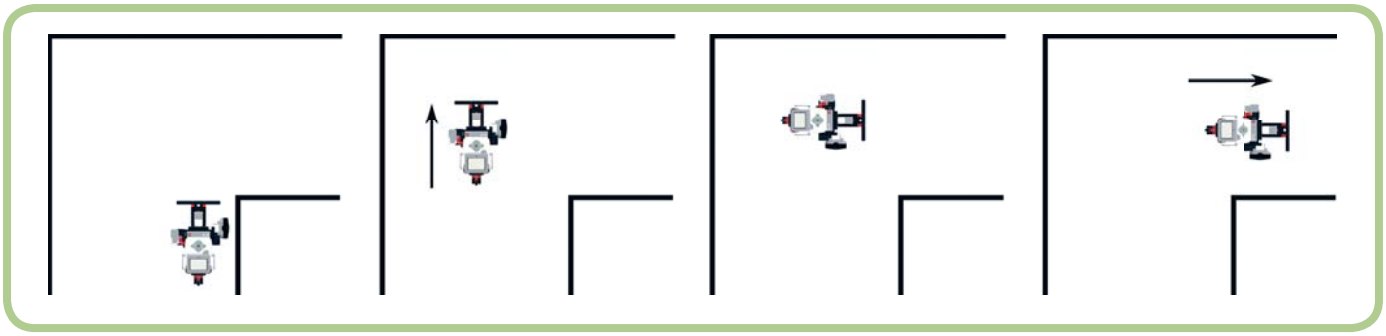


Рис. 7.17. Поворот и въезд в проход

датчик достигнет прохода раньше, чем задняя часть робота, поэтому перед выполнением поворота роботу нужно будет проехать еще немного вперед. После поворота на 90° по направлению к проходу роботу придется проехать вперед еще немного, прежде чем инфракрасный датчик окажется на нужном расстоянии от стены. На рис. 7.17 показано, как должен двигаться робот, а в листинге 7.4 приведен псевдокод для данного раздела программы.

Листинг 7.4. Псевдокод для въезда в проход

```
if the Ultrasonic Sensor detects an opening then
   stop the motors
   move forward a little
   spin a quarter-turn to the right
   move forward into the opening
end if
```

Перед добавлением в программу каких-либо блоков необходимо определить значения параметров для блока **Переключатель** (Switch) и блоков **Рулевое управление** (Move Steering). Для блока **Переключатель** (Switch) начни с порогового значения, равного 40 см (25 см для ультразвукового датчика). Это значение достаточно велико и позволяет убедиться в том, что робот обнаружил настоящий проход, а не простое углубление в стене. В блоках **Рулевое управление** (Move Steering) начни со значений, использованных ранее для выполнения отката из угла, а затем откорректируй эти значения после проведения некоторого тестирования.

Написание кода

Выполни следующие действия, чтобы заставить робота TriBot въехать в проход:

22. Добавь блок **Переключатель** (Switch) в конец кода внутри блока **Цикл** (Loop).

23. Выбери режим **Инфракрасный датчик** (Infrared Sensor) ⇒ **Сравнение** (Compare) ⇒ **Приближение** (Proximity). Для параметра **Пороговое значение** (Threshold value) задай значение **15**, а для параметра **Тип сравнения** (Compare Type) выбери вариант **Больше чем** (Greater Than).

24. Щелкни по переключателю **Плоский вид/Вид с вкладками** (Flat/Tabbed View). Тебе нужно добавить блоки только в случай «Истина».

25. Добавь блок **Рулевое управление** (Move Steering) в случай «Истина» блока **Переключатель** (Switch). Выбери режим **Выключить** (Off).

26. Добавь еще один блок **Рулевое управление** (Move Steering). Выбери режим **Включить на количество градусов** (On for Degrees), задай значение **25** для параметра **Мощность** (Power) и **150** для параметра **Градусы** (Degrees). В результате робот должен проехать вперед так, чтобы целиком оказаться рядом с проходом.

27. Добавь третий блок **Рулевое управление** (Move Steering). Выбери режим **Включить на количество градусов** (On for Degrees), задай значение **100** для параметра **Рулевое управление** (Steering), **25** для параметра **Мощность** (Power) и **210** для параметра **Градусы** (Degrees). В результате робот должен развернуться вправо по направлению к проходу.

28. Добавь четвертый блок **Рулевое управление** (Move Steering). Выбери режим **Включить на количество градусов** (On for Degrees), задай значение **25** для параметра **Мощность** (Power) и **150** для параметра **Градусы** (Degrees). В результате робот должен въехать в проход.

ПРИМЕЧАНИЕ **При использовании шин из образовательной версии конструктора задай 110° для перемещения вперед и 160° для поворота.**

Данная часть программы показана на рис. 7.18.

Тестирование

Протестируй новый код. Для этого расположи робота TriBot параллельно стене рядом с проходом и запусти программу. Внимательно понаблюдай за тем, как робот въезжает в проход и начинает двигаться вдоль новой стены. Вот некоторые нюансы, на которые следует обратить внимание:

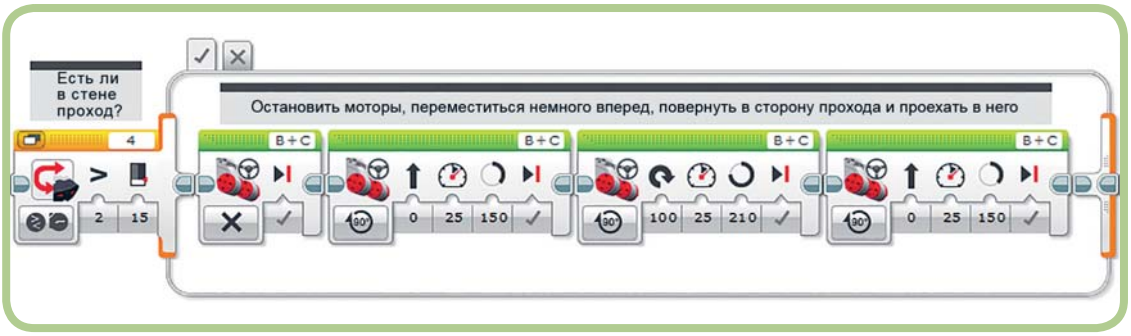


Рис. 7.18. *Поворот в проход*

ИСПОЛЬЗОВАНИЕ БЛОКОВ «ЗВУК» ДЛЯ ОТЛАДКИ ПРОГРАММЫ

Для выполнения поворота в направлении прохода надо использовать четыре блока **Рулевое управление** (Move Steering). При тестировании и корректировке этого кода пригодится информация, где заканчивается один блок и начинается следующий. Это позволит лучше понять, параметры какого блока требуется изменить.

Стоит добавить блоки **Звук** (Sound), воспроизводящие различные ноты перед каждым блоком, чтобы можно было определить, где начинается и заканчивается каждый из блоков **Рулевое управление** (Move Steering). Убедись в том, что в блоке **Звук** (Sound) для параметра **Тип воспроизведения** (Play Type) выбран вариант **Воспроизвести один раз** (Play Once) (вместо варианта **Ожидать завершения** (Wait for Completion)), чтобы робот TriBot не останавливался во время воспроизведения звука. На рис. 7.19 показан блок **Звук** (Sound), добавленный между первым и вторым блоками **Рулевое управление** (Move Steering).

Всякий раз при добавлении кода с целью отладки, как в случае с блоком **Звук** (Sound), следует убедиться в том, что добавленный фрагмент кода оказывает минимальное влияние на время выполнения программы; в противном случае удаление этого кода может привести к изменению поведения программы.

По окончании тестирования ты можешь удалить блоки **Звук** (Sound), однако после их удаления не забудь снова протестировать программу, чтобы убедиться в том, что она по-прежнему работает.

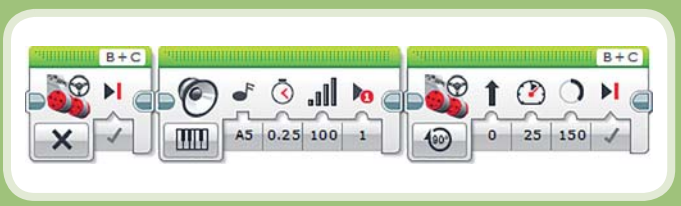


Рис. 7.19. Воспроизведение звука перед движением вперед

— Если пороговое значение для инфракрасного или ультразвукового датчика слишком велико, робот может не заметить проход.
— Если пороговое значение для инфракрасного или ультразвукового датчика слишком мало, робот повернет к стене там, где прохода нет.
— Если инфракрасный или ультразвуковой датчик не может обнаружить стену из-за свойств ее поверхности (слишком мягкая, слишком темная и т. д.), робот повернет к стене, посчитав, что обнаружил проход.
— Если робот не переместится вперед на достаточное расстояние, он повернет слишком рано, задев край стены.
— Если робот переместится слишком далеко вперед, прежде чем повернуть, он проедет мимо прохода или окажется слишком далеко от стены после выполнения поворота.
— Если поворот окажется слишком коротким, робот окажется слишком далеко от стены.
— Если поворот окажется слишком длинным, робот врежется в стену.
— Если последний ход окажется слишком коротким, робот не въедет в проход, и инфракрасный датчик не обнаружит новую стену.
— Если последний ход окажется слишком длинным, робот может проехать мимо расположенного поблизости угла или врезаться в другую стену.

В результате проведения тестирования я обнаружил неточность параметра продолжительности для всех трех ходов. После нескольких экспериментов было выбрано значение 300° для движения вперед, 190° для поворота и 350° для въезда в проход. Было обнаружено, что иногда робот TriBot ведет себя так, будто проход есть, даже когда его нет. Исправить эту проблему удалось благодаря заданию значения 20 для параметра **Пороговое значение** (Threshold value) в блоке **Переключатель** (Switch).

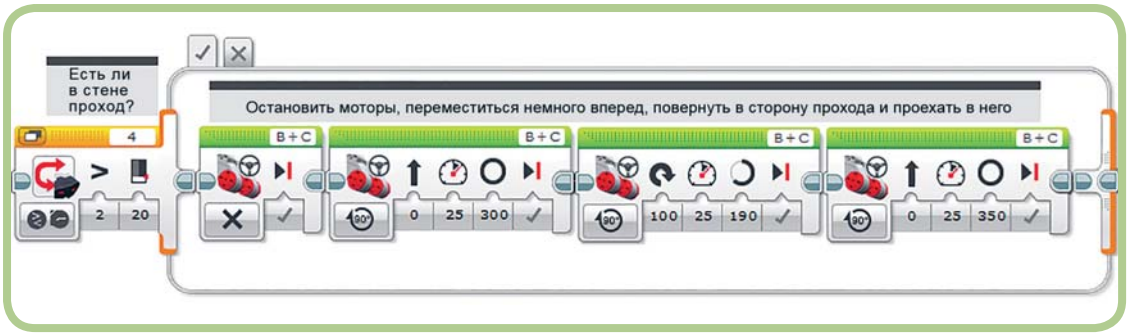


Рис. 7.20. Полученные в результате тестирования значения параметров для сворачивания в проход

ПРИМЕЧАНИЕ Используя образовательную версиию конструктора, попробуй установить значения продолжительности 225°, 140° и 260° и пороговое значение для ультразвукового датчика 33 см.

На рис. 7.20 показан данный раздел программы с итоговыми значениями.

Итоговый тест

Когда робот TriBot научится правильно въезжать в проход, настанет время для тестирования всей программы. Согласно списку требований к программе (см. подраздел «Требования к программе» ранее в этой главе), робот TriBot должен следовать вдоль прямой стены и реагировать на углы и проходы. Если робот справляется с этими задачами, он сможет найти выход из простого лабиринта. Попробуй использовать различные тестовые лабиринты, чтобы посмотреть, успешно ли программа справляется с несколькими поворотами и углами подряд.

Убедившись, что программа соответствует определенным ранее требованиям, посмотри, как она работает в других ситуациях. Например, измени расстояние между стенами (сделай коридоры более узкими или более широкими), чтобы проверить, как это влияет на поведение робота. Кроме того, ты можешь добавить изогнутые или наклоненные стены, чтобы посмотреть, как робот на них реагирует. Несмотря на то что программой не предусмотрено таких ситуаций, она может работать вполне нормально. Если это не так, подумай, как изменить программу, чтобы сделать ее более универсальной.

Дальнейшее исследование

Удостоверившись в работоспособности программы, попробуй выполнить следующие действия:

1. Если ты можешь заставить робота все время находиться на правильном расстоянии от стены, тебе необязательно использовать отдельный фрагмент кода для распознавания прохода. Робот будет автоматически поворачивать направо, используя код в первом разделе, который задает значение параметра **Рулевое управление** (Steering), исходя из показания инфракрасного или ультразвукового датчика. Задача состоит в нахождении порогового значения, которое будет удерживать робота на правильном расстоянии, чтобы он мог совершить поворот, не отклоняясь в сторону и не врезаясь в стену. Попробуй разные значения, чтобы посмотреть, сможешь ли ты решить эту задачу для своей тестовой зоны.

2. Текущая версия программы заставляет робота поворачиваться на 90°, поскольку было решено, что лабиринт будет содержать только прямые углы. Однако ни один из этих поворотов не должен быть идеальным, так как робот вскоре скорректирует свое положение на основе показаний инфракрасного или ультразвукового датчиков. Попробуй использовать лабиринт с коридорами, которые соединяются под разными углами, и поэкспериментируй с различными углами поворота, чтобы научить робота справляться с различными ситуациями. Что лучше, позволить роботу повернуть на слишком большое или на недостаточное количество градусов?

3. В программе *LineFollower* в гл. 6 были использованы вложенные блоки **Переключатель** (Switch) для уменьшения движения из стороны в сторону. Тот же метод может быть применен к коду, который заставляет робота TriBot следовать вдоль стены. Внеси необходимые изменения, посмотри, насколько плавным может быть движение робота вдоль стены, а затем оцени эффект, если таковой имеется, на качество распознавания проходов справа.

4. Вместо использования датчика касания для обнаружения стены попробуй использовать датчик цвета и ярко окрашенную область на пересечении стен. Пробуй разные цвета, чтобы дать программе особые инструкции, например воспроизвести определенный звук при распознавании красного цвета или изменить скорость или направление движения при распознавании синего цвета.

Заключение

Из этой главы ты узнал о процессе разработки простой программы EV3 для нахождения выхода из лабиринта. На каждом этапе мы добавляли логическую часть программы, проводили тестирование и затем вносили необходимые изменения для корректировки продолжительности движения и исправления ошибок.

В гл. 8 ты познакомишься с шинами данных — одной из самых мощных функций программирования EV3.

8

Шины данных

Из этой главы ты узнаешь, как применять шины данных для передачи информации из одного блока в другой. С их помощью можно изменять параметры блоков во время выполнения программы (например, используя новое показание датчика). Шины данных представляют собой одну из самых мощных функций программирования EV3, и изучение способов их использования позволит тебе создавать гораздо более сложные программы.

Начнем с простого примера, чтобы понять, что такое шина данных и как она работает. Далее разработаем более сложную программу, с помощью которой ты сможешь превратить робота TriBot в генератор звука. Параллельно также рассмотрим все базовые концепции, необходимые для успешного применения шин данных в твоих собственных программах. Кроме того, ты узнаешь о нескольких новых блоках, которые особенно полезны при работе с шинами данных.

Что такое шина данных?

Для выполнения действия для большей части блоков требуется информация — *данные*. Например, блок **Рулевое управление** (Move Steering) должен знать, какие моторы использовать, насколько быстро и как долго их нужно вращать. Это называется *входными данными* блока. До этого момента мы вручную задавали входные данные для каждого блока.

Некоторые блоки генерируют данные, которые могут быть использованы другими блоками. Такие данные называются *выходными*. Например, блок **Вращение мотора** (Motor Rotation) считывает показания датчика вращения мотора и может передать его другим блокам.

Шина данных получает выходные данные из одного блока и использует их в качестве входных данных в другом блоке. Это обеспечивает большую гибкость по сравнению с вводом значений параметров, поскольку допускает изменение параметров блока во время выполнения программы. Ты можешь использовать показания датчика для управления другим блоком.

Программа GentleStop

Программа *GentleStop* заставляет робота TriBot двигаться вперед, а затем постепенно замедляться и останавливаться при приближении к стене; в ней используется шина данных.

Ключевым моментом этой программы является то, что скорость робота зависит от расстояния до стены (рис. 8.1). По мере приближения к стене скорость робота постепенно уменьшается, пока он полностью не остановится. Блок **Инфракрасный датчик** (Infrared Sensor) измеряет расстояние от робота до стены, а блок **Рулевое управление** (Move Steering) использует это значение для настройки параметра **Мощность** (Power) для управления скоростью робота. На рис. 8.2 показана связь между этими двумя блоками в программе.

ПРИМЕЧАНИЕ **При использовании образовательной версии конструктора замени инфракрасный датчик и блок Инфракрасный датчик (Infrared Sensor) на ультразвуковой датчик и блок Ультразвуковой датчик (Ultrasonic Sensor). Программа *GentleStop* работает одинаково хорошо с обоими датчиками.**

По мере выполнения программы параметр **Мощность** (Power) блока **Рулевое управление** (Move Steering) корректируется в соответствии с изменением расстояния между роботом и стеной. Например, когда показание датчика равно 80, робот находится относительно далеко от стены, поэтому значение параметра **Мощность** (Power) также равно 80, и робот движется очень быстро. Но показание датчика, равное 20, говорит о том, что робот приближается к стене, поэтому значение параметра **Мощность** (Power) уменьшается до 20, а скорость робота значительно снижается. В конце концов робот врезается в стену, при этом нажимается кнопка датчика касания, и программа завершается.

ПРИМЕЧАНИЕ Для измерения расстояния до стены инфракрасный датчик должен быть направлен вперед. Если этот датчик не направлен вперед, установи его так, как показано на рис. 8.3.

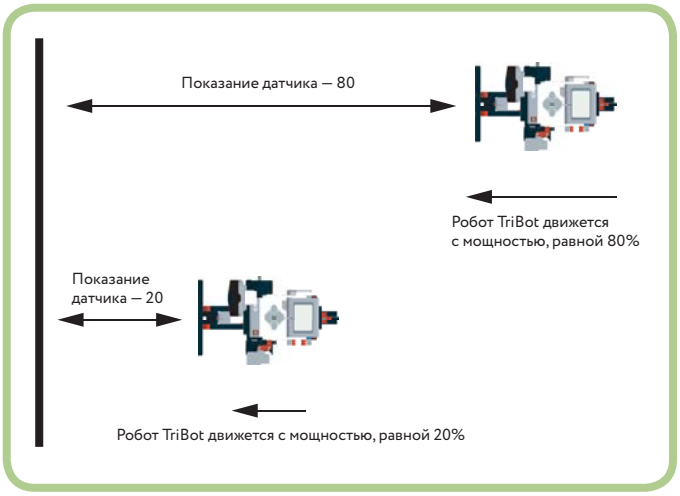


Рис. 8.1. Робот TriBot замедляется по мере приближения к стене

Рис. 8.2. Взаимосвязь между блоком **Инфракрасный датчик** *(Infrared Sensor) и блоком* **Рулевое управление** *(Move Steering)*

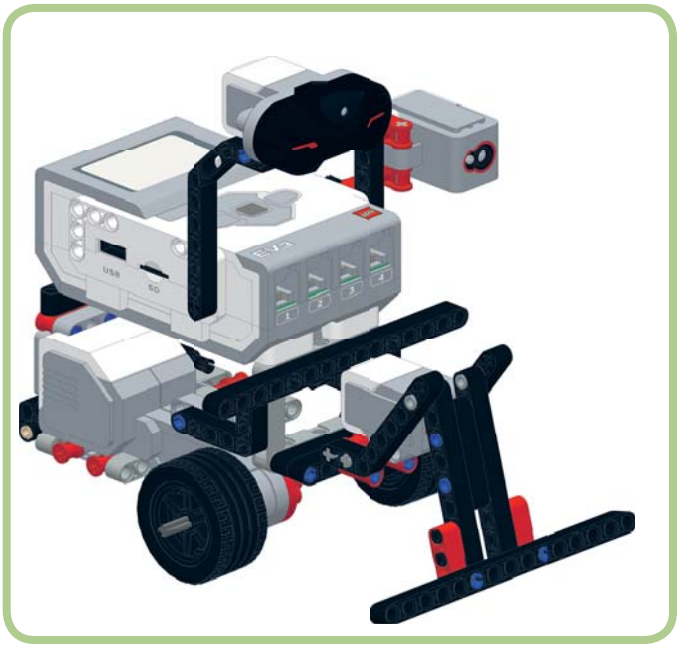


Рис. 8.3. Инфракрасный датчик, направленный вперед

Написание программы

Для создания этой программы ты поместишь блок **Инфракрасный датчик** (Infrared Sensor) и блок **Рулевое управление** (Move Steering) в блок **Цикл** (Loop). Выход из цикла произойдет, когда будет нажата кнопка датчика касания, таким образом, программа завершится, когда робот осторожно ударится о стену. Затем ты применишь шину данных для того, чтобы передать показание инфракрасного датчика на ввод **Мощность** (Power) блока **Рулевое управление** (Move Steering).

Блоки датчиков (например, блок **Инфракрасный датчик** (Infrared Sensor), блок **Датчик касания** (Touch Sensor), блок **Датчик цвета** (Color Sensor) и др.) предусматривают те же режимы и элементы управления, что и блоки **Ожидание** (Wait), **Переключатель** (Switch) и **Цикл** (Loop). Каждый блок датчика имеет один или несколько выводов, которые благодаря шинам данных можно применять для управления чем-то еще помимо блоков **Ожидание** (Wait), **Переключатель** (Switch) и **Цикл** (Loop). Например, в блоке **Инфракрасный датчик** (Infrared Sensor), показанном на рис. 8.4, используется режим **Измерение** (Measure) ⇒ **Маяк** (Beacon), в котором предусмотрены выводы, сообщающие о том, был ли обнаружен маяк, расстояние до маяка, а также направление, в котором он находится относительно датчика. Блок **Ультразвуковой датчик** (Ultrasonic Sensor), показанный на рис. 8.5, имеет только один вывод, сообщающий расстояние до ближайшего обнаруженного объекта.

Рис. 8.4. Блок ***Инфракрасный датчик*** *(Infrared Sensor)*

Рис. 8.5. Блок ***Ультразвуковой датчик*** *(Ultrasonic Sensor)*

Выполни следующие действия для создания программы:

1. Создай новый проект под названием *Chapter8*.
2. Создай новую программу под названием *GentleStop*.
3. Добавь блоки, изображенные на рис. 8.6. В блоке **Рулевое управление** (Move Steering) используется режим **Включить** (On), а в блоке **Инфракрасный датчик** (Infrared Sensor) — режим **Измерение** (Measure) ⇒ **Приближение** (Proximity).

ПРИМЕЧАНИЕ При использовании блока **Ультразвуковой датчик** (Ultrasonic Sensor) выбери режим **Измерение** (Measure) ⇒ **Расстояние в сантиметрах** (Distance Centimeters).

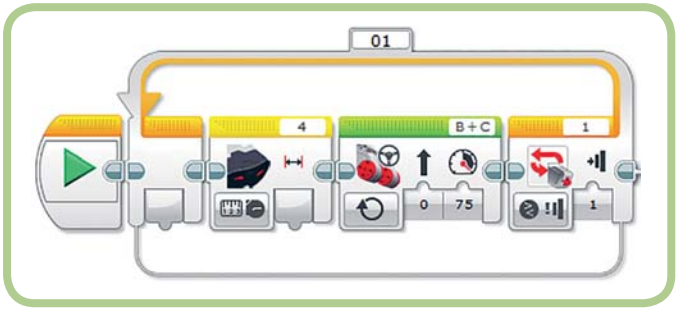


Рис. 8.6. Программа GentleStop перед подключением шины данных

Теперь, когда все три блока на месте, пришло время соединить блоки **Инфракрасный датчик** (Infrared Sensor) и **Рулевое управление** (Move Steering) с помощью шины данных.

Входными данными для изображенного на рис. 8.6 блока **Рулевое управление** (Move Steering) являются значения параметров **Рулевое управление** (Steernig) и **Мощность** (Power). В блоке **Инфракрасный датчик** (Infrared Sensor) в той конфигурации, в которой он представлен на рис. 8.6, не предусмотрены никакие входные данные; значение параметра **Приближение** (Proximity) — это выходные данные блока. Подтверждением того, что это именно выходные данные, служит небольшой полукруг на нижней границе серого квадрата (называемого *выводом*). Обрати внимание на то, что входные данные блока, например **Рулевое управление** (Move Steering), отображены в квадрате с полукругом вверху (называемом *вводом*).

При выполнении блока **Инфракрасный датчик** (Infrared Sensor) датчик измерит расстояние до стены и предоставит полученное значение в качестве выходного параметра **Приближение** (Proximity). Затем используем шину данных для того, чтобы передать показание датчика из вывода **Приближение** (Proximity) на ввод **Мощность** (Power) блока **Рулевое управление** (Move Steering).

4. Наведи указатель мыши на вывод блока **Инфракрасный датчик** (Infrared Sensor). При этом указатель должен принять вид катушки с проволокой (рис. 8.7).

Рис. 8.7. Вид указателя мыши при создании шины данных

5. Нажми и удерживай кнопку мыши на выводе. При перетаскивании указателя мыши между ним и блоком **Инфракрасный датчик** (Infrared Sensor) будет нарисован желтый провод, как показано на рис. 8.8.

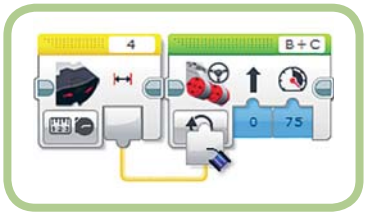


Рис. 8.8. Добавление шины данных

6. Перетащи вывод (и провод) на ввод **Мощность** (Power) блока **Рулевое управление** (Move Steering), чтобы завершить создание шины данных. Программа должна выглядеть так, как показано на рис. 8.9.

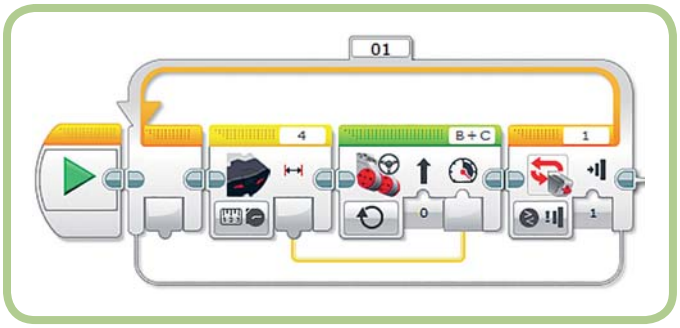


Рис. 8.9. Программа GentleStop

ПРИМЕЧАНИЕ **Если ты случайно подключишь шину не к тому вводу (что часто случается), выбери команду меню Редактировать** (Edit) ⇒ **Отменить** (Undo) **или нажми комбинацию клавиш Ctrl+Z, чтобы удалить шину данных и начать заново.**

Теперь пришло время загрузить и протестировать программу. Помести робота TriBot на середину комнаты так, чтобы он был направлен к стене. Сначала он должен двигаться быстро, а затем замедляться по мере приближения к стене. После легкого удара о стену он должен остановиться.

ПРИМЕЧАНИЕ **Не помещай робота слишком близко к стене. Моторы могут вообще не запуститься, если показание датчика будет меньше минимального значения параметра Мощность** (Power), **необходимого для перемещения робота (я указал 5).**

Советы по использованию шин данных

Обычно соединить два блока с помощью шины данных не составляет труда, как в случае с программой *GentleStop*. Однако иногда тебе может понадобиться удалить шину данных или получить больший контроль над процессом ее добавления. Вот несколько советов, которые облегчают работу с шинами данных:

— Подожди, пока указатель мыши примет вид катушки с проволокой, прежде чем создавать шину данных.
— Для удаления шины данных в процессе ее создания нажми клавишу **Esc**.
— Можно удалить шину данных сразу после ее создания, выбрав команду меню **Редактировать** (Edit) ⇒ **Отменить** (Undo).

– Для удаления шины данных перетащи ее с ввода блока. В программе *GentleStop* это будет параметр **Мощность** (Power) блока **Рулевое управление** (Move Steering).
– После добавления шины данных ты сможешь щелкнуть по ней и перетащить ее вверх или вниз. Это позволяет организовать несколько шин данных, используемых в одной и той же части программы.
– Выполни двойной щелчок по шине данных, чтобы программное обеспечение EV3 автоматически выбрало для нее место.
– Удаление блока приведет к удалению всех подключенных к нему шин данных.
– Во время выполнения программы ты можешь навести указатель мыши на шину данных, чтобы отобразить передаваемое ей значение, как показано на рис. 8.10. Если значения нет (например, если еще не запущен блок, который должен предоставить соответствующее значение), ты увидишь прочерк «----». Эта функция невероятно полезна при отладке программы, но она работает только тогда, когда модуль EV3 подключен к программному обеспечению, а программа запускается средой EV3 (а не с помощью кнопок модуля).

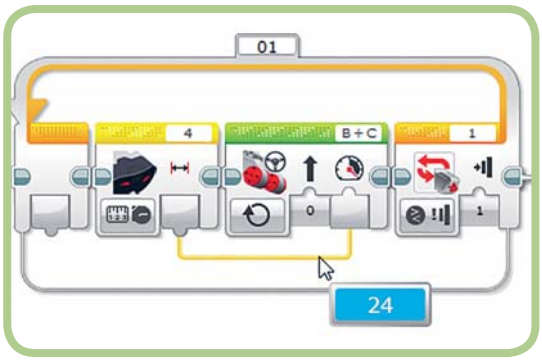


Рис. 8.10. Отображение значения в шине данных

– Каждый ввод может быть подключен только к одному выводу, иначе принимающий блок не знал бы, какое значение нужно использовать. Вывод может быть подключен к нескольким вводам, поскольку одно и то же значение можно без проблем передать нескольким блокам.

ПРАКТИКУМ 8.1

Прикрепи датчик цвета к передней части робота TriBot так, как мы делали это для программы *LineFollower*. Создай копию программы *GentleStop* и используй датчик цвета в режиме **Яркость внешнего освещения** (Ambient Light Intensity) вместо блока **Инфракрасный датчик** (Infrared Sensor). С помощью фонарика ты сможешь управлять движением робота. Робот не будет следовать за светом от фонарика, однако он будет двигаться быстрее или медленнее в зависимости от расстояния между лучом света и датчиком.

Программа SoundMachine

Следующая программа *SoundMachine* позволяет превратить робота TriBot в простой генератор звука. Колесо, прикрепленное к мотору B, регулирует громкость, а колесо, прикрепленное к мотору C, определяет тон (или высоту звука). Поверни колесо мотора B, чтобы уменьшить или увеличить громкость звука, поверни колесо мотора C, чтобы увеличить или уменьшить высоту звука, как показано на рис. 8.11.

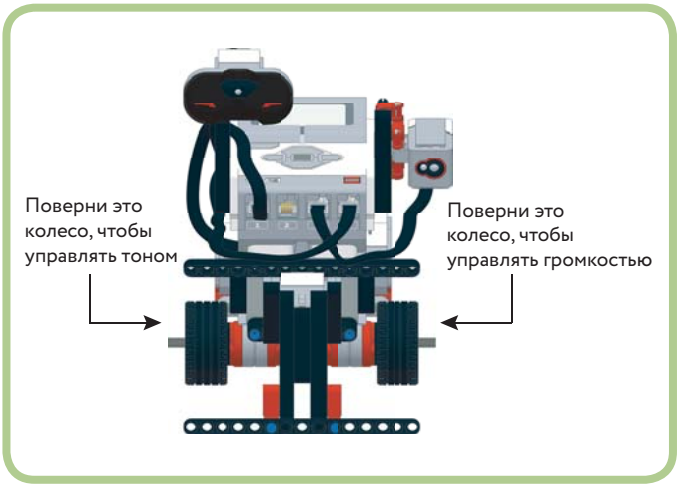


*Рис. 8.11. Использование колес для управления блоком **Звук***

В этой программе используется блок **Звук** (Sound) для воспроизведения звука и два блока **Вращение мотора** (Motor Rotation) для определения того, насколько повернулось каждое из колес. Ты будешь использовать шины данных для подключения выводов блоков **Вращение мотора** (Motor Rotation) к вводам блока **Звук** (Sound). Я представлю эту программу в трех частях. Сначала ты будешь управлять только громкостью, затем добавишь фрагмент кода для управления тоном и, наконец, отобразишь значения громкости и тона на экране модуля EV3.

Управление громкостью звука

Первая часть программы напоминает программу *GentleStop* за исключением того, что она использует датчик вращения мотора и блоки **Звук** (Sound) вместо инфракрасного датчика и блоков **Рулевое управление** (Move Steering). Ты настроишь блок **Звук** (Sound) так, чтобы он воспроизводил тон, и используешь показание датчика вращения мотора для управления его громкостью. Вся программа будет заключена в цикл.

Слишком быстрое повторение блока **Звук** (Sound) может привести к искажению звука. Небольшая пауза продолжительностью 0,02 секунды в конце тела цикла позволит решить эту проблему. В листинге 8.1 приведен псевдокод для этой части программы *SoundMachine*.

Листинг 8.1. Псевдокод для управления громкостью звука

```
begin loop
   read the Rotation Sensor for motor B
   use a Sound block to play a tone; use the
      Rotation Sensor value for Volume
   use a Wait block to pause for 0.02 seconds
loop forever
```

На рис. 8.12 показана программа перед подключением шины данных. Блок **Вращение мотора** (Motor Rotation) настроен на использование мотора B, а для блока **Звук** (Sound) выбран режим **Воспроизвести тон** (Play Tone). Для параметра **Тип воспроизведения** (Play Type) блока **Звук** (Sound) выбран вариант **Воспроизвести один раз** (Play Once), чтобы выполнение программы не приостанавливалось на время воспроизведения звука. По мере повторения цикла звук продолжает воспроизводиться, показание датчика вращения мотора постоянно проверяется, и в соответствии с его изменением регулируется громкость звука.

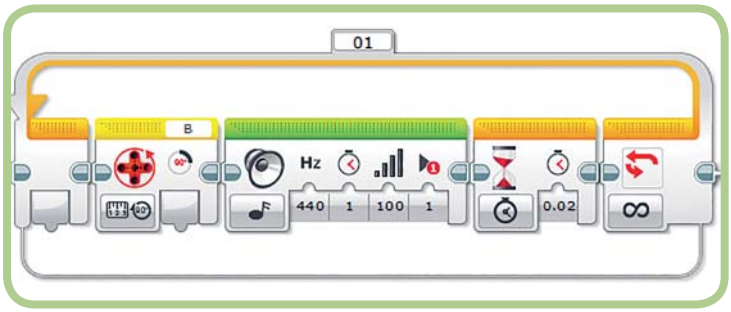


Рис. 8.12. Программа SoundMachine перед подключением шины данных

Теперь соедини вывод **Градусы** (Degrees) блока **Вращение мотора** (Motor Rotation) с вводом **Громкость** (Volume) блока **Звук** (Sound), чтобы завершить создание этой части программы (рис. 8.13).

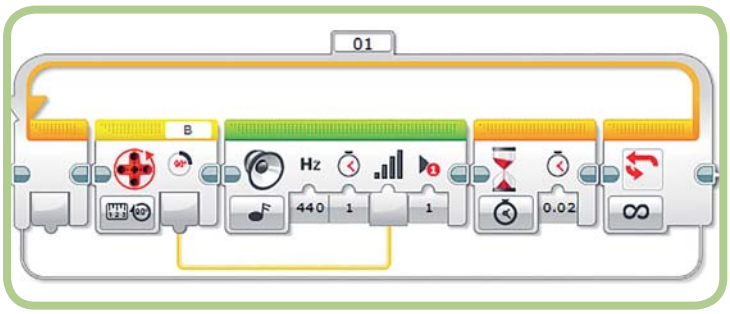


Рис. 8.13. Вращение колеса позволяет регулировать громкость звука

При запуске программы ты ничего не услышишь, поскольку показание датчика вращения мотора будет равно 0, но стоит повернуть мотор B на несколько градусов вперед, и звук станет громче.

Параметр **Громкость** (Volume) блока **Звук** (Sound) может принимать значения от 0 (без звука) до 100 (самый громкий звук). Значение, используемое в блоке **Вращение мотора** (Motor Rotation), измеряется в градусах, причем 100 градусов — это поворот колеса чуть больше чем на четверть. Т.е. ты можешь изменить уровень громкости от самого тихого значения до самого громкого, повернув колесо всего на четверть оборота.

Использование блока «Математика»

Следующая часть программы *SoundMachine* потребует некоторых математических вычислений, которые в среде EV3 выполняются с помощью блока **Математика** (Math), показанного на рис. 8.14. Блок **Математика** (Math) находится на красной вкладке палитры программирования, где содержатся блоки операций с данными.

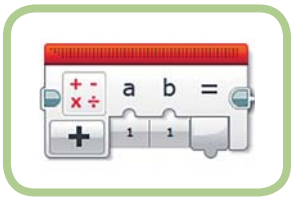


*Рис. 8.14. Блок **Математика** (Math)*

Блок **Математика** (Math) принимает в качестве входных данных одно или два числа и позволяет выбрать операцию, которую требуется над ними совершить. Ты можешь самостоятельно ввести эти числа или предоставить их с помощью шин данных. Результат операции будет подан на вывод блока.

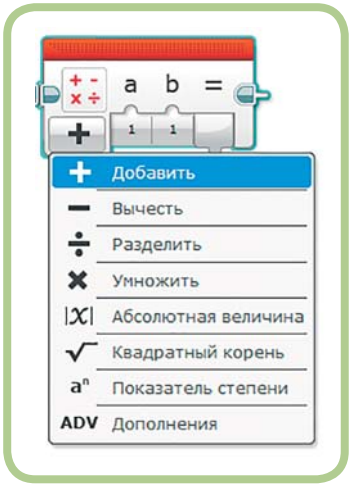


*Рис. 8.15. Выбор операции для блока **Математика** (Math)*

Режим блока **Математика** (Math) определяет, какая операция выполняется над заданными числами (рис. 8.15). Для режимов **Абсолютная величина** (Absolute Value) и **Квадратный корень** (Square Root) требуется только одно число, поэтому при выборе любого из них второй параметр (обозначенный буквой *b*) исчезает. Режим **Дополнения** (Advanced)

позволяет решать более сложные уравнения и рассматривается в гл. 13.

Добавление средства для настройки тона звука

Управление тоном звука в программе *SoundMachine* осуществляется с помощью мотора C. Ты будешь использовать еще один блок **Вращение мотора** (Motor Rotation), вывод которого соединишь с вводом **Частота** (Frequency) блока **Звук** (Sound). Значение параметра **Частота** (Frequency) измеряется в *герцах* (Гц) и варьируется в диапазоне от 300 Гц (сам низкий звук) до 10 000 Гц (самый высокий звук).

ПРИМЕЧАНИЕ **Ты можешь использовать окно контекстной справки или полный файл справки, чтобы подробнее узнать о значениях, принимаемых блоком в качестве входных данных. В окне контекстной справки отобразится диапазон значений и краткое описание. Файл справки содержит более подробную информацию (каждому блоку с таблицей соответствует свой раздел), в которой описаны все входные и выходные данные блока. Чтобы открыть окно контекстной справки, используй комбинацию клавиш Ctrl+H. Для открытия справки нажми клавишу F1.**

Диапазон значений частоты звука достаточно велик, поэтому управление параметром **Частота** (Frequency) блока **Звук** (Sound) является более сложным, чем параметром **Громкость** (Volume). Если соединить блок **Вращение мотора** (Motor Rotation) непосредственно с параметром **Частота** (Frequency), как это было сделано в случае с громкостью, то тебе придется совершить 27 полных оборотов колеса для достижения максимальной высоты звука, что не очень удобно.

Для решения этой проблемы ты можешь использовать блок **Математика** (Math), чтобы умножить значение, полученное от блока **Вращение мотора** (Motor Rotation), на 100, прежде чем передавать его на ввод **Частота** (Frequency). Это преобразует показание датчика вращения мотора, которое находится в диапазоне от 0 до 100 (который используется и для громкости), в значение диапазона от 0 до 10 000. Показания датчика вращения мотора, не превышающие 3, генерируют значения частоты, не превышающие минимального значения 300, вместо которых модуль EV3 будет использовать значение 300. Для упрощения задачи это можно проигнорировать.

В листинге 8.2 приведен псевдокод для программы, новые фрагменты выделены жирным шрифтом:

Листинг 8.2. Псевдокод для добавления средства настройки тона звука

```
begin loop
   read the Rotation Sensor for motor B
   read the Rotation Sensor for motor C
   use a Math block to multiply the motor C
   rotation by 100
   use a Sound block to play a tone; use the
      Rotation Sensor value for Volume;
   use the Math block
   result for the Tone Frequency
   use a Wait block to pause for 0.02 seconds
loop forever
```

На рис. 8.16 показана программа после внесения изменений. Вывод блока **Вращение мотора** (Motor Rotation) сначала передается на ввод блока **Математика** (Math), где значение умножается на 100, а затем передается на ввод **Частота** (Frequency) блока **Звук** (Sound). Чтобы сделать программу более легкой для восприятия, я увеличил размер блока **Цикл** (Loop), перетащив нижнюю границу вниз и опустив шину данных, относящуюся к параметру **Громкость** (Volume). Гораздо проще разобраться в программе, если шины данных не пересекаются друг с другом.

Теперь после запуска программы ты можешь настраивать как высоту звука, так и его громкость, используя колеса робота TriBot.

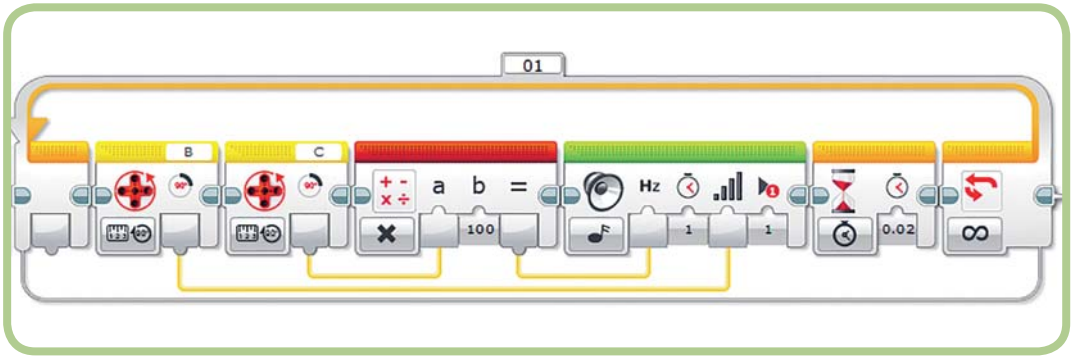


Рис. 8.16. Программа SoundMachine после добавления средства настройки тона звука

Типы данных

Прежде чем завершить создание программы *SoundMachine*, тебе следует узнать еще об одном аспекте шин данных. До сих пор все данные, с которыми мы работали (показания инфракрасного датчика и датчика вращения мотора, а также параметры мощности, громкости и частоты), представляли собой числа. Однако числа — это не единственный тип данных, который можно использовать в программах EV3; кроме них есть логические значения и текст. Подумай, как можно было бы ответить на следующие три вопроса:

— Как тебя зовут?
— Сколько тебе лет?
— Ты старший ребенок в семье?

В ответ на каждый вопрос требуется предоставить информацию разного типа. Ответом на первый вопрос будет слово, на второй — число, а на третий — вариант «да» или «нет». В мире компьютерного программирования разные виды информации называются *типами данных*. Твои ответы на предыдущие три вопроса соответствуют трем типам данных, которые ты будешь использовать при создании программ EV3:

— *Текстовые значения* в программах EV3 в основном используются для отображения информации на экране модуля и представляют собой группы символов, которые могут содержать буквы, цифры и знаки препинания. Например, в первой созданной тобой программе использован блок **Экран** (Display) для отображения текстового значения "Hello".
— *Числовые значения* используются для представления показаний датчиков и задания пороговых значений, а также для настройки многих других параметров, таких как **Мощность** (Power) и **Рулевое управление** (Steering) блока **Рулевое управление** (Move Steering).
— *Логические значения* могут быть либо истинными, либо ложными. Например, ты можешь использовать блок **Инфракрасный датчик** (Infrared Sensor), чтобы проверить, не превысил ли параметр **Расстояние** (Distance) пороговое значение. Результатом этого сравнения будет логическое значение: «Истина», если показание датчика превышает пороговое значение, и «Ложь», если не превышает. В зависимости от способа применения логического значения оно обозначается словами «**Истина»/«Ложь»** ("True"/"False") или «**Да»/«Нет»** ("Yes"/"No"). Этот тип значения часто называют *двоичным*, поскольку он может соответствовать только одному из двух возможных вариантов.

Кроме того, существует еще один тип данных EV3 — массив. Он содержит группу значений. Разделяют числовой массив и логический массив (подробнее см. гл. 15). До тех пор будем работать только с числовыми, текстовыми и логическими значениями.

Тип данных можно определить по форме ввода и вывода, в которых они используются. Числовое значение обозначается полукругом, логическое — треугольником, а текстовое — квадратом. Также на типы данных указывает цвет шины данных: желтый соответствует числовым значениям, зеленый — логическим, а оранжевый — текстовым. В табл. 8.1 показаны выводы блоков и шины данных для каждого типа значений. Вводы блоков выглядят так же, как и выводы, за исключением того, что отличительная форма находится на верхней границе серого квадрата.

Табл. 8.1. Выводы блоков и цвета шин данных в зависимости от типа данных

Тип данных	Вывод блока	Шина данных
Текст		
Число		
Логическое значение		

Каждый ввод блока предназначен для конкретного типа данных, однако модуль EV3 способен производить преобразование некоторых типов данных в автоматическом режиме. Благодаря этому иногда можно соединять выводы одного типа с вводами другого типа. Логическое значение можно передать на ввод, принимающий числовое значение, при этом значение "true" или "false" преобразуется в 1 или 0 соответственно. И логическое, и числовое значение может быть передано на текстовый ввод, при этом числовое будет преобразовано в текстовое. Например, числовое значение 5.3 автоматически превратится в текстовое значение «5.3». Однако в противоположном направлении преобразование не осуществляется, поэтому текстовое значение нельзя передать на ввод, который ожидает получить число.

Отображение значений частоты и громкости звука

Следующие изменения в программе *SoundMachine* позволят отобразить значения частоты и громкости звука на экране модуля EV3. Сначала используй блок **Экран** (Display) для отображения параметра **Частота** (Frequency).

1. Добавь блок **Экран** (Display) после блока **Звук** (Sound). Выбери режим **Текст** (Text) ⇒ **Сетка** (Grid) (рис. 8.17).

Раньше, используя блок **Экран** (Display), мы вводили текст в поле в верхней части блока. Для того чтобы сообщить блоку **Экран** (Display) о передаче текста с помощью шины данных, используй режим **Проводной** (Wired).

*Рис. 8.17. Добавление блока **Экран** (Display)*

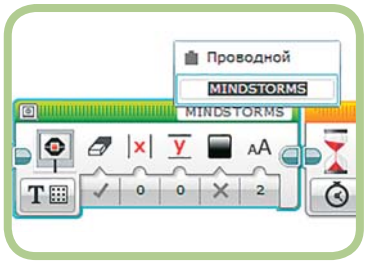


*Рис. 8.18. Выбор режима **Проводной** (Wired) для текстового ввода*

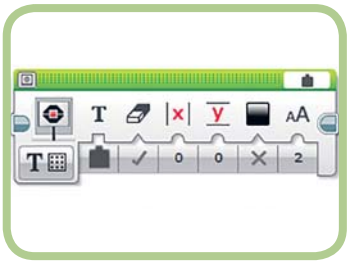


Рис. 8.19. Ввод, в котором предусмотрено использование текстового значения

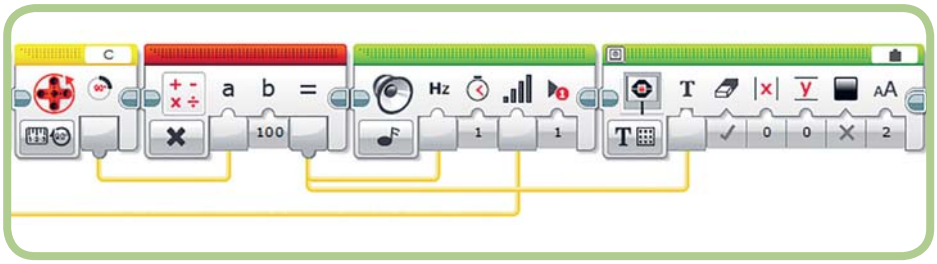


*Рис. 8.20. Подключение блока **Математика** (Math) к блоку **Экран** (Display)*

2. Щелкни по полю и выбери режим **Проводной** (Wired) (рис. 8.18). В блоке отобразится новый ввод для подключения шины данных (рис. 8.19).

Далее необходимо отобразить значение параметра **Частота** (Frequency) блока **Звук** (Sound), которое передается из блока **Математика** (Math). С помощью шины данных подключи блок **Математика** (Math) к блоку **Экран** (Display). Несмотря на то что текстовый ввод обычно ожидает получить текстовое значение, можно использовать число, полученное от блока **Математика** (Math), благодаря автоматическому преобразованию чисел в текст, которое рассмотрено выше.

3. Подключи вывод блока **Математика** (Math) к вводу **Текст** (Text) блока **Экран** (Display).

Эта часть программы должна выглядеть так, как показано на рис. 8.20. Обрати внимание на то, что вывод блока **Математика** (Math) можно подключить как к блоку **Звук** (Sound), так и к блоку **Экран** (Display). После запуска программы при вращении колеса робота на экране модуля EV3 должно отобразиться значение параметра **Частота** (Frequency).

Использование блока «Текст»

Для внесения в программу последних изменений используем блок **Текст** (Text) (рис. 8.21), с помощью которого можно объединить до трех текстовых фрагментов. Это пригодится для добавления меток к значениям, отображаемым на экране модуля EV3. Блок **Текст** (Text) находится на красной вкладке палитры программирования, содержащей блоки операций с данными.

Блок **Текст** (Text) имеет три входных параметра, которые могут принимать текстовые значения. В большинстве случаев ты будешь вводить одно или два значения, а третье передавать с помощью шины данных. Выходное значение блока **Текст** (Text) создается путем слияния (или объединения) трех текстовых фрагментов. Блок **Текст** (Text) не добавляет между этими фрагментами пробел, поэтому, если тебе нужно отделить метку от значения, придется добавить его самостоятельно (как показано в следующем разделе).

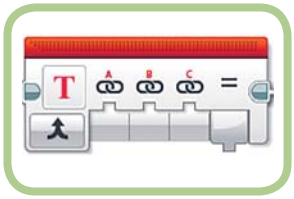


*Рис. 8.21. Блок **Текст** (Text)*

Добавление меток к отображаемым значениям

Предыдущая версия программы *SoundMachine* отображала значение параметра **Частота** (Frequency) на экране модуля EV3. Ты можешь доработать ее, используя блок **Текст** (Text) для добавления дополнительной информации. Так, если добавить метку и единицу измерения к простому числу, вместо отображения значения «2500» программа отобразит “Tone: 2500 Hz”.

На рис. 8.22 показаны внесенные в программу изменения. Вывод блока **Математика** (Math) передается на ввод блока **Текст** (Text) в качестве значения *b*, а значения *a* и *c* определяют текст метки и единицы измерения соответственно. Хотя на рис. 8.22 это не видно, после метки есть пробел. Введенным значением является “Tone: ”, а не “Tone:”. Аналогично в параметре *c* перед Hz тоже есть пробел.

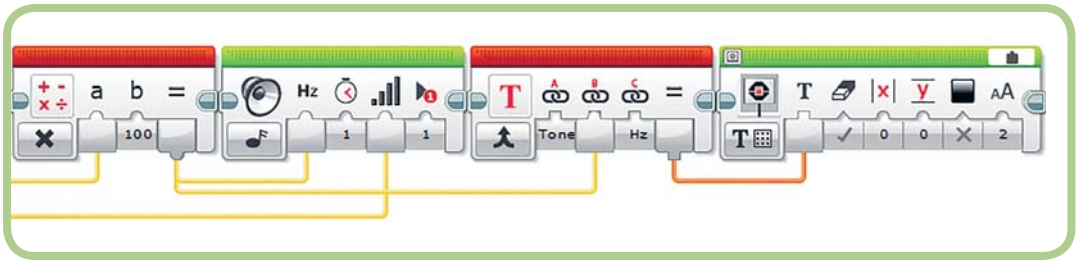


Рис. 8.22. Добавление метки к значению частоты

Внеси показанные здесь изменения, а затем протестируй программу. Ты заметишь, что она отлично работает в начале, когда значение частоты равно 0. Однако по мере увеличения частоты текст станет слишком длинным и уже не уместится на экране, что приведет к исчезновению части значения и единицы измерения. К счастью, это легко исправить с помощью параметра **Шрифт** (Font).

Параметром **Шрифт** (Font) блока **Экран** (Display) предусмотрены три варианта начертания шрифта: обычный (0), жирный (1) и большой (2). По умолчанию выбран вариант «большой», из-за чего текст легче читается, но занимает больше места. При выборе вариантов «обычный» и «жирный» размер букв практически в два раза меньше по сравнению с вариантом по умолчанию, поэтому на экране умещается больше текста. Для того чтобы значение частоты уместилось на экране, настрой блок **Экран** (Display), задав для параметра **Шрифт** (Font) значение 1 (рис. 8.23).

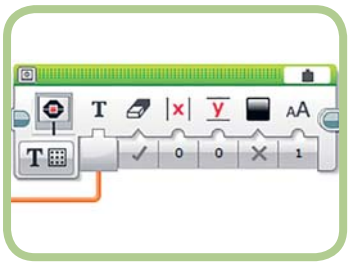


Рис. 8.23. Выбор шрифта меньшего размера

Теперь при запуске программы даже самое большое значение частоты должно уместиться на экране.

ПРАКТИКУМ 8.2

Измени программу *SoundMachine* так, чтобы можно было вращать колеса в любом направлении. Простой способ решения этой задачи: настрой передачу показаний датчика вращения мотора через блок **Математика** (Math) с выбранным режимом **Абсолютная величина** (Absolute Value).

Отображение значения громкости звука

Изменим программу последний раз, добавив в нее функцию отображения значения громкости звука на экране модуля. Новый код напоминает код для отображения частоты и предполагает представление выходного значения блока **Вращение мотора** (Motor Rotation) в виде “Volume: 50%”.

На рис. 8.24 показаны новые блоки **Текст** (Text) и **Экран** (Display). Не забудь добавить пробел после метки для параметра громкости (перед значком процентов пробел не нужен). В новом блоке **Экран** (Display) после выбора режима **Текст** (Text) ⇒ **Сетка** (Grid) измени параметр **Строка** (Row) так, чтобы значение громкости отображалось под значением частоты (а не над ней), и выбери для параметра **Очистить экран** (Clear Screen) вариант **Ложь** (False), чтобы предотвратить стирание значения частоты звука.

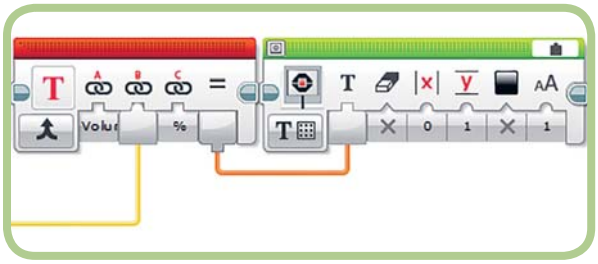


Рис. 8.24. Отображение значения громкости звука

Готовая программа показана на рис. 8.25. При ее выполнении значения частоты и громкости звука должны отображаться вместе с соответствующими подписями и единицами измерения.

ПРАКТИКУМ 8.3

Частота звука не изменится, пока ты немного не повернешь колесо, поскольку минимальное значение частоты равно 300. Исправь эту проблему, добавив в программу блок **Математика** (Math), который гарантирует значение частоты всегда больше 300.

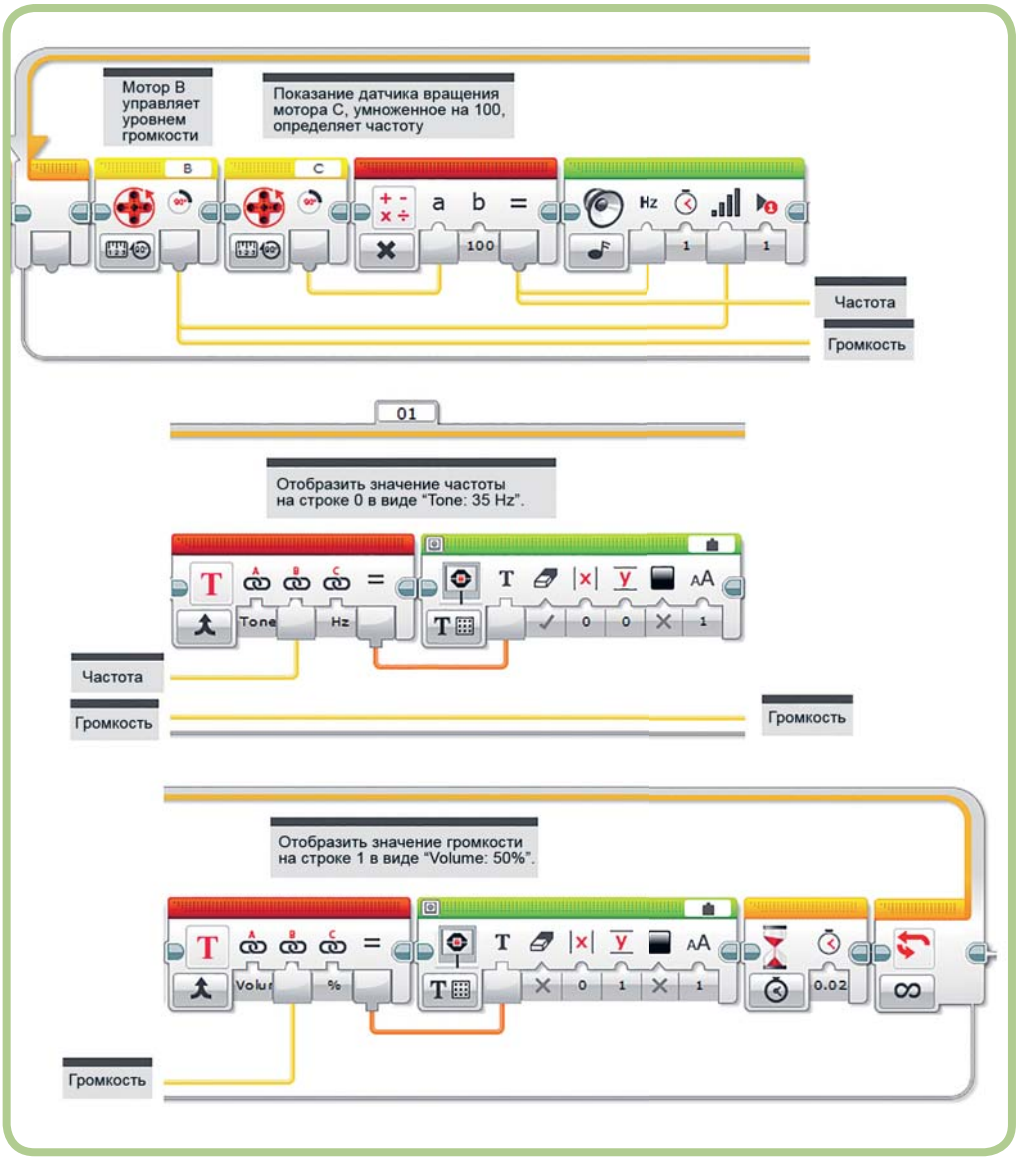


Рис. 8.25. Итоговая версия программы SoundMachine

ПРАКТИКУМ 8.4

Поворот колеса на 100° (чуть больше четверти оборота) позволяет охватить весь диапазон значений частоты. Это облегчает внесение больших изменений, но затрудняет тонкую настройку данного параметра. Для обеспечения большего контроля ты можешь изменить множитель, используемый в блоке **Математика** (Math). Если его значение меньше, вращение мотора медленнее изменяет частоту, что позволяет выполнить более точную настройку. Поэкспериментируй с разными значениями, чтобы найти наилучшую комбинацию.

Дальнейшее исследование

Выполни следующие действия, чтобы попрактиковаться в использовании шин данных:

1. Запусти программу *SoundMachine* с модулем, подключенным к программному обеспечению EV3. Наведи указатель мыши на шины данных для отображения их значений во время выполнения программы. Подумай, как ты можешь использовать эту функцию в процессе тестирования и отладки.

2. Параметр **Текущая мощность** (Current Power) блока **Вращение мотора** (Motor Rotation) измеряет скорость вращения мотора. Попробуй передать значение параметра **Текущая мощность** (Current Power) одного мотора на ввод **Мощность** (Power) второго мотора, чтобы второй мотор двигался так же, как и первый. Например, подключи подъемный рычаг к порту A модуля EV3 и напиши программу, которая использует параметр **Текущая мощность** (Current Power) мотора B для управления параметром **Мощность** (Power) блока **Средний мотор** (Medium Motor). При вращении колеса, прикрепленного к мотору B, подъемный рычаг должен повторять это же движение, подобно шлагбауму с дистанционным управлением.

3. Создай новую программу, благодаря которой робот TriBot будет следовать за инфракрасным маяком (входит в домашнюю версию конструктора). Этого можно добиться, изменив программу *GentleStop* так, чтобы она использовала инфракрасный датчик в режиме **Измерение** (Measure) ⇒ **Маяк** (Beacon) и передав значение **Направление маяка** (Beacon Heading) на ввод **Рулевое управление** (Steering) блока **Рулевое управление** (Move Steering). Подумай, как датчик цвета используется в программе *LineFollower*.

Заключение

Шины данных передают информацию из одного блока в другой, позволяя изменять параметры блока во время выполнения программы. В этой главе рассмотрены программы и несколько простых способов использования шин данных, а также блоки, предназначенные для работы с ними. Для того чтобы сделать показания датчиков доступными для других блоков программы, и нужны шины данных. Блоки **Математика** (Math) и **Текст** (Text) используются практически только с шинами данных для изменения данных или преобразования их типа.

Из гл. 9, 10 ты узнаешь о том, как применять шины данных с блоками **Переключатель** (Switch) и **Цикл** (Loop). Это отличная возможность попрактиковаться в использовании шин данных, ведь ты часто будешь применять их при создании программ в остальной части этой книги.

9

Шины данных и блок «Переключатель»

Блок **Переключатель** (Switch) можно применять для принятия решений о том, какие блоки следует запускать, выбирая между двумя или более альтернативными вариантами. Например, в программе *WallFollower* из гл. 7 используется блок **Переключатель** (Switch), считывающий показание инфракрасного датчика и решающий, приблизить робота TriBot к стене или отдалить от нее. Из этой главы ты узнаешь о том, как использовать блок **Переключатель** (Switch) для принятия решений на основе значения, переданного с помощью шины данных. Ты научишься передавать данные между блоками, находящимися внутри блока **Переключатель** (Switch), и теми, которые находятся до или после него.

Режимы блока «Переключатель»

До сих пор мы применяли блок **Переключатель** (Switch) для принятия решения на основе показания датчика. Кроме того, блок **Переключатель** (Switch) может принимать решение, исходя из значения, переданного с помощью шины данных. Режимы **Текст** (Text), **Логическое значение** (Logic) и **Числовое значение** (Numeric) (рис. 9.1) позволяют создавать случаи, которые соответствуют различным возможным значениям, переданным в блок **Переключатель** (Switch) с помощью шины данных. Единственным различием между этими тремя режимами является тип используемых данных.

В каждом из этих режимов блок **Переключатель** (Switch) имеет один-единственный параметр, значение которого, как правило, задается с помощью шины данных. В процессе выполнения блок выбирает необходимый случай, исходя из значения, содержащегося в шине данных. Например, на рис. 9.2 показан блок **Переключатель** (Switch) в режиме **Числовое значение** (Numeric) с предусмотренными случаями для значений 5, 7 и 9. В процессе выполнения в блоке оценивается значение в шине данных: если оно равно 5, блок выбирает верхний случай, если значение 7 — средний случай, а если значение 9 — нижний случай. При получении других значений используется верхний случай, поскольку он задан по умолчанию.

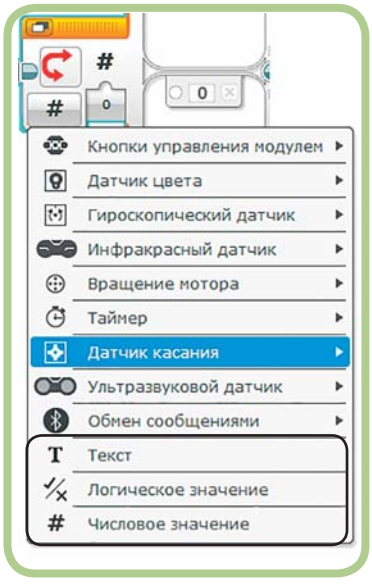


*Рис. 9.1. Режимы **Текст** (Text), **Логическое значение** (Logic) и **Числовое значение** (Numeric) блока **Переключатель** (Switch)*

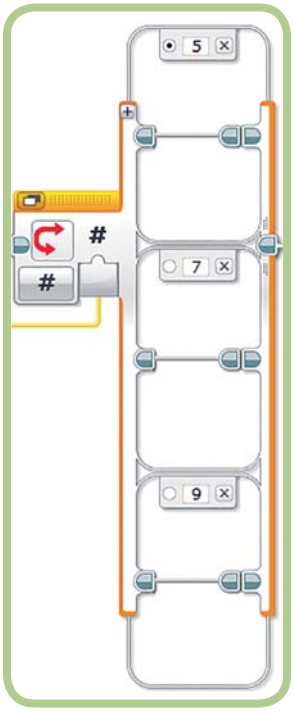


Рис. 9.2. Выбор случая на основе числового значения

При использовании режима **Числовое значение** (Numeric) значения должны быть целыми. Например, если ввести 5.25 для обозначения случая, это значение будет изменено на 5. В процессе выполнения блок **Переключатель** (Switch) округляет число из шины данных в меньшую сторону до ближайшего целого числа. Таким образом, в приведенном выше примере средний случай будет соответствовать любому значению от 7 до 8, включая 7, но не включая 8.

Режим **Текст** (Text) работает так же, как и режим **Числовое значение** (Numeric) за исключением того, что вместо чисел для каждого случая вводятся значения в виде букв или слов. Для режимов **Числовое значение** (Numeric) и **Текст** (Text) можно создать любое количество уникальных случаев. Однако логических значений всего два: «**Истина**» (“True”) и «**Ложь**» (“False”), поэтому режим **Логическое значение** (Logic) всегда предусматривает только два случая — ни больше ни меньше.

Переписываем программу GentleStop

Следует помнить о том, что программа *GentleStop* заставляет робота TriBot замедлять движение и постепенно останавливаться по мере приближения к стене. В первой версии программы робот останавливался при нажатии кнопки датчика касания, однако программу можно улучшить, обеспечив остановку моторов, когда робот будет находиться у самой стены, т. е. не дожидаясь нажатия кнопки датчика касания. В этом разделе ты перепишешь программу *GentleStop*, чтобы обеспечить именно такое поведение робота, используя выходные данные блока **Инфракрасный датчик** (Infrared Sensor) в качестве триггера для блока **Переключатель** (Switch).

ПРИМЕЧАНИЕ При использовании образовательной версии конструктора замени инфракрасный датчик и блок **Инфракрасный датчик** (Infrared Sensor) на ультразвуковой датчик и блок **Ультразвуковой датчик** (Ultrasonic Sensor). Программа работает одинаково хорошо с любым датчиком.

Сначала сделаем так, чтобы робот TriBot двигался вперед при значении параметра **Мощность** (Power), равном 75, а затем останавливался, как только показание инфракрасного датчика опустится ниже 20. Далее добавим шину данных, чтобы блок **Инфракрасный датчик** (Infrared Sensor) управлял значением параметра **Мощность** (Power), как в исходной программе. Это заставит робота двигаться вперед (при значении параметра **Мощность** (Power), соответствующем расстоянию между роботом и стеной), до тех пор, пока измеряемое инфракрасным датчиком расстояние не составит 20 или менее. В этот момент робот остановится.

В листинге 9.1 приведен псевдокод для программы, согласно которому робот TriBot будет двигаться вперед, пока значение инфракрасного датчика не опустится ниже 20, после чего моторы будут остановлены:

Листинг 9.1. Псевдокод для новой версии программы GentleStop

```
begin loop
   read the value from the Infrared Sensor
   if the distance > 20 then
      move forward, use the Infrared Sensor
         measurement for the Power parameter
   else
      stop moving
   end if
loop until the Touch Sensor is pressed
```

Начни с добавления и настройки блоков в соответствии с рис. 9.3. Блок **Инфракрасный датчик** (Infrared Sensor) использует режим **Сравнение** (Compare) ⇒ **Приближение** (Proximity) для считывания показания датчика и сравнения параметра **Приближение** (Proximity) с пороговым значением, чтобы выяснить, превышает ли оно 20. Результатом этого сравнения является логическое значение, поэтому выбери для блока **Переключатель** (Switch) режим **Логическое значение** (Logic), чтобы он принимал решение на его основе. Результат сравнения передается блоку **Переключатель**

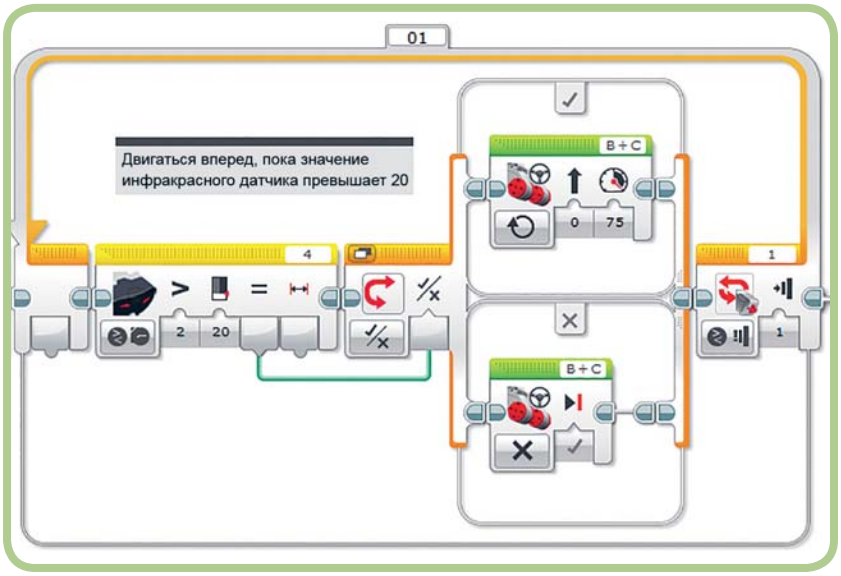


Рис. 9.3. Движение вперед, пока расстояние больше 20

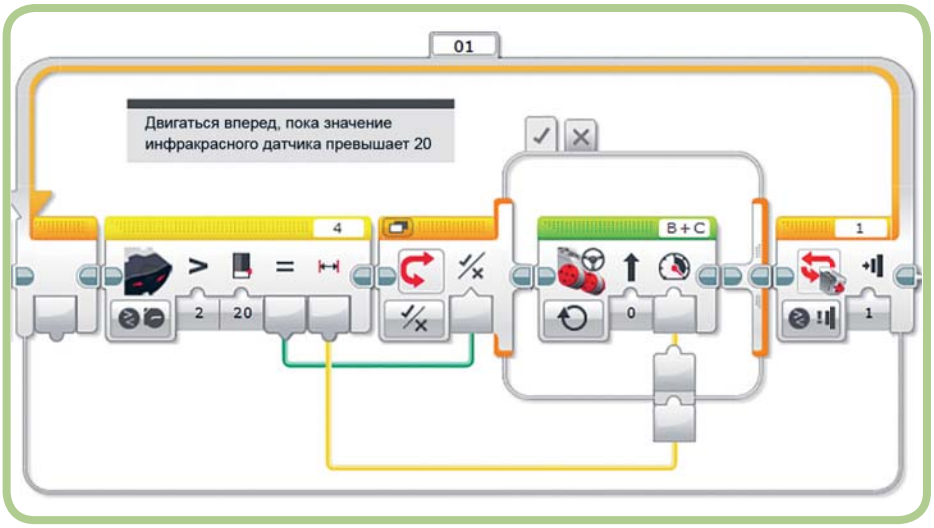


*Рис. 9.4. Соединение параметров **Приближение** (Proximity) и **Мощность** (Power) с помощью шины данных*

(Switch) с помощью шины данных, подключенной к выводу **Результат сравнения** (Compare Result) блока **Инфракрасный датчик** (Infrared Sensor). Если значение истинно, т. е. расстояние до стены больше 20, блок **Переключатель** (Switch) выполняет блок **Рулевое управление** (Move Steering) в верхнем случае, перемещая робота вперед. Если расстояние равно или меньше 20, блок **Переключатель** (Switch) использует блок **Рулевое управление** (Move Steering) в нижнем случае, и робот останавливается.

ПРИМЕЧАНИЕ Для блока **Ультразвуковой датчик** (Ultrasonic Sensor) выбери режим **Измерение** (Measure) ⇒ **Расстояние в сантиметрах** (Distance Centimeters) и задай для параметра **Пороговое значение** (Threshold value) значение 15.

Далее подключим еще одну шину данных к одному из случаев блока **Переключатель** (Switch), чтобы использовать значение параметра **Приближение** (Proximity) для управления параметром **Мощность** (Power).

Передача данных в блок «Переключатель»

Теперь, используя шину данных, мы сделаем так, чтобы скорость робота TriBot зависела от расстояния между ним и стеной. Чтобы соединить с помощью шины данных наружную часть блока **Переключатель** (Switch) со случаем внутри него, нужно изменить режим отображения блока **Переключатель** (Switch) на **Вид с вкладками** (Tabbed View), нажав кнопку **Плоский вид/Вид с вкладками** (Flat/Tabbed View). В этом режиме можно добавить шину данных, соединив вывод **Приближение** (Proximity) блока **Инфракрасный датчик** (Infrared Sensor) с вводом **Мощность** (Power) блока **Рулевое управление** (Move Steering), как показано на рис. 9.4.

При перетаскивании шины данных через границу блока **Переключатель** (Switch) добавляются два блока ввода, называемые *туннелем*, — по одному блоку снаружи и внутри границы. Ввод располагается внутри блока **Переключатель** (Switch) для каждой вкладки, и в каждом случае ты можешь решить, использовать его или нет. В программе *GentleStop* значение параметра **Приближение** (Proximity) не используется в случае, в котором предусмотрена остановка моторов (рис. 9.5), поэтому ввод блока остается неподключенным.

Рис. 9.5. Шина данных не используется для случая «Ложь»

После запуска эта версия программы должна работать как исходная за исключением того, что робот TriBot должен останавливать моторы, как только приблизится к стене, вместо того, чтобы врезаться в нее. Еще одно отличие заключается в том, что исходная программа завершалась, когда робот ударялся о стену, что приводило к нажатию кнопки датчика касания. В данной версии программа продолжает работать после остановки робота, поскольку кнопка датчика касания никогда не нажимается. Мы решим эту проблему в гл. 10.

Преимущества использования блока датчика

В этой версии программы *GentleStop* используем блок **Инфракрасный датчик** (Infrared Sensor) для проведения измерения, сравнения показания с пороговым значением и передачи результата в блок **Переключатель** (Switch) вместо того, чтобы проводить измерение с помощью блока **Переключатель** (Switch) в режиме **Инфракрасный датчик** (Infrared Sensor) ⇒ **Сравнение** (Compare) ⇒ **Приближение** (Proximity). У такого способа есть несколько преимуществ по сравнению с применением блока **Переключатель** (Switch):

- Блок датчика предоставляет доступ к показанию датчика, а также к результату сравнения. Мы использовали его в программе *GentleStop*, где значение параметра **Приближение** (Proximity) применялось для управления параметром **Мощность** (Power) блока **Рулевое управление** (Move Steering), а значение параметра **Результат сравнения** (Compare Result) — для остановки робота.
- Возможно, ты решишь использовать более сложное условие, чем сравнение «больше или меньше». Используя шины данных, блоки **Математика** (Math), **Логические операции** (Logic Operations) и **Сравнение** (Compare) (см. гл. 13), можно проверить практически любое возможное условие.
- Значение, применяемое в блоке **Переключатель** (Switch), также можно передать другим блокам программы и использовать для управления другими действиями робота.

Передача данных из блока «Переключатель»

Теперь создадим программу *LogicToText* (рис. 9.6) в качестве простого примера передачи данных из блока **Переключатель** (Switch). Эта программа определяет состояние кнопки датчика касания и отображает на экране модуля слово “True”, если кнопка нажата, и “False”, если не нажата. Оба случая блока **Переключатель** (Switch) содержат блок **Текст** (Text) с выводом, подключенным к блоку **Экран** (Display). Обрати внимание на то, что блоки **Текст** (Text) должны быть подключены к блоку **Экран** (Display) в обоих случаях.

Для создания этой программы настрой блоки в соответствии с рис. 9.7. В блоке **Датчик касания** (Touch Sensor) использует выбранный по умолчанию режим **Измерение** (Measure) ⇒ **Состояние** (State), и результат измерения применяется в качестве порогового значения для блока **Переключатель** (Switch). Блок **Текст** (Text) в верхнем случае блока **Переключатель** (Switch) генерирует текстовое значение “True”, а блок в нижнем случае генерирует значение “False”. Для отображения этого текста активируй для блока **Экран** (Display) режим **Текст** (Text) ⇒ **Сетка** (Grid) и выбери вариант **Проводной** (Wired) в поле, в которое обычно вводится текст.

Теперь подключи выводы блоков **Текст** (Text) к блоку **Экран** (Display), выполнив следующие действия:

1. Щелкни по кнопке **Плоский вид/Вид с вкладками** (Flat/Tabbed View) в блоке **Переключатель** (Switch). Блоки **Переключатель** (Switch) и **Экран** (Display) должны выглядеть так, как показано на рис. 9.8.

2. Соедини с помощью шины данных вывод графы **Результат** (Result) блока **Текст** (Text) с вводом графы **Текст** (Text) блока **Экран** (Display) (рис. 9.9). Обрати внимание: там, где шина данных пересекает границу блока **Переключатель** (Switch), создается два блока вывода.

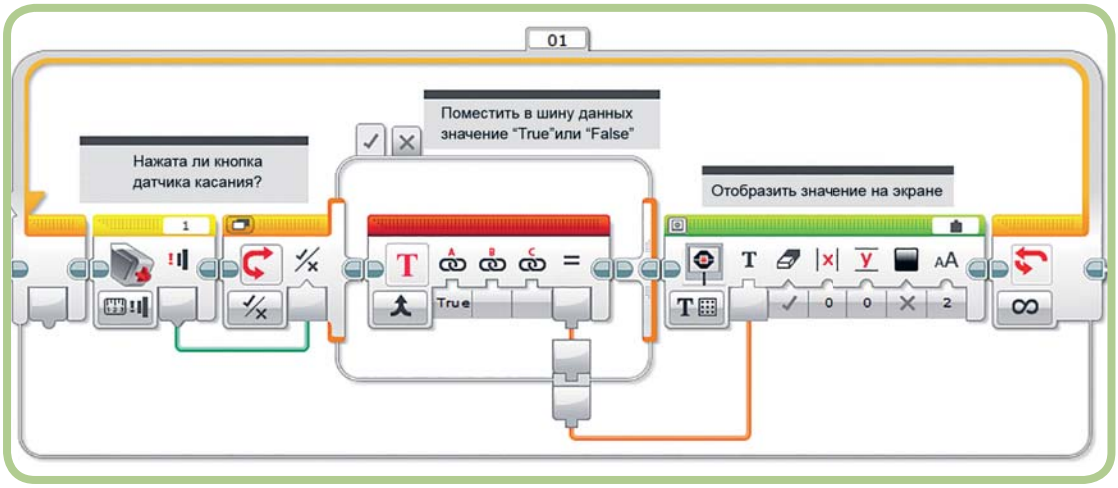


Рис. 9.6. Программа LogicToText

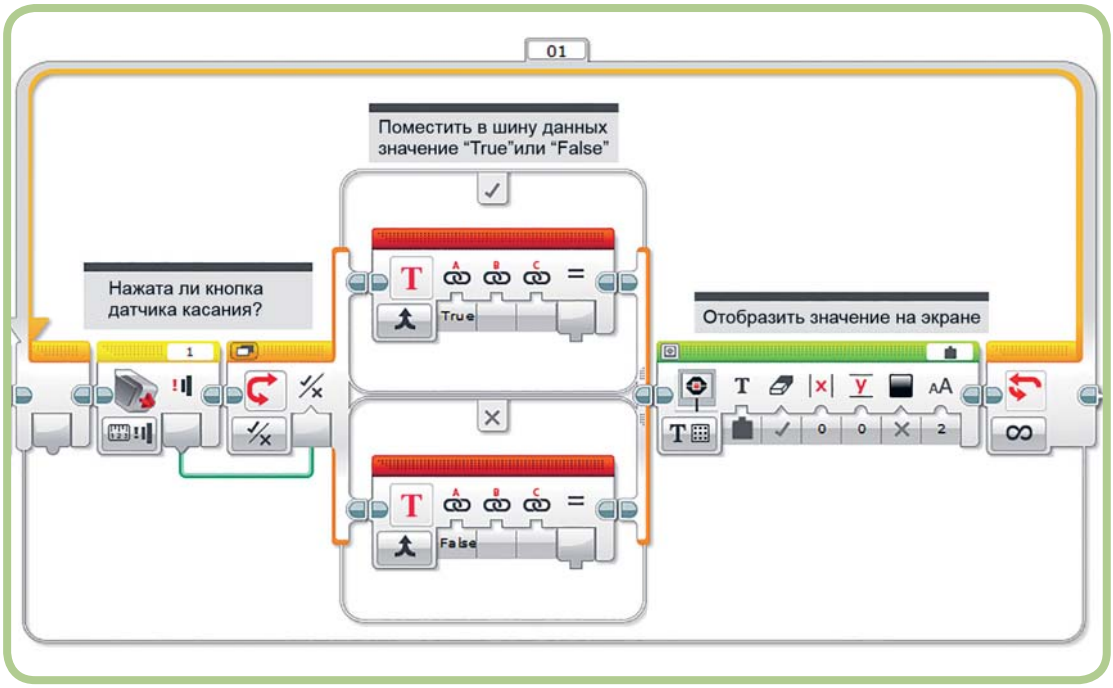


Рис. 9.7. Отправная точка для программы LogicToText

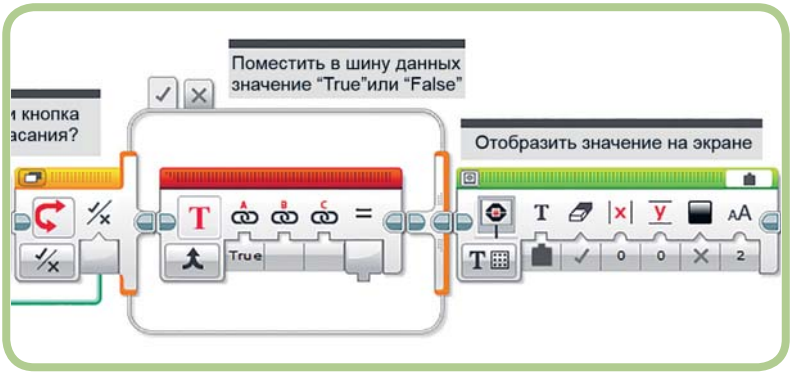


*Рис. 9.8. Блок **Переключатель** (Switch) в режиме отображения **Вид с вкладками** (Tabbed View)*

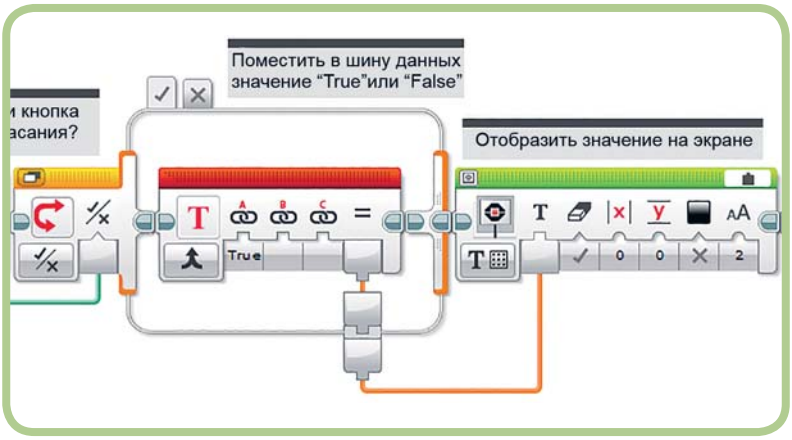


Рис. 9.9. Подключение первой шины данных

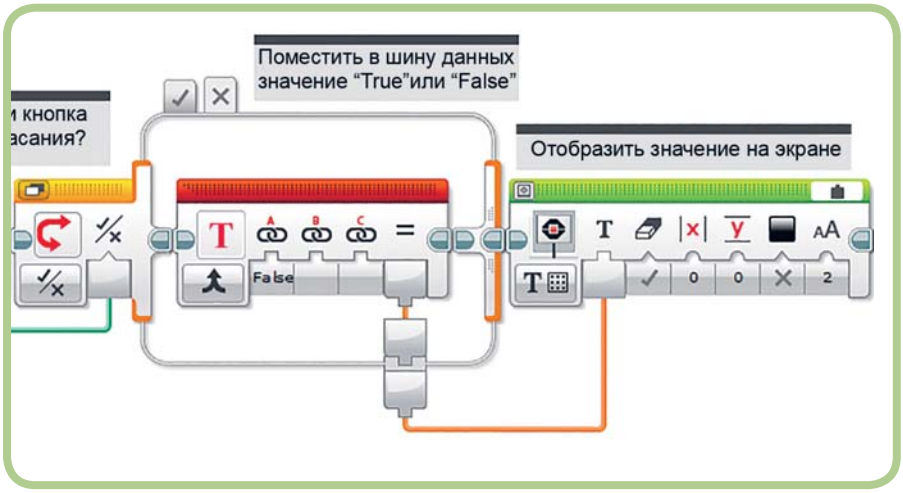


Рис. 9.10. Подключение второго блока ***Текст*** *к блоку* ***Экран*** *(Display)*

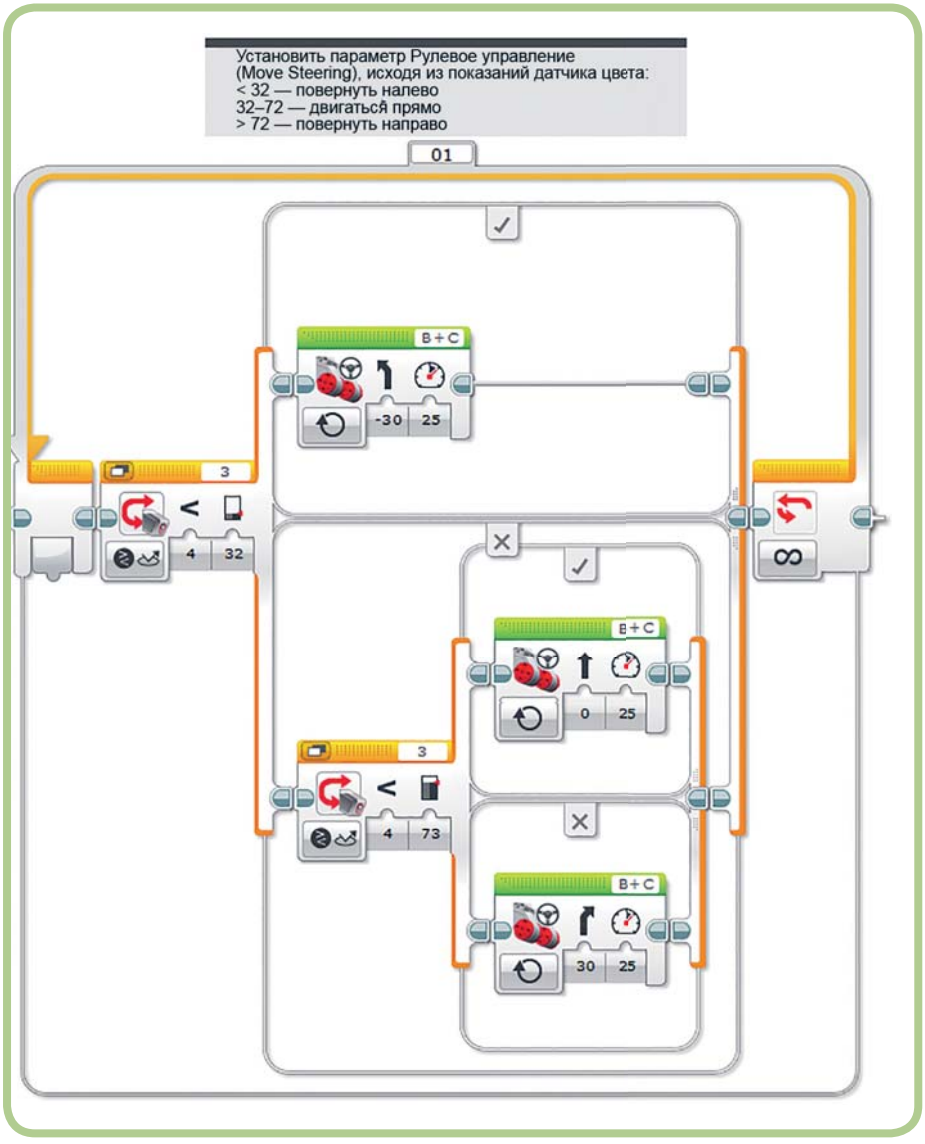


Рис. 9.11. Программа LineFollower, в которой используются вложенные блоки ***Переключатель*** *(Switch)*

3. Щелкни по вкладке **X** в верхней части блока **Переключатель** (Switch), чтобы отобразить другой случай.

4. Подключи вывод графы **Результат** (Result) блока **Текст** (Text) к выводу блока на границе блока **Переключатель** (Switch). Эта часть программы теперь должна выглядеть так, как показано на рис. 9.10.

В процессе выполнения в блоке **Переключатель** (Switch) используется только одна вкладка (относящаяся к случаю «Истина», либо к случаю «Ложь») и соответствующий вывод блока **Текст** (Text) передается на ввод блока **Экран** (Display). Попробуй запустить программу; на экране модуля должно отобразиться слово "True", когда кнопка датчика касания нажата, и "False", когда не нажата.

ПРАКТИКУМ 9.1

Напиши программу *ColorToText* на основе программы *LogicToText*, которая преобразует цвет, обнаруженный датчиком цвета, в соответствующую текстовую строку ("No Color" («Нет цвета»), "White" («Белый»), "Green" («Зеленый») и т. д.).

Упрощаем программу LineFollower

Программа *LineFollower*, о которой шла речь в гл. 6 (рис. 9.11), использует вложенные блоки **Переключатель** (Switch) для выбора одного из трех блоков **Рулевое управление** (Move Steering). Один блок заставляет робота поворачивать влево, другой — двигаться прямо, а третий — поворачивать вправо.

В блоке **Переключатель** (Switch) в режиме **Датчик** (Sensor) ⇒ **Сравнение** (Compare) может быть предусмотрено только два случая: показание датчика соответствует заданным критериям или нет. Для выбора одного из трех случаев (как в вышеприведенной программе) необходимы два вложенных блока **Переключатель** (Switch). В то же время для выбора одного из пяти случаев придется использовать четыре вложенных блока **Переключатель** (Switch). Как видишь, все это может быстро выйти из-под контроля.

Поскольку блок **Переключатель** (Switch) в режиме **Числовое значение** (Numeric) может иметь любое количество случаев, можно заменить группу вложенных блоков **Переключатель** (Switch), в которых используется один режим датчика, единственным блоком **Переключатель** (Switch) в режиме **Числовое значение** (Numeric). При режиме **Числовое значение** (Numeric) можно добавить любое количество случаев, не используя вложенные блоки. Тем не менее непрактично было бы создавать новый случай для каждого возможного показания датчика (скажем, от 1 до 100). Идея заключается в том, чтобы сгруппировать все ожидаемые значения в небольшие наборы чисел. Для этого в программировании часто используется подход *бининг* (binning), позволяющий разделить диапазон показаний датчиков на небольшие интервалы. В этой версии программы *LineFollower* ты разделишь диапазон ожидаемых показаний датчика на три интервала: один набор значений будет заставлять робота поворачивать влево, другой набор будет заставлять его двигаться прямо, а третий — поворачивать вправо.

Для деления диапазона на интервалы потребуются три числа: самое низкое и самое высокое значение, которое мы ожидаем получить от датчика, и количество интервалов. По результатам тестирования датчика цвета, проведенное с линией и фоном в гл. 6, наименьшее значение, которое можно ожидать, составляет 13, а наибольшее — 92. На рис. 9.12 этот диапазон возможных показаний датчика представлен графически. При выполнении программы *LineFollower* ожидаем, что все показания датчика цвета попадут в интервал, отмеченный серым прямоугольником.

Рис. 9.12. Диапазон ожидаемых значений

Первым этапом бининга является сдвиг диапазона влево так, чтобы он начинался с 0 (рис. 9.13). Чтобы это сделать, вычти из показания датчика 13 (самое низкое из ожидаемых значений). В результате этой операции ты получишь значение, принадлежащее диапазону от 0 до 79. Для корректного выполнения следующего этапа диапазон значений должен начинаться с 0.

Рис. 9.13. Диапазон значений, начинающийся с нуля

Второй этап предполагает деление диапазона на три интервала (рис. 9.14). Для того чтобы упростить выполнение математических операций, пронумеруем их 0, 1 и 2. В результате ожидаем получить 79 значений. Можно разделить это число на 3, чтобы определить размер каждого интервала: 79 / 3 = 26,33333. Однако с целыми числами работать удобнее, чем с десятичными. Кроме того, граница между интервалами не обязательно должна быть точной, поэтому округлим полученное значение до 27. Чтобы перейти от показания датчика к номеру интервала, достаточно вычесть 13 для получения значения в диапазоне от 0 до 79, а затем разделить его на 27. В результате ты получишь десятичное число, однако после его передачи в блок **Переключатель** (Switch) оно будет округлено в меньшую сторону до ближайшего целого числа, которое будет соответствовать номеру интервала. Например, если ты разделишь 60 на 27, то получишь значение 2,22, которое блок **Переключатель** (Switch) округлит до 2, чтобы сообщить тебе правильный номер интервала.

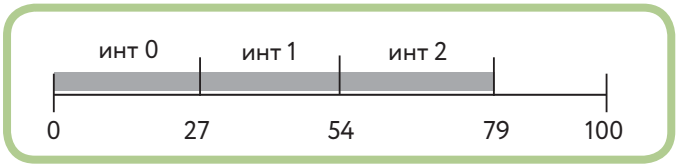


Рис. 9.14. Деление диапазона на три интервала

В новой версии программы содержится один блок **Переключатель** (Switch) с тремя случаями — по одному для каждого блока **Рулевое управление** (Move Steering). Значение для каждого случая соответствует номерам интервалов (0, 1 и 2). Программа Процесс бининга применяется для преобразования показания датчика цвета в номер одного из трех случаев. Так, программа сначала вычитает из него 13, а затем делит полученную разность на 27 (и округляет результат). Это можно выразить в виде следующей формулы:

Номер случая = (Показание датчика — 13) / 27.

В табл. 9.1 указан интервал значений для каждого случая и соответствующее ему поведение программы. Перечисленные здесь интервалы несколько отличаются от используемых в гл. 6, поскольку тогда их границы определялись другим способом.

Табл. 9.1. Поведение программы, основанное на показании датчика цвета

Показание датчика цвета	Номер случая	Поведение программы
13–39	0	Поворот налево
40–66	1	Движение прямо
67–92	2	Поворот направо

Теперь перепишем программу. В первой ее части применяется блок **Датчик цвета** (Color Sensor) для считывания показания датчика и два блока **Математика** (Math) для вычитания числа 13 и деления на 27 (рис. 9.15). В блоке **Датчик цвета** (Color Sensor) используется режим **Измерение** (Measure) ⇒ **Яркость отраженного света** (Reflected Light Intensity), как и в исходной программе.

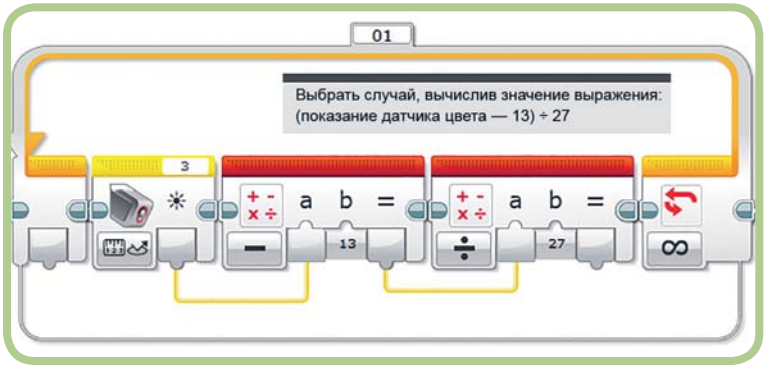


Рис. 9.15. Преобразование показания датчика в номер интервала

Во второй части программы используется блок **Переключатель** (Switch) в режиме **Числовое значение** (Numeric) для сопоставления номера интервала с соответствующим блоком **Рулевое управление** (Move Steering) (рис. 9.16). В моем случае в блоке **Переключатель** (Switch) выбран режим представления **Плоский вид** (Flat View), чтобы были видны все три случая; в своей собственной программе ты можешь использовать **Вид с вкладками** (Tabbed View) для экономии места. Нижний случай (соответствующий интервалу 2) выбран по умолчанию для обеспечения адекватного поведения программы при показании датчика, превышающем 92.

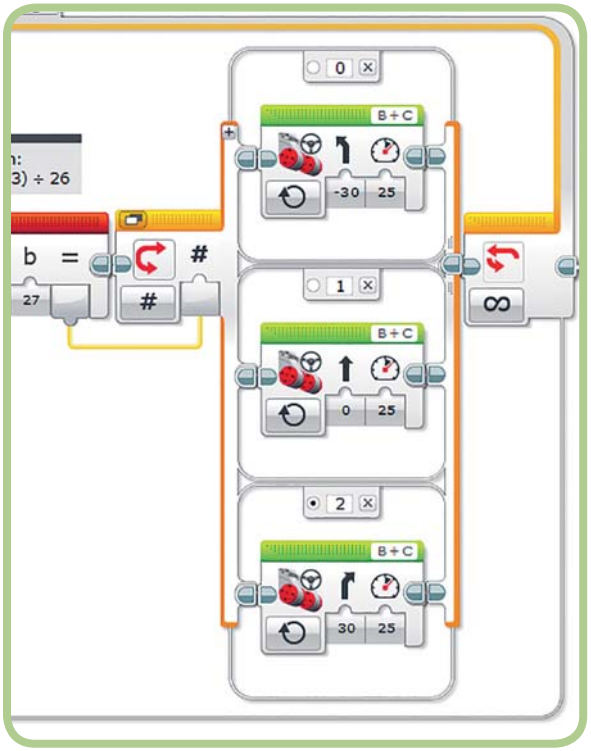


Рис. 9.16. Движение робота, зависящее от номера интервала

Что произойдет, если показание датчика будет меньше 13? В этом случае результатом выполнения второго блока **Математика** (Math) станет небольшое отрицательное число, которое округлится до 0 блоком **Переключатель** (Switch), благодаря этому будет выбран правильный случай. Получается, что до тех пор, пока показание датчика находится в пределах половины интервала (в данном примере: 27/2 = 13,5), если считать от наименьшего ожидаемого значения, программа работает корректно.

В данном случае все возможные показания, не превышающие наименьшего значения (0–13), находятся в пределах половины интервала, поэтому процесс бининга затрагивает и их, однако имей это в виду при использовании бининга в программах с другими диапазонами. Очень важно выбрать правильные значения для наибольших и наименьших ожидаемых показаний датчика, так как показание, находящееся за пределами этого диапазона, может вызвать неожиданное поведение программы.

После запуска программа должна вести себя так же, как и предыдущая версия. По сути, мы упростили блок **Переключатель** (Switch) благодаря добавлению блоков **Математика** (Math). В целом программа стала более элегантной и теперь ее легче усовершенствовать (например так — см. ниже).

ПРАКТИКУМ 9.2

Доработай программу *LineFollower* так, чтобы робот поворачивал плавно, находясь рядом с линией, и совершал более резкий поворот, находясь на большем расстоянии от линии. Это предполагает пять случаев с поворотами двух типов для каждой стороны. Все, что необходимо сделать, — изменить значение второго блока **Математика** (Math) с 27 на 16 и добавить еще два случая в блок **Переключатель** (Switch).

Этот практикум очень похож на практикум 6.1, в котором предлагалось расширить программу до пяти случаев с использованием вложенных блоков **Переключатель** (Switch). Сравни количество усилий, необходимых для изменения программы с помощью каждого из подходов (вложенные блоки **Переключатель** (Switch) и бининг), а также размер и визуальную сложность программ в обоих случаях.

Дальнейшее исследование

Далее перечислены некоторые варианты, как можно использовать шины данных с блоком **Переключатель** (Switch):

1. Поэкспериментируй с подключением шины данных к вводу и выводу блока **Переключатель** (Switch). Попробуй сделать следующее:
   - Измени режим представления блока **Переключатель** (Switch) на **Плоский вид** (Flat View) и попробуй подключить шину данных к вводу и выводу блока.
   - Измени режим представления блока **Переключатель** (Switch) на **Вид с вкладками** (Tabbed View) и соедини с помощью шины данных блоки, находящиеся снаружи и внутри блока **Переключатель** (Switch). Затем щелкни по кнопке **Плоский вид/Вид с вкладками** (Flat/Tabbed View).
   - Подключи несколько шин данных к вводам блока **Переключатель** (Switch). Практикуйся в перетаскивании шин данных, чтобы посмотреть, как они могут располагаться внутри и снаружи блока **Переключатель** (Switch). В дополнение к перемещению шин данных ты можешь перетащить туннели туда, где шина пересекает границу блока **Переключатель** (Switch).

2. Измени программу *SoundMachine* так, чтобы на экране модуля отображались слова “Soft” («Тихо»), “Medium” («Средне»), “Loud” («Громко») или “Very Loud” («Очень громко») вместо значения громкости в процентах. Для выбора отображаемого текста раздели диапазон значений громкости на четыре интервала, используя процесс бининга.

3. Ты можешь использовать блок **Переключатель** (Switch) с шинами данных, чтобы ограничить значение. Например, блок датчика может сравнить показание с пороговым значением и передать результат блоку **Переключатель** (Switch) в режиме **Логическое значение** (Logic). В одном случае блок **Переключатель** (Switch) передавал бы значение из блока датчика через пустой блок **Переключатель** (Switch) и дальше через шину данных (рис. 9.17), в другом — максимальное значение.

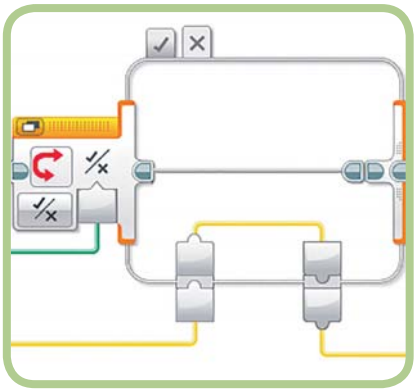


*Рис. 9.17. Передача значения через блок **Переключатель** (Switch) в неизмененном виде*

4. Напиши программу для перемещения робота TriBot, используя показатель яркости внешнего освещения для управления параметром **Мощность** (Power) блока **Рулевое управление** (Move Steering). Но при этом сделай так, чтобы значение не превышало 75.

Заключение

Применение шины данных для передачи информации в блок **Переключатель** (Switch) обеспечивает большую гибкость в плане решений, применяемых твоими программами. Используя блок датчика, ты можешь произвести сравнение за пределами блока **Переключатель** (Switch), а значит, принимать более сложные решения по сравнению с теми, которые поддерживает блок **Переключатель** (Switch). Кроме того, с помощью шин данных можно передавать информацию между блоками внутри блока **Переключатель** (Switch) и теми, которые находятся до или после него. В результате ты можешь легко настраивать каждый выбор или принимать решения, влияющие на остальную часть программы.

Используя в качестве входных данных числовое или текстовое значение, можно сделать так, чтобы блок **Переключатель** (Switch) выбирал более чем из двух возможных альтернатив. Это упрощает процесс принятия сложных решений программой и позволяет обходиться без глубоко вложенных блоков **Переключатель** (Switch). Для решения этой задачи очень часто применяется процесс бининга, используемый в программе *LineFollower*.

10

Шины данных и блок «Цикл»

Эта глава посвящена применению двух специальных функций блока **Цикл** (Loop), которые применяются с шинами данных. С помощью режима **Логическое значение** (Logic) блока **Цикл** (Loop) можно гибко настраивать время завершения цикла, а благодаря выводу **Параметр цикла** (Loop Index) можно узнать, сколько раз этот цикл выполнялся.

Режим «Логическое значение»

С помощью режима **Логическое значение** (Logic) блока **Цикл** (Loop) можно выбрать момент выхода из цикла, используя логическое значение из шины данных. На рис. 10.1 показан блок **Цикл** (Loop) с режимом **Логическое значение** (Logic) и подключенной шиной данных. После выполнения блоков тела цикла происходит проверка значения в шине данных. Если в ней содержится значение "False", цикл продолжает выполняться и снова запускает блоки, содержащиеся в теле цикла. Если в шине данных содержится значение "True", — цикл завершается. Условие цикла всегда проверяется после выполнения его тела, поэтому содержащиеся в нем блоки, даже если в шине данных изначально содержится значение "True", запускаются как минимум один раз.

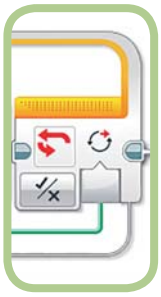


*Рис. 10.1. Режим **Логическое значение** (Logic) блока **Цикл** (Loop)*

Для большинства программ достаточно той гибкости, которая обеспечивается режимами датчиков блока **Цикл** (Loop), однако в некоторых ситуациях удобнее использовать режим **Логическое значение** (Logic). Например, если у тебя уже есть блок, позволяющий производить нужное сравнение, можно просто передать результат этого сравнения блоку **Цикл** (Loop). Программа *GentleStop* использует блок **Инфракрасный датчик** (Infrared Sensor) для определения расстояния до стены, его также можно использовать для выбора момента выхода из цикла. Другим примером такой ситуации является принятие решения на основе показаний нескольких датчиков, например об остановке цикла, если будет нажата кнопка датчика касания *или* если инфракрасный датчик обнаружит объект ближе, чем на заданном расстоянии (подробнее о принятии подобных решений — см. гл. 13).

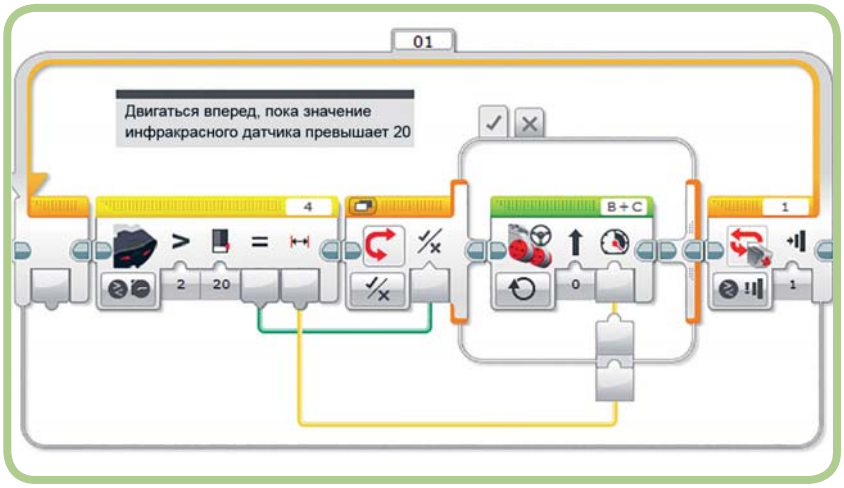


Рис. 10.2. Программа GentleStop из гл. 8

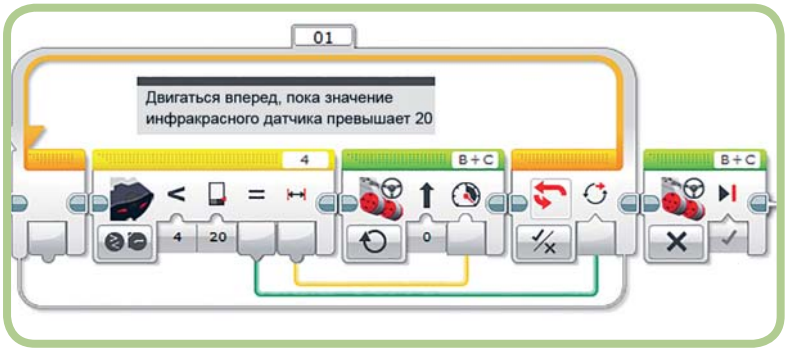


Рис. 10.3. Программа GentleStop, в которой используется режим ***Логическое значение*** *(Logic)*

Используя режим **Логическое значение** (Logic), ты можешь переписать программу *GentleStop* из гл. 8 (рис. 10.2) и сделать ее более простой. Вместо того чтобы использовать блок **Переключатель** (Switch) для определения момента остановки моторов, для блока **Цикл** (Loop) можно выбрать режим **Логическое значение** (Logic) и выйти из цикла, когда робот TriBot приблизится к стене. Это означает, что программа больше не будет останавливаться при столкновении робота с препятствием. Однако, поскольку программа была изменена так, чтобы моторы останавливались до того, как робот достигнет стены. Но это не должно быть проблемой, если между роботом и стеной нет препятствий.

Обновленная версия программы показана на рис. 10.3. Помимо удаления блока **Переключатель** (Switch) и перемещения блоков **Рулевое управление** (Move Steering) необходимо выбрать для блока **Цикл** (Loop) режим **Логическое значение** (Logic) и изменить тип сравнения, используемый блоком **Инфракрасный датчик** (Infrared Sensor). Поскольку выход из блока **Цикл** (Loop) происходит, когда в шине данных содержится значение "True", необходимо выбрать тип сравнения **Меньше** (Less Than), чтобы значением стало "False" до тех пор, пока робот не приблизится к стене. Обрати внимание на то, что параметр **Мощность** (Power) блока **Рулевое управление** (Move Steering) по-прежнему зависит от значения параметра **Приближение** (Proximity) инфракрасного датчика, поэтому робот все равно должен замедляться по мере приближения к стене.

Загрузи и запусти программу. Она должна функционировать почти так же, как и предыдущая версия, заставляя робота сначала быстро двигаться вперед, затем замедляться и, наконец, останавливаться рядом со стеной. На самом деле новая версия ведет себя чуть лучше, чем предыдущая, поскольку программа завершается после остановки робота. Программа, в которой используется блок **Переключатель** (Switch), продолжает работать, несмотря на то, что робот уже не двигается.

ПРАКТИКУМ 10.1

Добавь в программу *GentleStop* звуковой эффект, используя расстояние до стены для управления громкостью. При запуске программы звук должен звучать громко и постепенно затихать по мере приближения робота TriBot к стене.

Вывод «Параметр цикла»

Вывод **Параметр цикла** (Loop Index) отслеживает число повторений тела цикла и находится с левой стороны блока **Цикл** (Loop) (рис. 10.4). При первом выполнении цикла его значение равно 0. Каждый раз, когда программа возвращается к началу тела цикла, значение этого параметра увеличивается на единицу, а после завершения цикла — оказывается на единицу меньше числа повторов цикла, так как оно не обновляется после его последнего запуска.

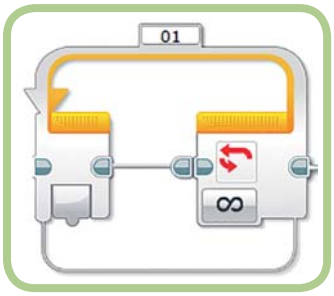


Рис. 10.4. Вывод ***Параметр цикла*** *(Loop Index)*

Программа LoopIndexTest

Программа *LoopIndexTest* (рис. 10.5) демонстрирует поведение вывода **Параметр цикла** (Loop Index). В блоке **Цикл** (Loop) используется режим **Подсчет** (Count), настроенный на пять повторов. При каждом выполнении цикла блок **Экран** (Display) отображает значение вывода **Параметр цикла** (Loop Index), а блок **Ожидание** (Wait) обеспечивает короткую паузу, благодаря чему ты можешь прочитать это значение. После запуска данной программы на экране должны отображаться значения: «0», «1», «2», «3» и «4».

Эта и две следующие программы не очень интересны (ты вряд ли впечатлишь своих друзей, создав робота, который умеет считать до четырех). Однако они демонстрируют, как работает блок **Цикл** (Loop). Приступая к работе с новыми, не использованными ранее блоками или функциями, полезно писать такие небольшие программы, помогающие изучить принципы их работы. Это позволит использовать данные блоки и функции в более сложных программах.

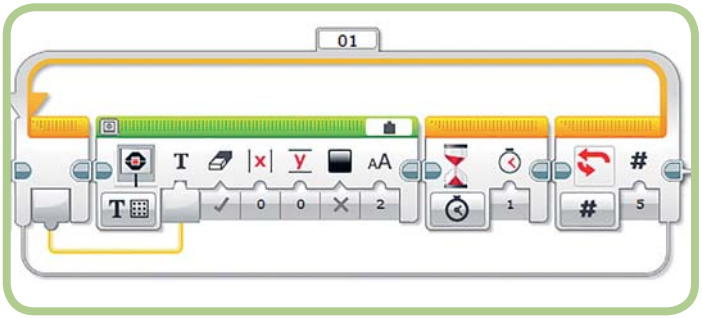


Рис. 10.5. Программа LoopIndexTest

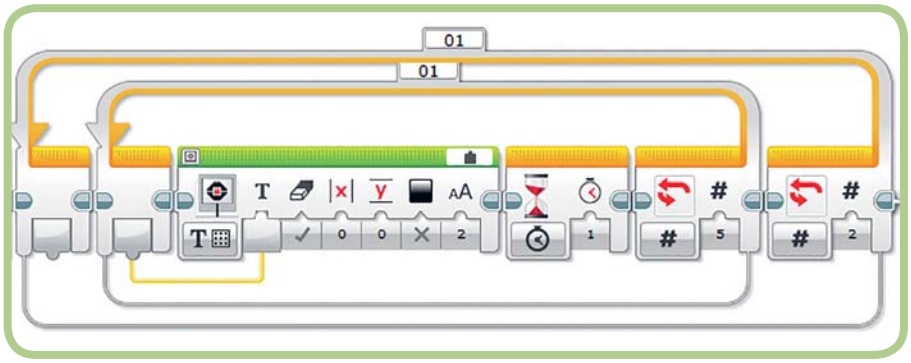


Рис. 10.6. Программа LoopIndexTest2

Повторный запуск цикла

Программа *LoopIndexTest2*, показанная на рис. 10.6, иллюстрирует принцип работы вывода **Параметр цикла** (Loop Index), когда он вложен в другой блок **Цикл** (Loop). Внешний блок **Цикл** (Loop) настроен на два повтора.

При первом запуске внутреннего цикла, как и случае с программой *LoopIndexTest*, на экране должны отображаться значения «0», «1», «2», «3» и «4». Но что произойдет при втором выполнении внутреннего цикла? Робот снова отобразит значения «0», «1», «2», «3» и «4» или продолжит подсчет и выведет на экран значения «5», «6», «7», «8» и «9»?

При запуске программы на экране дважды отобразится последовательность значений «0», «1», «2», «3» и «4». Значит, значение вывода **Параметр цикла** (Loop Index) сбрасывается на 0 каждый раз при достижении вложенного блока **Цикл** (Loop).

Итоговое значение вывода «Параметр цикла»

Программы *LoopIndexTest* и *LoopIndexTest2* используют вывод **Параметр цикла** (Loop Index) в теле цикла. Это же значение может быть установлено в блоках, которые следуют за блоком **Цикл** (Loop), как показано в программе *LoopIndexTest3* (рис. 10.7). Программа *LoopIndexTest3* выполняет цикл **Цикл** (Loop) пять раз, а затем выводит на экран модуля EV3 итоговое значение вывода **Параметр цикла** (Loop Index).

Данная программа отображает значение вывода **Параметр цикла** (Loop Index), которое передается с помощью шины данных после последнего выполнения цикла. Это значение будет на единицу меньше общего числа повторов цикла. Помни, что после последнего выполнения цикла значение вывода **Параметр цикла** (Loop Index) не обновляется, и программа просто переходит к следующему блоку. Поскольку блок **Цикл** (Loop) настроен на пять повторов, после запуска программа должна отобразить на экране модуля значение «4».

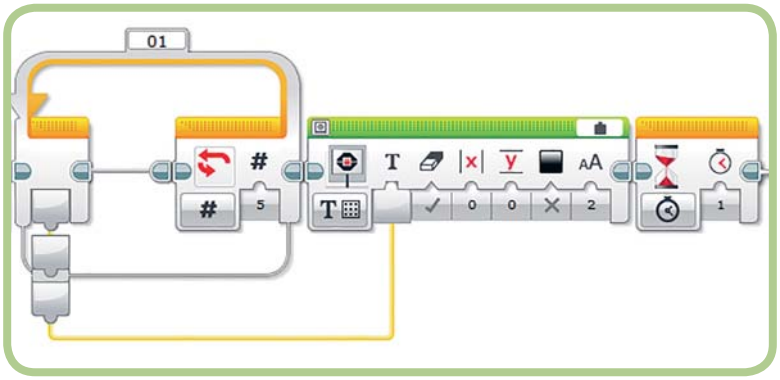


Рис. 10.7. Программа LoopIndexTest3

ПРИМЕЧАНИЕ **В мире компьютерного программирования принято начинать отсчет с нуля, а не с единицы. Обычно это не представляет большой проблемы. Однако из-за этого можно легко столкнуться с *ошибкой неучтенной единицы*, когда количество повторов цикла оказывается на один больше или меньше заданного.**

Программа SpiralLineFinder

С помощью программы *LineFinder* из гл. 5 (рис. 10.8) робот TriBot двигается вперед в поиске темной линии. Программа работает нормально, если изначально направить робота в правильном направлении. Ты можешь сделать ее более эффективной, направив робота двигаться по спирали, а не по прямой линии.

В прямоугольной спирали (рис. 10.9) каждый сегмент длиннее предыдущего, что создает постоянно расширяющуюся траекторию, начиная от начальной центральной точки. Благодаря программе *SpiralLineFinder* робот двигается по прямоугольной спиральной траектории и останавливается, как только датчик цвета распознает линию.

Движение по спирали

Для движения по спирали робот TriBot должен повторять следующие действия: переместиться вперед; повернуть на 90°; переместиться вперед на чуть большее расстояние, чем в первый раз; повернуть на 90° и т. д. Сейчас уже очевидно, что применение блока **Цикл** (Loop) — самый лучший способ обеспечить повторение подобных действий. Ранее мы уже использовали блоки **Рулевое управление** (Move Steering), чтобы заставить робота двигаться вперед и поворачивать. Здесь добавляется новая задача — изменить расстояние, на которое перемещается робот при каждом повторении цикла, с помощью вывода **Параметр цикла** (Loop Index).

На рис. 10.10 показан один из способов, с помощью которого можно заставить робота двигаться по спирали. Для первого блока **Рулевое управление** (Move Steering) выбран режим **Включить на количество оборотов** (On for Rotations), число оборотов задается значением вывода **Параметр цикла** (Loop Index). При каждом выполнении цикла с помощью первого блока робот TriBot перемещается чуть дальше, чем в предыдущий раз. Благодаря второму блоку **Рулевое управление** (Move Steering) робот поворачивает на 90°.

ПРИМЕЧАНИЕ **Применяя шины из образовательной версии конструктора, задай для параметра Продолжительность** (Duration) **второго блока Рулевое управление** (Move Steering) **значение 160° вместо 210°.**

Загрузи и запусти эту программу и посмотри, что произойдет. Робот двигается по прямоугольной спиральной траектории, однако неожиданно то, что первым действием робота является поворот, а не движение вперед. Почему так происходит?

Помни о том, что начальным значением вывода **Параметр цикла** (Loop Index) является 0. Это означает, что при первом выполнении цикла первый блок **Рулевое управление** (Move Steering) настроен на перемещение робота на 0 оборотов

Рис. 10.8. Программа LineFinder (см. гл. 5)

Рис. 10.9. Прямоугольная спираль

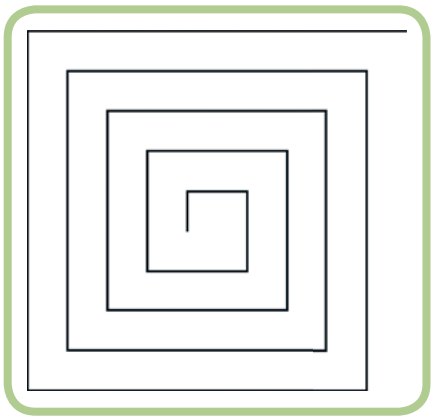


Рис. 10.10. Движение по прямоугольной спирали

мотора, поэтому робот вообще не двигается. В случае с этой программой не важно, куда направлен робот в самом начале, поэтому, если он повернет один раз, прежде чем двигаться вперед, проблем возникнуть не должно. Если ты хочешь изменить это, можешь использовать блок **Математика** (Math) для прибавления 1 к значению вывода **Параметр цикла** (Loop Index), прежде чем передавать значение в блок **Рулевое управление** (Move Steering).

Немного изменим программу *SpiralLineFollower* так, чтобы в процессе движения вперед она постоянно проверяла, есть ли линия. Мы по-прежнему будем использовать значение вывода **Параметр цикла** (Loop Index) для контроля над пройденным роботом расстоянием, но передадим это значение блоку **Цикл** (Loop), как будет показано далее.

Обнаружение линии при движении по спирали

Программа *LineFinder* (см. рис. 10.8) использует блок **Ожидание** (Wait) в режиме **Датчик цвета** (Color Sensor) ⇒ **Яркость отраженного света** (Reflected Light Intensity) для обнаружения линии, при этом блок **Рулевое управление** (Move Steering) в режиме **Включить** (On) заставляет робота продолжать движение вперед. Однако программа *SpiralLineFollower* не может просто использовать блок **Рулевое управление** (Move Steering) в режиме **Включить** (On), поскольку нам нужно, чтобы робот двигался по более сложной траектории.

Мы будем использовать блок **Переключатель** (Switch) внутри блока **Цикл** (Loop) для проверки показания датчика цвета, пока робот движется вперед, и выбора момента остановки мотора. Начни с добавления блоков в соответствии с рис. 10.11. В первом блоке сбрасывается показание датчика вращения мотора B на 0. Затем с помощью блока **Рулевое управление** (Move Steering) в режиме **Включить** (On) робот начинает движение вперед, с помощью блока **Цикл** (Loop) моторы работают до тех пор, пока показание датчика вращения не будет равно значению вывода **Параметр цикла** (Loop Index). Если это значение равно 2, робот двигается вперед, пока мотор B не совершит два оборота. После выполнения блока **Цикл** (Loop) благодаря блоку **Рулевое управление** (Move Steering) робот поворачивает.

На данном этапе в программе должно быть прописано движение робота по спирали, как и в предыдущей программе (см. рис. 10.10). Однако на этот раз мы будем использовать блок **Цикл** (Loop), в котором есть место для блока **Переключатель** (Switch), с помощью него мы будем проверять показание датчика цвета. Удостоверься в том, что программа работает, как нужно, перед добавлением остальной части кода.

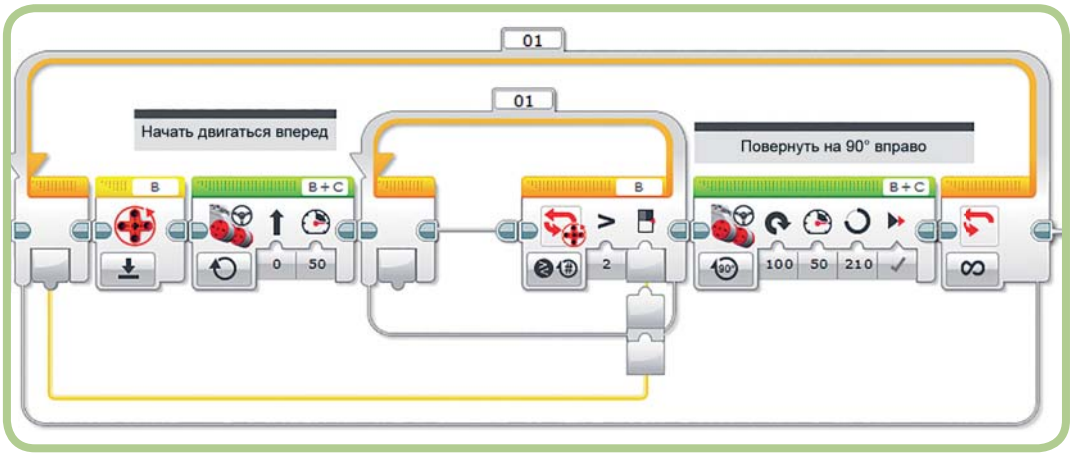


Рис. 10.11. Использование цикла и показания датчика вращения мотора для перемещения робота вперед

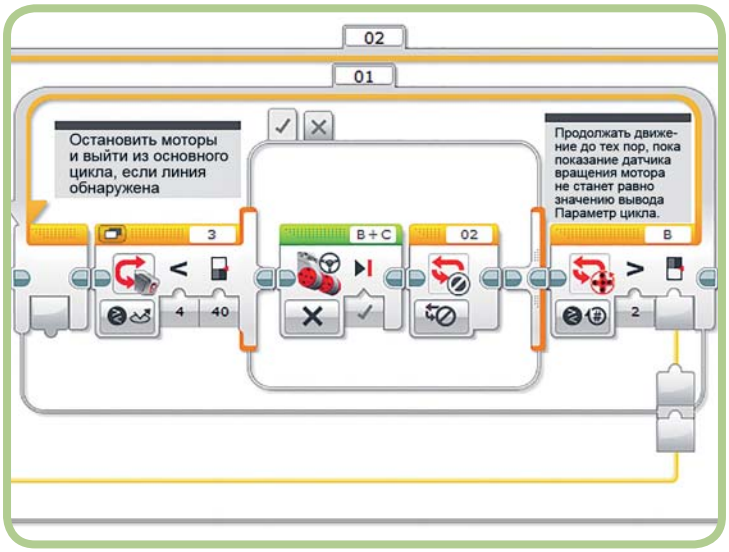


Рис. 10.12. Остановка моторов и выход из цикла после обнаружения линии

Далее необходимо добавить блок **Переключатель** (Switch) в блок **Цикл** (Loop). Блок **Переключатель** (Switch) проверяет показания датчика цвета и останавливает моторы, если яркость отраженного света не превышает порогового значения (это означает, что робот обнаружил линию). Для того чтобы остановить программу после обнаружения линии, мы будем использовать блок **Прерывание цикла** (Loop Interrupt) для осуществления выхода из внешнего цикла.

На рис. 10.12 показан блок **Переключатель** (Switch), который необходимо добавить в программу. В этом блоке **Переключатель** (Switch) используется то же пороговое значение, что и в блоке **Ожидание** (Wait) в исходной программе. Когда условие истинно, моторы останавливаются, и цикл завершается. Для внешнего блока **Цикл** (Loop) задано имя *02*, в соответствии с которым настроен блок **Прерывание цикла** (Loop Interrupt).

После запуска программы робот TriBot должен двигаться по прямоугольной спиральной траектории вплоть до обнаружения темной линии, после чего он остановится, и программа завершится. Поэкспериментируй с разными значениями скорости, продолжительности блока, отвечающего за выполнение поворота, и пороговыми значениями блока **Переключатель** (Switch), чтобы определить параметры, которые дают наилучшие результаты для твоей тестовой области.

ПРАКТИКУМ 10.2

Расстояние между витками спирали зависит от того, насколько увеличивается длина каждого последующего отрезка пути, преодолеваемого роботом. При увеличении длины каждого отрезка на один оборот мотора расстояние между витками спирали равно двум оборотам. Чтобы убедиться в этом, нарисуй прямоугольную спираль на листе графической бумаги и оцени расстояние между соседними линиями. Для увеличения или уменьшения расстояния между ними, прежде чем передавать значение в блок Рулевое управление (Move Steering), умножь или раздели значение вывода Параметр цикла (Loop Index). При небольшой ширине целевой линии более плотная спираль уменьшит вероятность того, что робот ее пропустит. При использовании менее плотной спирали робот сможет быстрее исследовать обширную область. Поэкспериментируй с разными значениями и шириной целевой линии, чтобы посмотреть, как параметр Продолжительность (Duration) блока Рулевое управление (Move Steering) влияет на качество и скорость обнаружения этой линии.

Использование гироскопического датчика для выполнения более точных поворотов

Двигаясь по спиральной траектории в поисках линии, робот TriBot совершает много поворотов, однако его невозможно заставить каждый раз поворачивать ровно на 90°. Придется поэкспериментировать с разными значениями продолжительности, чтобы максимально приблизиться к повороту на 90°. Однако движения робота повторяются не вполне точно, и на них влияет слишком много других факторов (например, уровень заряда аккумулятора, гладкость пола, собачья шерсть на колесах и др.). При каждом повороте за угол появляется небольшая погрешность, которая накапливается по мере работы программы. Со временем ты можешь заметить, что прямоугольник спирали начинает слегка отклоняться в сторону.

С помощью гироскопического датчика, который входит в образовательную версию конструктора, но продается и отдельно, ты можешь уменьшить эту погрешность. С помощью гироскопического датчика можно определить момент, когда при повороте робота количество градусов достигает 90°, вместо того, чтобы каждый раз пытаться заставить робота поворачивать ровно на 90°. На рис. 10.13 показан блок **Рулевое управление** (Move Steering), отвечающий за выполнение поворота в исходной программе. На рис. 10.14 показаны альтернативные блоки, выполняющие ту же функцию, но использующие для этого гироскопический датчик.

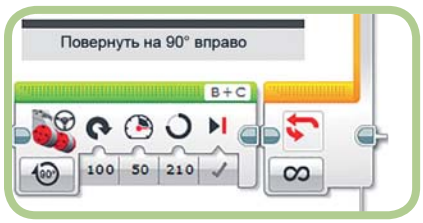


*Рис. 10.13. Выполнение поворота с помощью одного блока **Рулевое управление** (Move Steering)*

С помощью первого нового блока запускается робот TriBot, используется режим **Включить** (On), чтобы одновременно с этим в программе могли функционировать другие блоки во время движения робота. При каждом запуске цикла робот должен поворачивать на 90°. С помощью блока **Математика** (Math) значение вывода **Параметр цикла** (Loop Index) умножается на 90 для вычисления порогового значения, используемого блоком **Ожидание** (Wait). В блоке **Ожидание** (Wait) используется режим **Гироскопический датчик** (Gyro Sensor) ⇒ **Сравнение** (Compare) ⇒ **Угол** (Angle) для

Рис. 10.14. Поворот с помощью гироскопического датчика

сообщения о том, что робот переместился на достаточное расстояние. По этой причине при значении вывода **Параметр цикла** (Loop Index), равном 1, робот выполняет поворот до тех пор, пока показание гироскопического датчика не достигнет 90°. Когда значение вывода **Параметр цикла** (Loop Index) составляет 2, робот выполняет поворот до тех пор, пока показание гироскопического датчика не достигнет 90° × 2 или 180°. А это составляет еще один поворот на 90°, поскольку показание гироскопического датчика не сбрасывается ни на каком этапе. Конечный блок последовательности останавливает моторы.

Загрузи и протестируй программу. Ты обнаружишь, что траектория, по которой следует робот TriBot, со временем не отклоняется в сторону. В этой версии каждый совершаемый роботом поворот необязательно более точен, чем в версии, в которой используется единственный блок **Рулевое управление** (Move Steering). Робот по-прежнему может каждый раз поворачивать чуть более чем на 90°. Разница заключается в том, что в данной версии погрешности каждого поворота не накапливаются. При втором повороте робот поворачивает до тех пор, пока показание гироскопического датчика не станет равно 180° вне зависимости от каких-либо погрешностей, возникших при выполнении предыдущего поворота.

Ты также можешь заметить, что теперь робот сначала движется вперед, а не поворачивает. Это связано с тем, что начальным значением вывода **Параметр цикла** (Loop Index) является 0. Значит, при первом выполнении цикла робот поворачивает только до тех пор, пока показание гироскопического датчика больше или равно 0, поэтому поворот тут же прекращается.

Дальнейшее исследование

Выполни следующие действия, чтобы попрактиковаться в использовании блоков **Цикл** (Loop) с шинами данных:

1. Попробуй заставить робота TriBot двигаться по круговой, а не по прямоугольной спирали. Тебе придется корректировать значение параметра **Рулевое управление** (Steering) при каждом выполнении цикла, начав со значения около 100, и затем, постепенно уменьшая его, чтобы медленно увеличивать размер спирали.

2. Используй блок **Цикл** (Loop) для подсчета нажатий кнопок удаленного инфракрасного маяка. Необходимо лишь настроить блок **Ожидание** (Wait) на нажатие одной из двух кнопок, например кнопки 1 и кнопки 2. В случае нажатия кнопки 1 цикл должен повториться, а если нажата кнопка 2 — завершиться. После выхода из цикла значение вывода **Параметр цикла** (Loop Index) будет соответствовать количеству нажатий кнопки 1. Подсказка: тебе понадобятся два блока **Ожидание** (Wait). Первый — ожидание нажатия кнопки, а второй — изменения ее состояния. Без второго блока **Ожидание** (Wait) цикл будет повторяться несколько раз при каждом нажатии кнопки.

3. С помощью кода из последнего практикума используй количество нажатий кнопки для настройки какого-нибудь параметра далее в программе. Например, ты можешь задать уровень мощности блока **Рулевое управление** (Move Steering), чтобы протестировать программу при разных значениях скорости. В начале программы добавь код для подсчета количества нажатий кнопок, а затем умножь его на 10 и используй полученный результат в качестве значения скорости вращения мотора.

Заключение

В блоке **Цикл** (Loop) предусмотрено использование шины данных как для вывода **Параметр цикла** (Loop Index), так и для определения условия цикла. В этой главе мы изменили программу *GentleStop* так, чтобы информация из блока **Инфракрасный датчик** (Infrared Sensor) применялась для определения момента выхода из цикла. Шина данных потребуется тебе для задания условия цикла в гл. 13, в которой речь пойдет о том, как комбинировать показания нескольких датчиков.

Как только ты привыкнешь к тому, что первым значением вывода **Параметр цикла** (Loop Index) является 0, ты сможешь с легкостью использовать его для управления блоками в теле цикла. В программе *SpiralLineFinder* используется значение этого вывода для управления параметром **Продолжительность** (Duration) блока **Рулевое управление** (Move Steering), благодаря чему робот следует по спиральной траектории. Вывод **Параметр цикла** (Loop Index) можно использовать, когда требуется, чтобы значение какого-то параметра блока увеличивалось или уменьшалось на единицу при каждом выполнении цикла.

11

Переменные

С помощью переменных ты можешь сохранять значение, чтобы затем использовать его в программе. Например, ты хочешь считать показание датчика, чтобы в будущем сравнить его с другими показаниями того же датчика. Для этого следует сохранить первое показание датчика в переменной, к которой можно было бы получить доступ позже для того, чтобы сравнить ее значение с другими показаниями. Эта глава посвящена применению переменных, в ней также описаны типы задач, которые можно решить с их использованием. Я также расскажу о блоке **Константа** (Constant), с помощью которого можно использовать одно значение для управления несколькими блоками программы.

Блок «Переменная»

Думай о *переменной* как о ячейке памяти модуля, в которой можно хранить значение. С помощью блока **Переменная** (Variable), содержащегося на вкладке, в которой содержатся блоки операций с данными, хранится или извлекается значение переменной. В переменной можно сохранить любое значение из шины данных, т. е. сделать *запись в переменную*. Далее в программе ты можешь извлечь это значение (это называется *чтением значения*) и использовать его в качестве входных данных для других блоков.

Для того чтобы разобраться в том, как применяются переменные, создадим программу *VariableTest*, с помощью которой в переменной сохраняются показания датчика цвета, считывается и на экране модуля отображается значение из переменной. В блоке **Датчик цвета** (Color Sensor) в режиме **Измерение** (Measure) ⇒ **Цвет** (Color) отображается показание, которое сохраняется с помощью блока **Переменная** (Variable).

Начнем с того, что активируем в блоке **Переменная** (Variable) режим, определяющий операцию над переменной (чтение или запись), а также ее тип данных (рис. 11.1). Для сохранения показания датчика цвета (которое представляет собой число), выбери режим **Записать** (Write) ⇒ **Числовое значение** (Numeric). В этой главе используются только простые типы данных (текст, число и логическое значение). Массивы подробно рассмотрены в гл. 15.

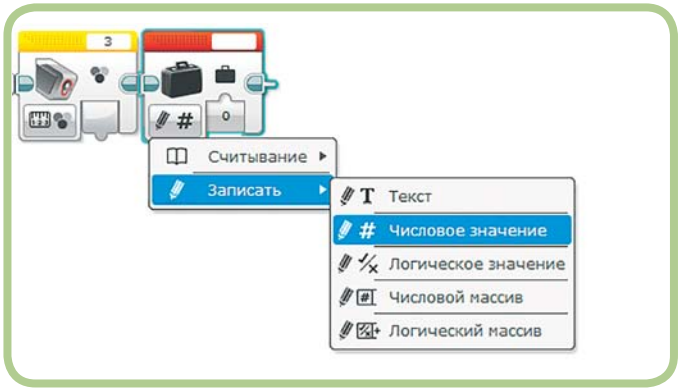


*Рис. 11.1. Выбор режима блока **Переменная** (Variable)*

Операция **Записать** (Write) сохраняет значение в переменной. Это значение можно ввести вручную или передать с помощью шины данных. В этой программе мы подключим вывод блока **Датчик цвета** (Color Sensor) к вводу блока **Переменная** (Variable) (рис. 11.2). В других программах ты можешь вручную задать начальное значение переменной, а затем использовать шину данных для ее изменения далее в программе.

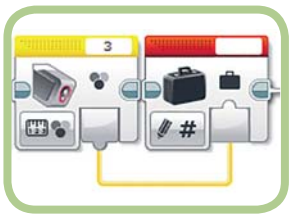


Рис. 11.2. Сохранение показания датчика цвета в переменной

После выбора режима задай имя переменной, щелкнув по полю в правом верхнем углу блока. Появится меню, в котором содержатся все определенные ранее переменные указанного типа, здесь также можно добавить новую переменную (рис. 11.3). Поскольку мы еще не добавляли никаких переменных в этот проект, единственным пунктом меню является **Добавить переменную** (Add Variable).

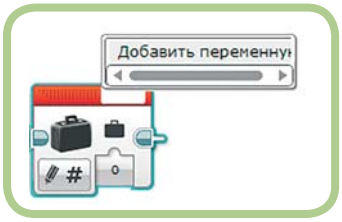


Рис. 11.3. Ввод имени переменной

После выбора команды **Добавить переменную** (Add Variable) появляется диалоговое окно (рис. 11.4). Введи имя для новой переменной. В каждой переменной могут храниться значения только одного типа (текст, число, логическое значение, числовой массив или логический массив). При создании переменной в режиме блока **Переменная** (Variable) определяется тип данных, которые могут храниться в переменной. Для целей этой программы назовем переменную **Color**.

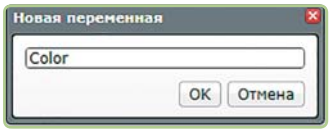


Рис. 11.4. Диалоговое окно для добавления новой переменной

Теперь необходимо добавить еще один блок **Переменная** (Variable) для чтения значения и блок **Экран** (Display) для отображения значения на экране модуля EV3. Блок **Ожидание** (Wait) в конце программы предотвращает очистку экрана, что позволяет тебе увидеть результат. Готовая программа показана на рис. 11.5.

После добавления в программу трех новых блоков выбери для нового блока **Переменная** (Variable) режим **Считывание** (Read) ⇒ **Числовое значение** (Numeric). В качестве имени переменной автоматически указывается Color, поскольку это единственная числовая переменная в проекте, однако в большинстве случаев тебе придется щелкнуть в поле с именем переменной и выбрать нужный вариант. Последним этапом создания этой программы является подключение вывода блока **Переменная** (Variable) к вводу **Текст** (Text) блока **Экран** (Display).

После запуска эта программа считывает и сохраняет в переменной Color показание датчика цвета, считывается и отображается на экране значение переменной. Чтобы увидеть значение, передаваемое в блок **Переменная** (Variable) и из него, подключи модуль к компьютеру и запусти программу из среды EV3. Это позволит тебе проверять значения в шинах данных во время выполнения программы, как показано в гл. 8 (возможно, ты решишь добавить более долгую паузу с помощью блока **Ожидание** (Wait), чтобы иметь больше времени на проверку значений).

Программа RedOrBlueCount

В этом разделе приведены этапы создания программы *RedOrBlueCount* на основе программы *RedOrBlue* из гл. 6 (рис. 11.6). В новой программе используются две переменные с именами Red Total и Blue Total для подсчета количества красных и синих объектов. По мере выполнения программы на экране EV3 будет отображаться нарастающий итог. После запуска программы это значение для обоих цветов равно нулю, поэтому на экране должны отобразиться значения “Red: 0” и “Blue: 0”. В дальнейшем по мере увеличения этих значений обновится содержимое экрана. В листинге 11.1 приведен псевдокод для программы. Фрагменты, которые необходимо добавить в исходную программу *RedOrBlue*, выделены жирным шрифтом. Обрати внимание на то, что программа не считает объекты, цвет которых отличается от красного и синего.

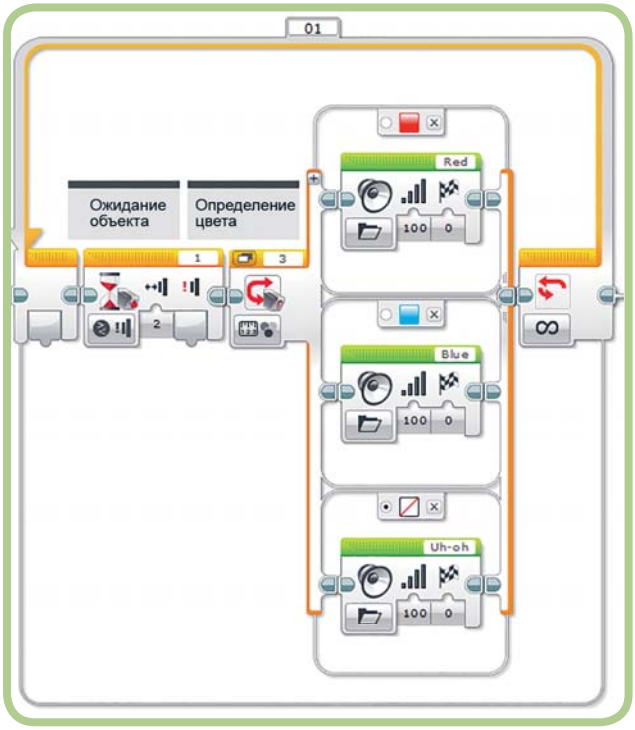


Рис. 11.6. Программа RedOrBlue

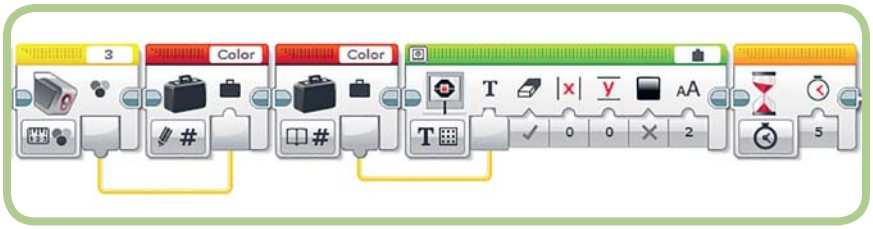


Рис. 11.5. Программа VariableTest

Листинг 11.1. *Программа RedOrBlueCount*

```
set Red Total to 0
set Blue Total to 0
display "Red: 0"
display "Blue: 0"
begin loop
   wait for the Touch Sensor to be bumped
   if the object is red then
      use a Sound block to say "Red"
      read the Red Total value
      add one to the Red Total value
      write the new value to Red Total
      display "Red: " followed by the Red Total
         value
   else if the object is blue then
      use a Sound block to say "Blue"
   read the Blue Total value
   add one to the Blue Total value
   write the new value to Blue Total
   display "Blue: " followed by the Blue Total
      value
   else
      use a Sound block to say "Uh-oh"
   end if
loop forever
```

Создание и инициализация переменных

Сначала необходимо создать две переменные и задать для каждой из них начальное значение. Этот процесс называется *инициализацией переменных*. Для подсчета красных объектов значение переменной Red Total должно начинаться с нуля и увеличиваться на единицу при обнаружении очередного красного объекта. При запуске программы значение переменной типа «число» по умолчанию — 0. Тем не менее лучше всегда задавать начальное значение переменной на случай, если позже ты решишь переместить код или повторно использовать некоторую его часть в других программах.

Начни создавать программу *RedOrBlueCount*, выполнив следующие действия:

1. Убедись в том, что проект *Chapter11* открыт.

2. Открой проект *Chapter6* и скопируй программу *RedOrBlue* в проект *Chapter11*. Переименуй новую программу (в проекте *Chapter11*) в *RedOrBlueCount*.

3. Добавь блок **Переменная** (Variable) в начало программы. Оставь выбранным режим **Записать** (Write) ⇒ **Числовое значение** (Numeric) со значением, равным 0 (режим и значение по умолчанию).

Щелкни в поле с именем переменной. Появится меню, содержащее пункт **Добавить переменную** (Add Variable) и все созданные ранее числовые переменные (рис. 11.8).

ВЫБОР ИМЕНИ ПЕРЕМЕННОЙ

Задавай переменной такое имя, чтобы программа была понятной. Например, имя Red Total в программе для подсчета красных и синих объектов содержит информацию о том, как используется данная переменная (это поможет и людям, читающим твою программу, и тебе, если позже ты решишь к ней вернуться). Имена переменных также полезно согласовывать между собой. Например, при использовании имени Red Total для обозначения количества красных объектов, для обозначения количества синих объектов тебе следует использовать Blue Total, а не что-то другое, вроде Blue Count. Точно так же важно избегать имен, которые сложно расшифровать, или слишком коротких сокращений, например TtlR или просто R. Кроме того, имей в виду, что пространство для хранения имени в блоке **Переменная** (Variable) ограничено, поэтому избегай использования излишне длинных имен переменных, которые начинаются одинаково. Например, если ты используешь имена Total Red и Total Blue, то блок **Переменная** (Variable) выглядит одинаково при выборе любого из этих имен (рис. 11.7), что затрудняет понимание действий программы. Чтобы увидеть полное имя переменной, наведи указатель мыши на поле с именем.

Рис. 11.7. Невозможно определить, какую переменную использует этот блок: Total Red или Total Blue

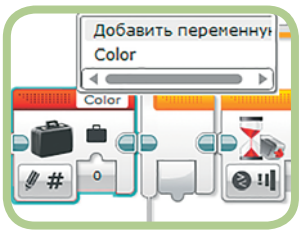


Рис. 11.8. Задание имени переменной

4. Выбери команду **Добавить переменную** (Add Variable), чтобы открыть соответствующее диалоговое окно. Введи Red Total (рис. 11.9) и щелкни по кнопке **OK**. Теперь в блоке **Переменная** (Variable) должны отображаться первые несколько букв имени переменной (рис. 11.10).

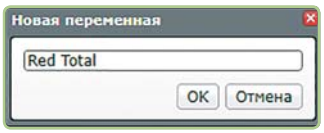


Рис. 11.9. Создание переменной Red Total

Рис. 11.10. Задание для переменной Red Total значения 0

5. Добавь еще один блок **Переменная** (Variable) справа от первого.

6. Введи имя переменной **Blue Total**.

На данном этапе начало программы должно выглядеть так, как показано на рис. 11.11. Эти два блока инициализируют две переменные нулем.

Рис. 11.11. Инициализация переменных Red Total и Blue Total нулем

Отображение исходных значений

Для отображения исходных значений помести два блока **Экран** (Display) перед блоком **Цикл** (Loop):

7. Добавь блок **Экран** (Display) после второго блока **Переменная** (Variable). Выбери режим **Текст** (Text) ⇒ **Сетка** (Grid) и задай значение **2** для параметра **Строка** (Row). Для параметра **Текст** (Text) задай значение **Red: 0**.

8. Добавь еще один блок **Экран** (Display) после первого. Выбери режим **Текст** (Text) ⇒ **Сетка** (Grid) и задай значение **4** для параметра **Строка** (Row). Для параметра **Текст** (Text) задай значение **Blue: 0**.

9. Выбери для параметра **Очистить экран** (Clear Screen) вариант **Ложь** (False).

Теперь начало программы должно выглядеть так, как показано на рис. 11.12.

Подсчет красных объектов

Необходимо, чтобы при обнаружении красного объекта программа прибавляла единицу к значению переменной Red Total и отображала на экране модуля новое значение. Для этого необходимы три блока: блок **Переменная** (Variable) для считывания текущего значения и его передачи с помощью шины данных; блок **Математика** (Math) для прибавления единицы к текущему значению; второй блок **Переменная** (Variable) для сохранения нового значения. Сделай следующее:

10. Добавь блок **Переменная** (Variable) в предусмотренный для красных объектов случай блока **Переключатель** (Switch) после блока **Звук** (Sound).

Выбери режим **Считывание** (Read) ⇒ **Числовое значение** (Numeric) и убедись в том, что в качестве имени переменной выбрано **Red Total**. Блок **Переключатель** (Switch) теперь должен выглядеть так, как показано на рис. 11.13.

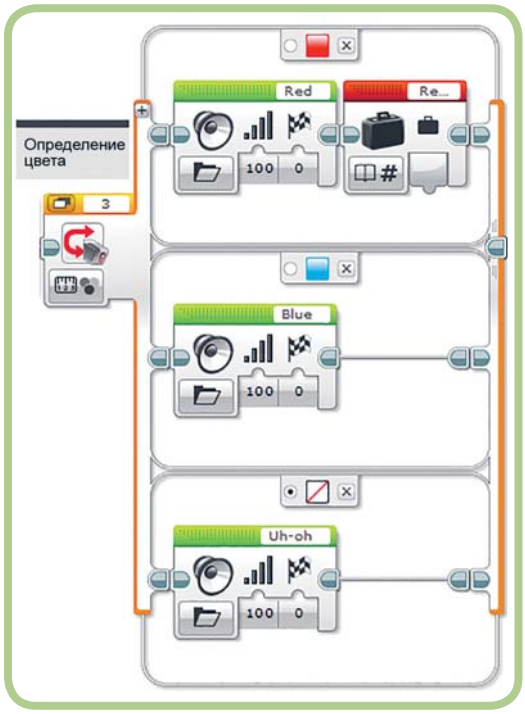


Рис. 11.13. Чтение текущего значения переменной Red Total

11. Добавь блок **Математика** (Math) после блока **Переменная** (Variable). Убедись, что в качестве значения *b* задана **1**.

12. Добавь еще один блок **Переменная** (Variable) после блока **Математика** (Math). Убедись в том, что для него выбран режим **Записать** (Write) ⇒ **Числовое значение** (Numeric). Выбери пункт **Red Total** в качестве имени переменной.

13. С помощью одной шины данных соедини вывод первого блока **Переменная** (Variable) с вводом *a* блока **Математика** (Math), а с помощью другой — вывод блока **Математика** (Math) с вводом второго блока **Переменная** (Variable), как показано на рис. 11.14.

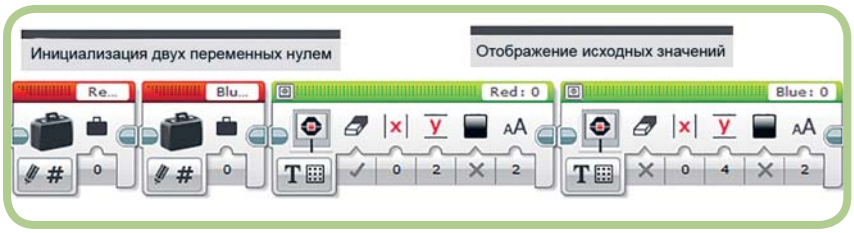


Рис. 11.12. Отображение исходных значений

После обновления итога должно отобразиться новое значение с помощью блоков **Текст** (Text) и **Экран** (Display). Этот же способ мы использовали в программе *SoundMachine* в гл. 8.

14. Добавь блок **Текст** (Text) после второго блока **Переменная** (Variable) и введи **Red:** в качестве значения *a*. Не забудь добавить пробел после двоеточия.

15. Подключи вывод **Результат** (Result) блока **Математика** (Math) к вводу *b* блока **Текст** (Text).

Добавь блок **Экран** (Display) после блока **Текст** (Text) и выбери режим **Текст** (Text) ⇒ **Сетка** (Grid). Для параметра **Строка** (Row) задай значение **2**. Обязательно выбери для параметра **Очистить экран** (Clear Screen) вариант **Ложь** (False), чтобы не стереть количество синих объектов при отображении количества красных.

Щелкни в текстовом поле в правом верхнем углу блока **Экран** (Display) и выбери в появившемся меню вариант **Проводной** (Wired).

Подключи вывод **Результат** (Result) блока **Текст** (Text) к вводу **Текст** (Text) блока **Экран** (Display).

Эта часть программы теперь должна выглядеть так, как показано на рис. 11.15.

Этот раздел кода содержит последовательность действий для обновления переменной, с которой ты будешь часто работать для чтения текущего значения (с помощью первого блока **Переменная** (Variable)), его изменения с помощью блока **Математика** (Math) и записи нового результата в переменную с помощью второго блока **Переменная** (Variable).

ПРИМЕЧАНИЕ **Прежде чем продолжить, протестируй свою программу, убедись в том, что она подсчитывает и отображает общее количество красных объектов. Тебе предстоит продублировать этот код для случая, соответствующего синим объектам, поэтому, если в нем содержится ошибка, лучше исправить ее сейчас.**

Подсчет синих объектов

Код для подсчета синих объектов практически идентичен коду для подсчета красных объектов, поэтому вместо написания кода вручную ты можешь скопировать его и внести несколько изменений. Вот как можно продублировать код:

16. Выбери пять новых блоков в верхнем случае блока **Переключатель** (Switch), начиная с первого блока **Переменная** (Variable) и заканчивая блоком **Экран** (Display), нарисовав вокруг них рамку выделения или выделив блок **Переменная** (Variable), а затем щелкнув по остальным блокам, удерживая нажатой клавишу **Shift**.

17. Нажав и удерживая нажатой клавишу **Ctrl**, щелкни по одному из блоков и перетащи его в нижний случай. При перетаскивании блоков при нажатой клавише **Ctrl** они копируются, а не просто перемещаются.

Теперь, когда блоки на месте, внеси следующие изменения:

18. В обоих блоках **Переменная** (Variable) выбери переменную **Blue Total** вместо **Red Total**.

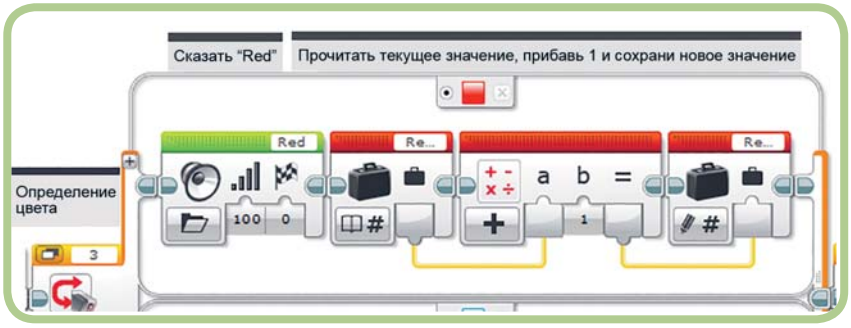


Рис. 11.14. Прибавление единицы к значению переменной Red Total

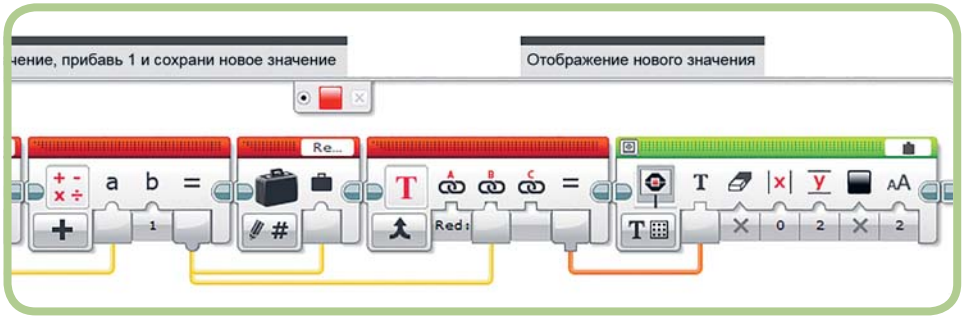


Рис. 11.15. Отображение нового значения переменной Red Total

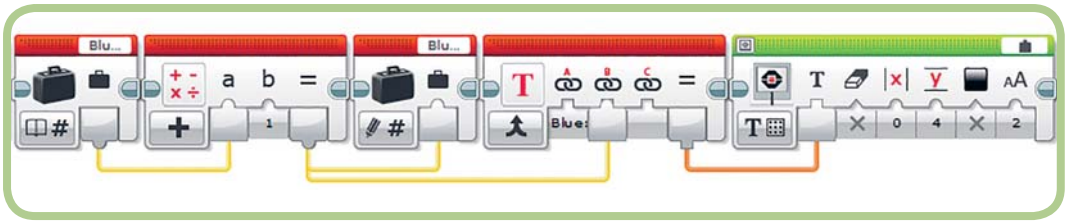


Рис. 11.16. Подсчет синих объектов

19. В блоке **Текст** (Text) задай текст метки **Blue:** (с пробелом после двоеточия).

20. Для параметра **Строка** (Row) блока **Экран** (Display) задай значение **4**.

На рис. 11.16 показан данный раздел программы с блоками в нижнем случае.

После копирования блоков и изменения параметров программа должна правильно подсчитывать и отображать количество красных и синих объектов. Загрузи и протестируй программу, чтобы убедиться в том, что она работает для объектов обоих цветов.

Управление переменными с помощью страницы «Свойства проекта»

Для программы *RedOrBlueCount* мы создали две переменные с помощью двух блоков **Переменная** (Variable). Переменные также можно создавать и удалять, используя вкладку **Переменные** (Variables) на странице **Свойства проекта** (Project Properties), для открытия которой щелкни по значку в виде гаечного ключа слева от вкладки с именем программы (рис. 11.17).

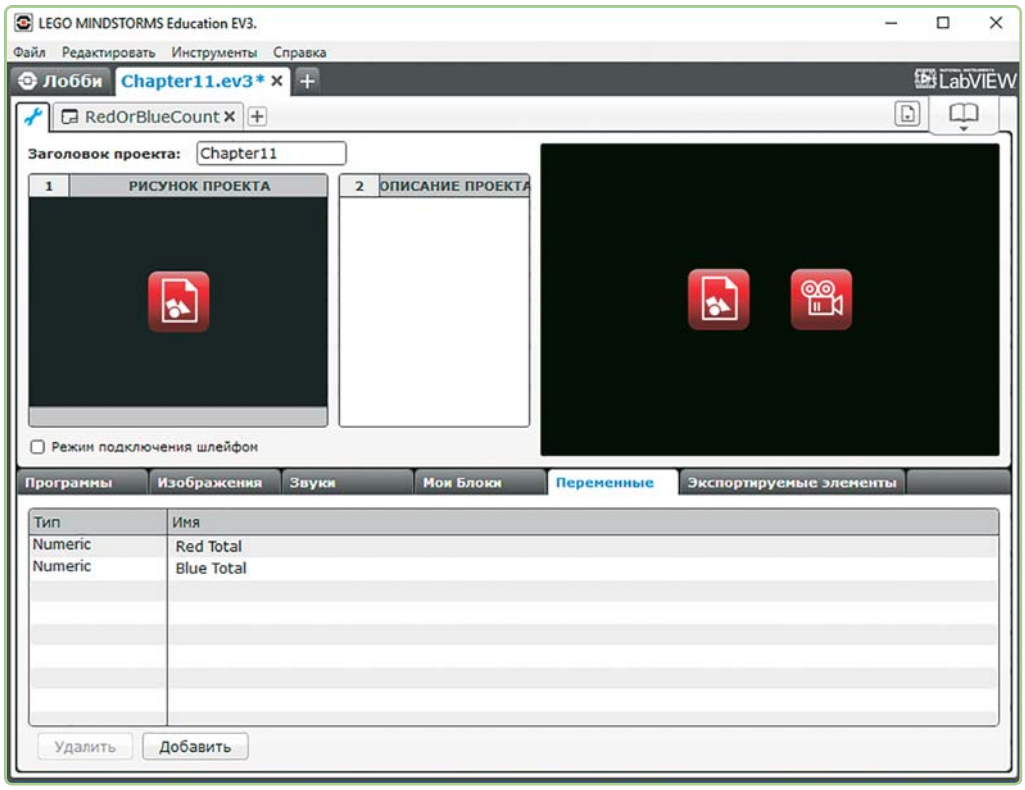


*Рис. 11.17. Вкладка **Переменные** (Variables) на странице **Свойства проекта** (Project Properties)*

На вкладке **Переменные** (Variables) отображаются имя и тип всех переменных, используемых в текущем проекте. Для добавления переменной в проект просто щелкни по кнопке **Добавить** (Add). В появившееся диалоговое окно (рис. 11.18) можно ввести имя и тип новой переменной. Ты можешь удалить переменную, если решишь не использовать ее в своих программах. Для удаления переменной выбери ее в списке и щелкни по кнопке **Удалить** (Delete) в нижней части окна. Если ты удалишь переменную, которая по-прежнему используется программой, программа продолжит работу, однако переменная не будет отображаться на странице **Свойства проекта** (Project Properties) или в списке, появляющемся при щелчке в поле для имени переменной блока **Переменная** (Variable).

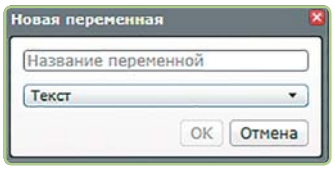


Рис. 11.18. Добавление новой переменной

Блок «Сравнение»

Для создания следующей программы ты будешь использовать блок **Сравнение** (Compare), изображенный на рис. 11.19. Давай посмотрим, как работает этот блок. Блок **Сравнение** (Compare) находится на вкладке, содержащей блоки операций с данными. Ты можешь предоставить два входных значения с помощью шины данных или вручную настроить один или оба параметра. Блок сравнивает эти два значения в соответствии с выбранным режимом, как показано на рис. 11.20, и результат передается на вывод блока. Например, для блока, показанного на рис. 11.19, выбран режим **Равно** (Equal To), поэтому он проверяет, равны ли два значения, и отправляется результат на вывод.

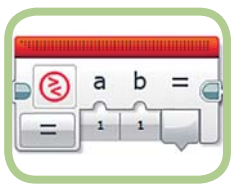


*Рис. 11.19. Блок **Сравнение** (Compare)*

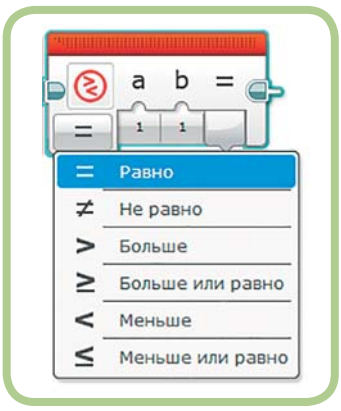


*Рис. 11.20. Режимы блока **Сравнение** (Compare)*

Результат из блока **Сравнение** (Compare) всегда представляет собой логическое значение («Истина» или «Ложь»). Использование блока **Сравнение** (Compare) напоминает постановку вопроса: «Превышает ли показание ультразвукового датчика значение 20?» Ответом будет логическое значение.

Блок **Сравнение** (Compare) полезен при принятии решений, поскольку полученное логическое значение можно применять для управления блоками **Переключатель** (Switch) и **Цикл** (Loop), и это обеспечивает большую гибкость по сравнению с использованием одного только блока **Переключатель** (Switch) или **Цикл** (Loop). Например, ты можешь сравнить показания двух датчиков вращения мотора или использовать блок **Математика** (Math) для изменения показания датчика, прежде чем сравнивать его с целевым значением.

Как правило, блок **Сравнение** (Compare) используется так, как показано на рис. 11.21. С помощью шин данных в блок передаются два числа, которые требуется сравнить. В данном примере в качестве режима блока выбран вариант **Меньше** (Less Than).

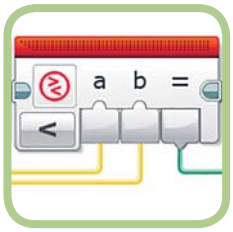


*Рис. 11.21. Блок **Сравнение** (Compare) с шинами данных*

В процессе работы этот блок сравнивает две входные величины, полученное логическое значение помещается в выходную шину данных. Например, если число A равно 7, а число B равно 12, результатом является **Истина** (True), поскольку 7 меньше 12. Однако, если число A равно 25, а число B — 8, результатом будет **Ложь** (False), поскольку 25 не меньше 8.

Программа LightPointer

Программа *LightPointer* — хороший пример для демонстрации применения переменных для запоминания значений, которые будут использоваться далее в программе. Данная программа использует датчик цвета, чтобы направить робота TriBot в сторону источника света, вращая его на месте и запоминая положение, в котором датчик зафиксировал самый яркий свет. Разработанные здесь код и идеи можно применить в качестве фрагмента более крупной программы, например программы для следования за светом фонарика или для нахождения самого длинного свободного пути через область с препятствиями.

В зависимости от целей программы датчик цвета можно прикрепить к передней (рис. 11.22) или к боковой (рис. 11.23) части робота. Разумеется, ты можешь скорректировать

местоположение датчика в зависимости от высоты, на которой расположен источник света.

Рис. 11.22. Датчик цвета, прикрепленный к передней части робота TriBot

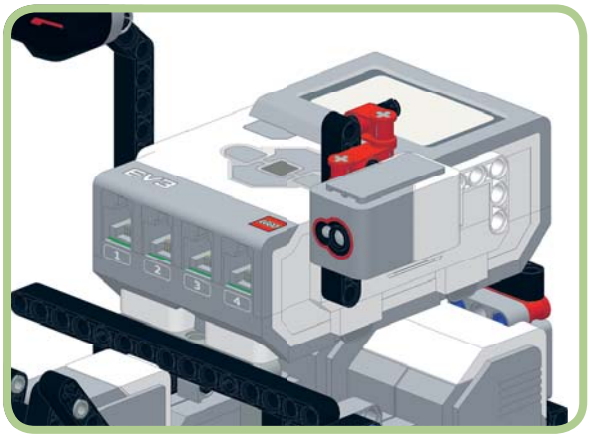


Рис. 11.23. Датчик цвета, прикрепленный к боковой части робота TriBot

В этой программе предусмотрены два разных этапа: поиск источника света и после этого отправка робота в его сторону. На первом этапе робот медленно вращается на месте, а датчик непрерывно измеряет яркость внешнего освещения. Каждое его показание сравнивается с максимальным показанием, зафиксированным до сих пор, и при обнаружении большего значения соответствующее положение робота запоминается.

На рис. 11.24 показано, как изменяется показание датчика по мере вращения робота TriBot. Сначала робот отвернут от света, поэтому показание датчика небольшое (10). Когда робот поворачивается в сторону фонарика, показание датчика увеличивается до 40, как показано на втором изображении. Когда робот TriBot направлен прямо на фонарик, фиксируется максимальное показание (70 в данном примере). Показание датчика уменьшается, когда робот отворачивается от фонарика, как показано на последнем изображении.

На втором этапе программа возвращает робота в положение, соответствующее максимальному показанию, в результате чего робот должен быть направлен точно на источник света.

Определение переменных

Для этой программы требуются две переменные для хранения двух разных числовых значений. В первой переменной, Max Reading, содержится максимальное показание датчика, зафиксированное до сих пор. Во второй переменной, Position, хранится положение робота, при котором было зафиксировано максимальное значение. Создай эти две числовые переменные, используя вкладку **Переменные** (Variables) на странице **Свойства проекта** (Project Properties). Не забудь указать тип **Числовое значение** (Numeric) в окне для создания новой переменной. После создания двух этих переменных вкладка **Переменные** (Variables) должна выглядеть так, как показано на рис. 11.25.

Поиск источника света

Первым действием при поиске источника света является вращение робота на месте. Для этого мы будем использовать блок **Рулевое управление** (Move Steering) со значением параметра **Рулевое управление** (Steering), равным –100. Также потребуется отследить положение робота во время его вращения, для этого мы будем использовать значение датчика вращения мотора C, которое увеличивается по мере вращения робота TriBot.

Рис. 11.24. Показания датчика в четырех положениях робота

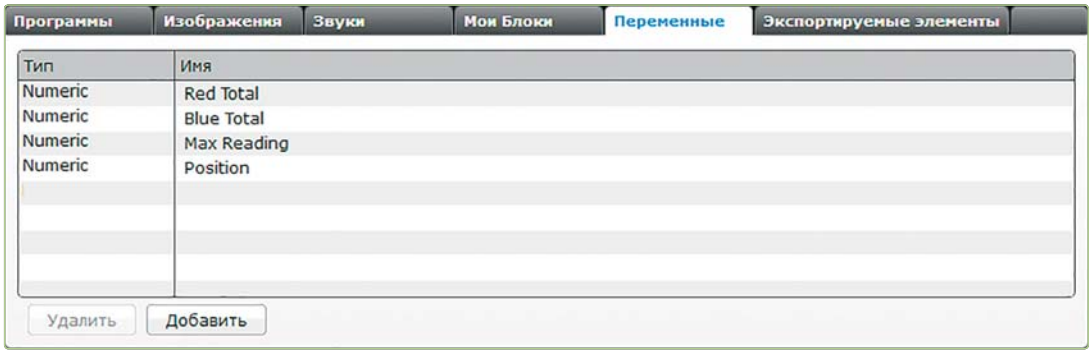


Рис. 11.25. Вкладка ***Переменные*** *(Variables) после создания новых переменных*

Рис. 11.26. Изменение значений переменных Max Reading и Position по мере вращения робота

Робот должен совершить полный оборот, чтобы найти источник света, где бы тот ни находился. Небольшой эксперимент показал, что значение параметра **Продолжительность** (Duration), равное 900°, обеспечивает полный оборот робота. Это значение не обязательно должно быть точным; программа будет работать, даже если робот повернет на чуть большее количество градусов.

ПРИМЕЧАНИЕ **При использовании шин из образовательной версии конструктора задай значение 700° для совершения полного оборота.**

По мере вращения робота программа сравнивает показание датчика цвета с максимальным показанием, зафиксированным до сих пор. При обнаружении более высокого показания в переменной Max Reading сохраняется новое (более высокое) значение, а в переменной Position — положение мотора C. На рис. 11.26 показаны значения переменных, соответствующие положениям робота, показанным на рис. 11.24. Обрати внимание, на четвертом изображении показаны необновленные значения переменных Max Reading и Position, поскольку в этом положении показание датчика не превышает зафиксированного ранее.

В листинге 11.2 приведен псевдокод для этой части программы. Запись немного сокращена, поскольку у тебя уже есть некоторый опыт работы с переменными:

```
Max Reading = Color Sensor reading
```

Вышеприведенная строка — это более краткий вариант выражения: «Возьми показание датчика цвета и сохрани его в переменной Max Reading».

Эта запись соответствует способу обновления значения переменной, используемому во многих других языках программирования, в которых знак равенства означает: «Сохрани значение справа в переменной слева».

Листинг 11.2. Поиск источника света

```
start the robot spinning slowly
begin loop
   if Color Sensor reading > Max Reading then
      Max Reading = Color Sensor reading
      Position = B motor Rotation Sensor reading
   end if
loop until B motor Rotation Sensor > 900
```

Создание программы LightPointer

Первые три блока в программе *LightPointer* инициализируют переменные Max Reading и Position, а также сбрасывают показание датчика вращения мотора C (рис. 11.27).

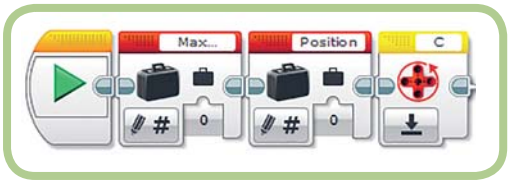


Рис. 11.27. Инициализация переменных и сброс показания датчика вращения мотора

После инициализации переменной можно приступать к написанию кода для поиска источника света. Используй в качестве руководства псевдокод, разработанный в предыдущем разделе. Заставь робота TriBot поворачиваться на месте с помощью блока **Рулевое управление** (Move Steering), как показано на рис. 11.28. Для параметра **Мощность** (Power) задай значение 20, чтобы робот вращался медленно и не пропустил источник света.

Рис. 11.28. Начало вращения робота

В следующей части программы (рис. 11.29), с помощью блока **Цикл** (Loop) и режима **Вращение мотора** (Motor Rotation) ⇒ **Сравнение** (Compare) ⇒**Градусы** (Degrees) робот вращается до тех пор, пока показание датчика вращения мотора не превысит 900°. При каждом выполнении цикла показание датчика цвета сравнивается со значением переменной Max Reading. Если показание датчика превышает текущее значение Max Reading, выполняется код в блоке **Переключатель** (Switch). С помощью этого кода обновляется переменная Max Reading и в переменной Position сохраняется текущее положение мотора, при этом используется показание датчика вращения. Другая вкладка блока **Переключатель** (Switch) блоков не содержит.

Теперь рассмотрим блоки, используемые в этом разделе программы.

1. Блок **Цикл** (Loop) функционирует до тех пор, пока показание датчика вращения мотора C не превысит 90°. Благодаря этому цикл повторяется до тех пор, пока робот не совершит полный оборот на месте.

2. В блоке **Датчик цвета** (Color Sensor) используется режим **Измерение** (Measure) ⇒ **Яркость внешнего освещения** (Ambient Light Intensity) для измерения яркости света перед датчиком. С помощью шин данных это значение передается другим блокам.

3. В блоке **Переменная** (Variable) считывается и отправляется в блок **Сравнение** (Compare) с помощью шины данных текущее значение переменной Max Reading.

ИНИЦИАЛИЗАЦИЯ ПЕРЕМЕННЫХ

Прежде чем писать код для первой части этой программы, подумай о том, какие значения должны иметь переменные в начале программы. Несмотря на то что код, используемый для инициализации значений, обычно располагается в начале программы, тебе часто может потребоваться сначала разработать всю программу (или по крайней мере ее основные фрагменты), чтобы решить, как эти значения следует инициализировать.

Переменная Max Reading предназначена для хранения максимального показания датчика цвета, которое будет находиться в диапазоне от 0 до 100. Значение 0 (минимально возможное показание) в качестве значения переменной Max Reading в начале программы — гарантия того, что показание датчика и положение робота будут зафиксированы, как только показание датчика превысит 0.

Несмотря на то что значение переменной Position будет задано, когда датчик впервые обнаружит свет, установка ее начального значения является хорошей идеей. Инициализация всех переменных — это хорошая практика программирования, которая позволяет избежать некоторых ошибок, которые трудно будет обнаружить в дальнейшем. В коде, представленном в следующем разделе, значение переменной Position устанавливается на 0 в начале программы.

В дополнение к двум переменным в листинге 11.2 используется датчик вращения мотора C. Для того чтобы убедиться в корректной работе цикла, следует сбросить показание датчика вращения в начале программы. Показание датчика вращения по умолчанию при запуске программы устанавливается на 0. Однако, если ты хочешь повторно использовать этот код в другой программе, тебе необходимо сбросить показание датчика вращения до того, как с помощью блока **Рулевое управление** (Move Steering) робот начнет вращаться. Применять программу повторно будет намного проще, если ты явно инициализируешь все переменные вместо того, чтобы полагаться на их автоматическую инициализацию при запуске программы.

4. В блоке **Сравнение** (Compare) сравниваются показание из блока **Датчик цвета** (Color Sensor) и значение переменной Max Reading. Если значение Max Reading меньше нового показания датчика, в блок **Переключатель** (Switch) передается значение **Истина** (True); в противном случае передается значение **Ложь** (False).

5. Если результатом блока **Сравнение** (Compare) является **Истина** (True), с помощью блока **Переключатель** (Switch) запускаются три блока, предусмотренные для случая «Истина», показанные на рис. 11.29. Для случая «Ложь» никаких блоков не предусмотрено.

6. С помощью первого блока **Переменная** (Variable) в блоке **Переключатель** (Switch) сохраняется последнее показание датчика цвета в переменной Max Reading. При следующем выполнении цикла это значение будет использовано в блоке **Сравнение** (Compare).

7. Благодаря блоку **Вращение мотора** (Motor Rotation) считывается текущее положение мотора C.

8. Вторым блоком **Переменная** (Variable) сохраняется положение мотора C в переменной Position.

Работа блока **Цикл** (Loop) продолжается до тех пор, пока робот TriBot не совершит полный оборот вокруг своей оси. После завершения цикла переменная Position должна содержать положение мотора C, при котором был зафиксирован самый яркий свет. Во второй части программы используется это значение для того, чтобы заставить робота вращаться в противоположном направлении для его возвращения в это положение.

С помощью блока **Рулевое управление** (Move Steering) с параметром **Рулевое управление** (Steering), установленным на 100, робот начнет вращаться в обратную сторону, что приведет к уменьшению показания датчика вращения мотора C. Робот должен продолжать вращаться, пока показание датчика вращения превышает значение переменной Position, чтобы вернуться в то положение, при котором был зафиксирован самый яркий свет. В листинге 11.3 приведен псевдокод для данного раздела программы.

Листинг 11.3. Возвращение робота в то положение, в котором он направлен прямо на источник света

```
start the robot spinning slowly in the opposite
   direction from the first move
read the Position variable
wait until the Rotation Sensor reading reaches the
   Position value
stop the motors
```

На рис. 11.30 показан данный раздел программы. Обрати внимание на то, что с помощью блока **Ожидание** (Wait) считывается показание датчика вращения мотора и сравнивается его со значением, сохраненным в переменной Position. В качестве типа сравнения выбран вариант **Меньше** (Less Than), в результате чего блок находится в режиме ожидания до тех пор, пока показание датчика вращения мотора не станет меньше значения, сохраненного в переменной Position. После выполнения блока **Ожидание** (Wait) от блока **Рулевое управление** (Move Steering) поступит команда, и моторы остановятся.

Теперь попробуй запустить готовую программу. Робот TriBot должен медленно совершить полный оборот, а затем повернуться в обратном направлении и остановиться в тот момент, когда датчик цвета будет фиксировать самый яркий свет. Попробуй проверить это поведение в темной комнате, посветив на робота фонариком.

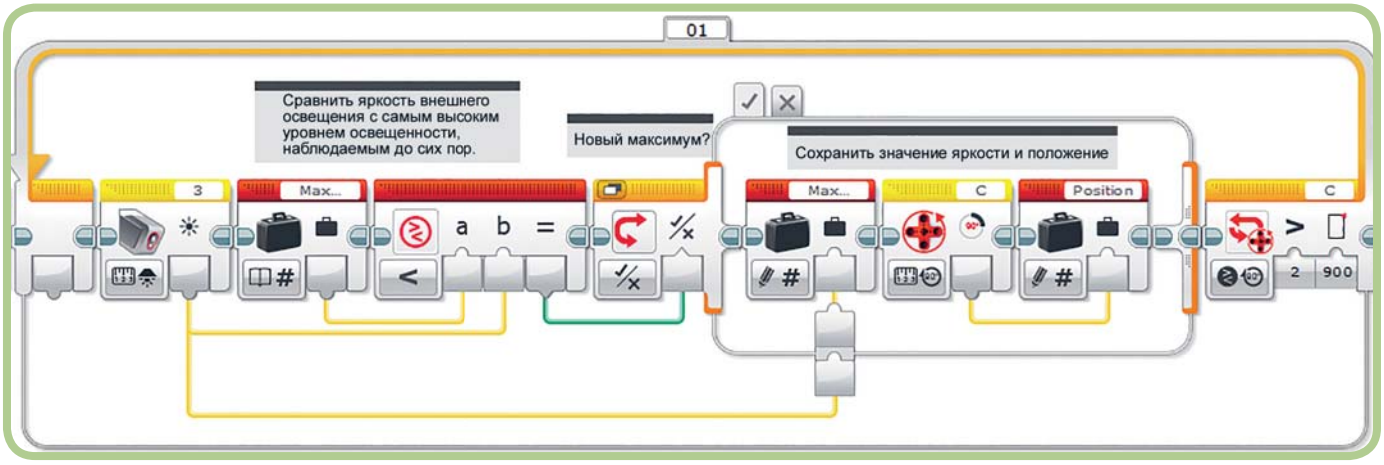


Рис. 11.29. Поиск самого яркого источника света

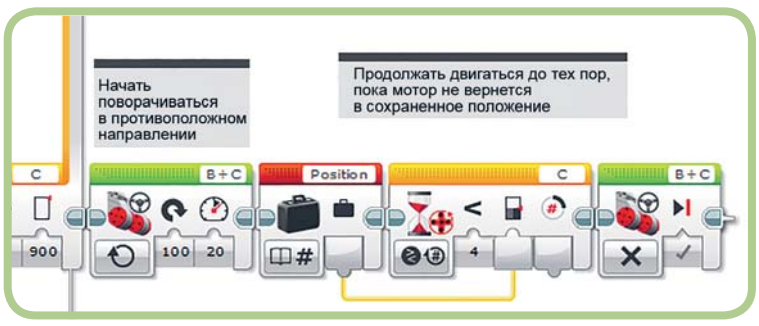


Рис. 11.30. Возврат в сохраненное положение

Блок «Константа»

Часто в программе применяются несколько блоков с одинаковыми параметрами. Например, программа *WallFollower* содержит семь блоков **Рулевое управление** (Move Steering), в которых используется одно и то же значение параметра **Мощность** (Power). Если ты решишь изменить это значение с 35 на 45, тебе придется делать это во всех семи блоках. Это может быть проблемой, если необходимо протестировать несколько разных параметров мощности. Не исключены ошибки, например, ты можешь случайно забыть изменить параметр в одном из блоков.

С помощью блока **Константа** (Constant) можно сохранить значение, которое затем можно использовать для настройки параметров в блоках твоей программы. На рис. 11.31 показан блок **Константа** (Constant) с раскрывающимся списком выбора режима. Этот блок напоминает блок **Переменная** (Variable) за исключением того, что он предусматривает только режимы чтения. Это связано с тем, что во время выполнения программы ты не можешь записать новое значение в блок **Константа** (Constant); ты можешь только заранее задать значение в поле в верхнем правом углу блока.

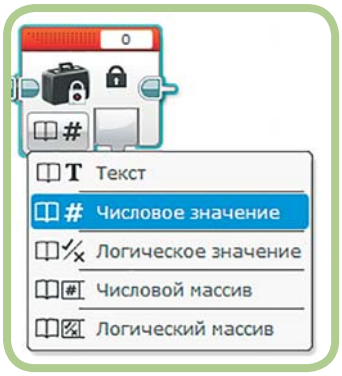


*Рис. 11.31. Блок **Константа** (Constant)*

Например, на рис. 11.32 показана измененная версия программы *AroundTheBlock* из гл. 4. Здесь мы используем блок **Константа** (Constant) для управления обоими блоками **Рулевое управление** (Move Steering) вместе того, чтобы задавать значение мощности для каждого отдельного блока **Рулевое управление** (Move Steering). Таким образом, для тестирования разных параметров мощности необходимо изменить только блок **Константа** (Constant) вместо того, чтобы вносить изменения в каждый блок **Рулевое управление** (Move Steering). Это также предотвращает возможность того, что ты изменишь параметр в одном блоке и не изменишь его в другом. Это может быть особенно полезно при работе с такими крупными программами, как *WallFollower*, в которых ты можешь использовать один блок **Константа** (Constant) для настройки параметра **Мощность** (Power) сразу в семи блоках.

Дальнейшее исследование

Для того чтобы попрактиковаться в использовании переменных и констант, выполни следующие действия:

1. Используй блок **Константа** (Constant) для настройки параметра **Мощность** (Power) во всех блоках **Рулевое управление** (Move Steering) в программе *WallFollower*. Затем вместо блока **Константа** (Constant) используй переменную. Тебе понадобится один блок **Переменная** (Variable) в начале программы, и один или несколько блоков **Переменная** (Variable) для чтения значения и его передачи одному или нескольким блокам **Рулевое управление** (Move Steering). При использовании блока **Переменная** (Variable) можно избежать путаницы, связанной с длинными шинами данных. Какое решение кажется тебе более предпочтительным: в котором используется один блок **Константа** (Constant) с более длинными шинами данных или в котором используется несколько блоков **Переменная** (Variable) и более короткие шины данных? В данном случае не может быть правильного или неправильного ответа, это, скорее, вопрос стиля и предпочтений.

2. Создай новую программу под названием *ObstaclePointer* на основе программы *ObstacleAvoider*. Измени программу так, чтобы робот TriBot указывал в направлении ближайшего препятствия. Тебе нужно, чтобы программа

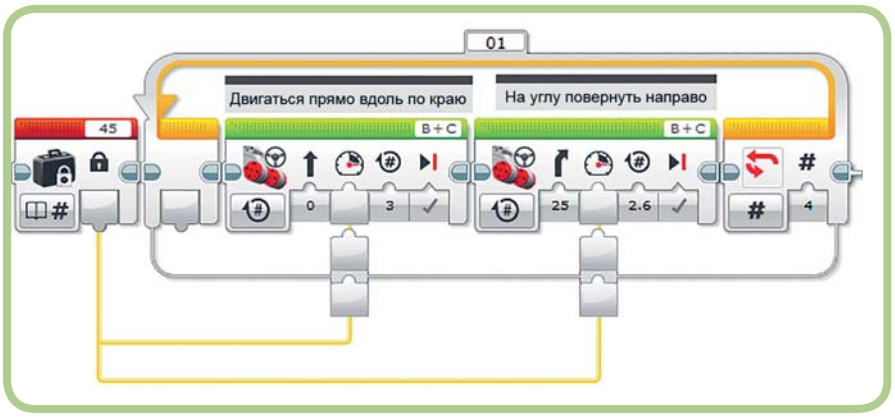


*Рис. 11.32. Использование блока **Константа** (Constant) для задания уровня мощности*

определяла направление, при котором показание датчика является минимальным, а не максимальным. Протестируй программу на объектах с разной формой, цветом и текстурой. Ты заметишь, что показание датчика зависит не только от расстояния между роботом и объектом. Подсказка: вместо переменной Max Reading тебе понадобится переменная Min Reading, в качестве начального значения которой задано 100, а не 0.

3. Помести программу *ObstacleAvoider* в цикл и заставь робота передвигаться вперед на небольшое расстояние в конце каждого цикла. Другими словами, робот должен проверить наличие свободного от препятствий пути, переместиться немного вперед (скажем, на пять оборотов мотора), снова проверить наличие свободного от препятствий пути, снова переместиться вперед и т. д. Благодаря повторению этой последовательности действий робот должен начать передвигаться по области, заполненной препятствиями.

ПРАКТИКУМ 11.1

Создай новую программу под названием *ObstacleAvoider* на основе программы *LightPointer*. Используй инфракрасный или ультразвуковой датчик, чтобы развернуть робота в направлении, в котором датчик фиксирует наибольшее значение (это должно привести к тому, что он будет указывать в направлении наименьшего количества препятствий).

ПРАКТИКУМ 11.2

Если у тебя есть гироскопический датчик, используй его вместо датчика вращения мотора C, чтобы отметить положение, соответствующее максимальному показанию датчика, и заставить робота TriBot повернуться датчиком к свету. Приводит ли это к лучшему результату по сравнению с применением датчика вращения мотора? Попробуй увеличить скорость в блоках **Рулевое управление (Move Steering) и посмотри, как это влияет на эффективность каждого из подходов.**

Заключение

Переменные позволяют хранить и обновлять данные, используемые в программах. Блок **Переменная** (Variable) применяется для создания и получения доступа к переменным в твоей программе. Переменные обеспечивают программе большую гибкость и необходимы для решения многих задач. На примере представленных в этой главе программ ты изучил несколько способов использования переменных. В следующих главах описаны другие способы их применения.

Константы, создаваемые с помощью блока **Константа** (Constant), полезны, когда необходимо применить неизменяющееся значение для управления несколькими блоками. Благодаря этому можно легко консолидировать параметры нескольких блоков в одном месте, а значит, появляется возможность влиять на множество блоков путем изменения только одного значения.

В этой главе также был представлен блок **Сравнение** (Compare), с помощью которого программа может принимать более сложные решения, чем было возможно с помощью блока **Переключатель** (Switch). Из гл. 13 ты узнаешь больше о блоках **Математика** (Math) и **Логические операции** (Logic Operations), используемых для принятия еще более сложных решений.

Прежде чем разбираться с другими блоками, имеющими отношение к математике, тебе нужно познакомиться с контейнерами «Мой блок», о которых речь пойдет в гл. 12. Мои блоки — это особые блоки, создаваемые из частей программы и позволяющие сделать программы более компактными, удобными для повторного использования и менее подверженными ошибкам.

12

Мои блоки

Создание контейнера «Мой блок» — простой способ группировки и повторного использования нескольких блоков. *Мой блок* — это контейнер с блоками, который затем можно использовать в программе как единый блок программирования EV3. Так, благодаря группировке последовательности связанных между собой блоков в один блок программы становятся более компактными и удобными для восприятия.

Из этой главы ты узнаешь, как создавать контейнеры «Мой блок» и применять их в своих программах. Мы рассмотрим процесс создания нескольких таких контейнеров, начиная с самого простого, который просто издает звуковой сигнал, и заканчивая более сложным, способным отображать на экране модуля число с меткой. По мере чтения главы ты узнаешь все необходимое для создания своих собственных блоков.

Создание контейнера «Мой блок»

Начнем с того, что создадим простой контейнер «Мой блок» на основе части программы *DoorChime* из гл. 5 (рис. 12.1). С помощью блоков **Звук** (Sound) в этой программе воспроизводятся два тона звукового сигнала. Ты превратишь эти блоки **Звук** (Sound) в контейнер «Мой блок», что сократит программу и позволит создать блок **Chime**, который можно будет использовать в других программах.

1. Скопируй программу *DoorChime* из проекта *Chapter5* в новый проект *Chapter12*.

2. Выдели два блока **Звук** (Sound), нарисовав вокруг них прямоугольную рамку выделения (рис. 12.2) или щелкнув по ним, удерживая нажатой клавишу **Shift**.

Выбери команду меню **Инструменты** (Tools) ⇒ **Конструктор мой блок** (Tools) (My Block Builder), чтобы создать контейнер «Мой блок» из выделенных блоков. Появится окно **Конструктор мой блок** (My Block Builder), как показано на рис. 12.3.

В окне **Конструктор мой блок** (My Block Builder) введи имя и описание нового блока. В нижней части окна находится раздел, в котором ты можешь выбрать значок для этого блока.

3. Введи текст Chime в поле **Имя** (Name).

4. Введи текст **Воспроизвести звуковой сигнал**, используя блоки **Звук (Sound)** в поле **Описание** (Description).

5. Выбери значок блока **Звук** (Sound) ( ) в нижней части окна.

6. Щелкни по кнопке **Завершить** (Finish).

После щелчка по кнопке **Завершить** (Finish) будет создан блок Chime, который заменит блоки **Звук** (Sound) в программе *DoorChime* (как показано на рис. 12.4). После перемещения блоков ближе друг к другу (для этого нужно щелкнуть по правому выходному разъему блока), программа будет выглядеть так, как показано на рис. 12.5.

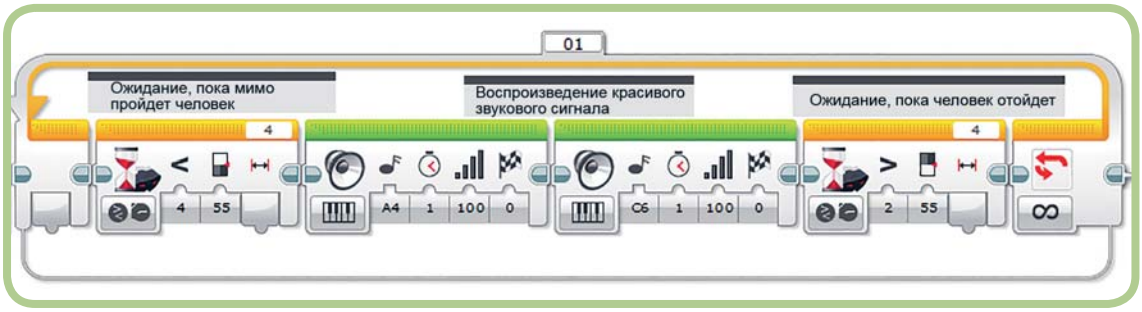


Рис. 12.1. Программа DoorChime

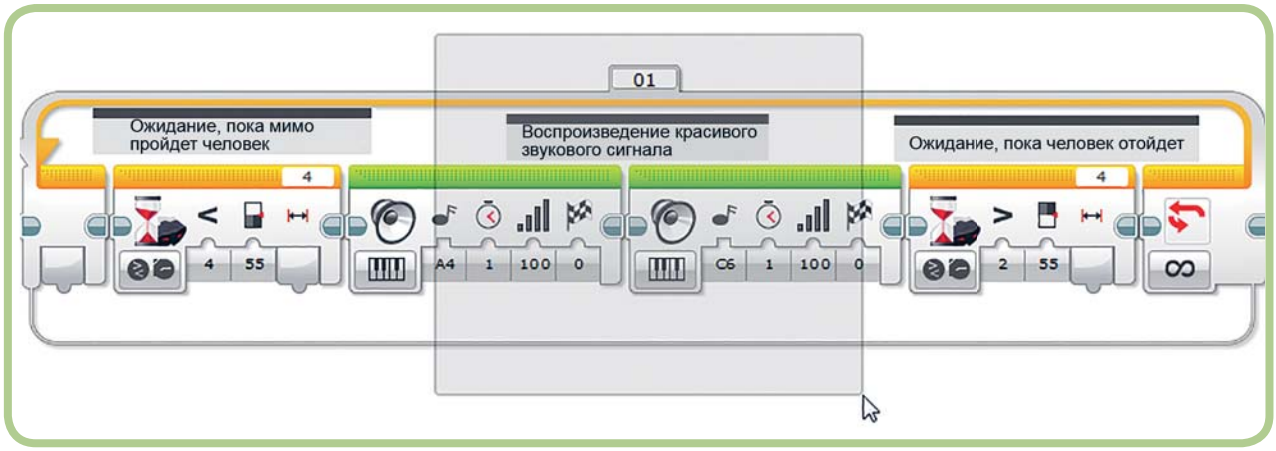


Рис. 12.2. Выделение обоих блоков ***Звук*** *(Sound)*

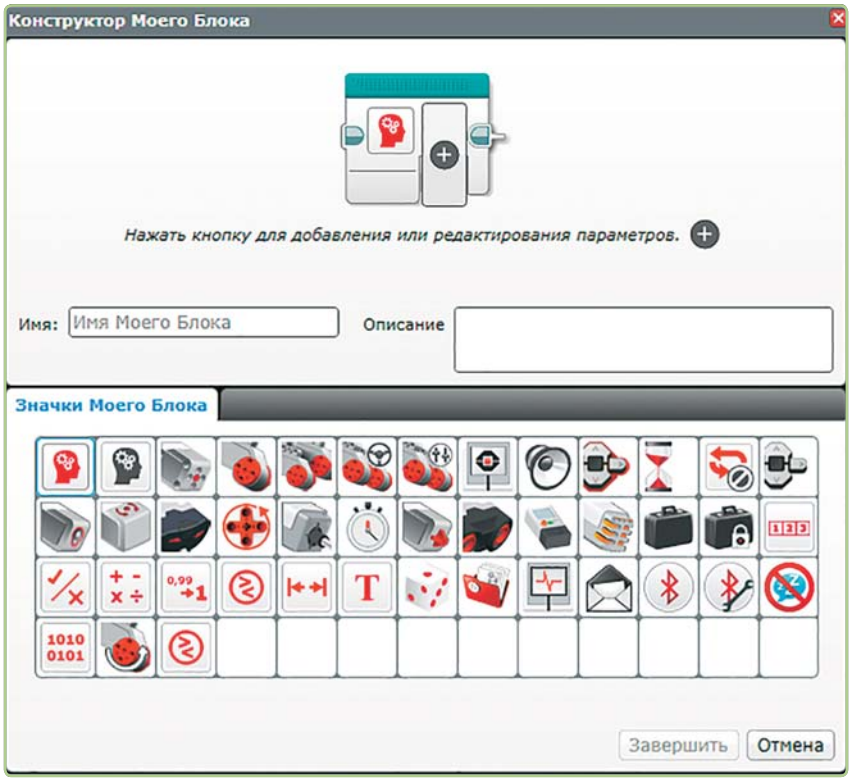


Рис. 12.3. Окно ***Конструктор мой блок*** *(My Block Builder)*

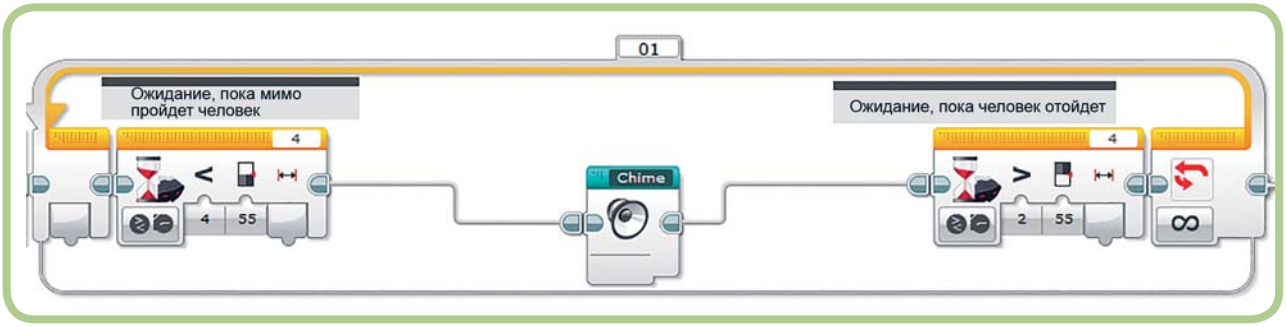


Рис. 12.4. Программа DoorChime после создания блока Chime

Рис. 12.5. Программа DoorChime после перемещения блоков ближе друг к другу

Теперь программа стала более компактной и простой. Кроме того, гораздо легче понять назначение одного блока, чем разобраться в способе работы двух блоков **Звук** (Sound). Изменение способа организации блоков не повлияло на работу программы; после запуска она должна работать так же, как и до создания блока Chime.

Палитра «Мои блоки»

После создания контейнера «Мой блок» ты можешь применять его в любой программе своего проекта, как и любой другой блок. Все контейнеры «Мой блок», имеющие отношение к твоему проекту, отображаются на голубой вкладке **Мои блоки** (My Blocks), расположенной в правом конце палитры программирования. Каждый контейнер «Мой блок» отмечен значком, который был выбран при его создании. Чтобы увидеть имя контейнера, наведи указатель мыши на блок, как показано на рис. 12.6.

*Рис. 12.6. Блок Chime на палитре **Мои блоки** (My Blocks)*

Твои контейнеры «Мой блок» также отображаются на вкладке **Мои блоки** (My Blocks) страницы **Свойства проекта** (Project Properties) (рис. 12.7). Эту вкладку можно применять для удаления неиспользуемого контейнера «Мой блок», его копирования и вставки из одного проекта в другой, а также для его экспортирования или импортирования в файл или из файла на своем компьютере.

Изменение контейнера «Мой блок»

Для того чтобы изменить контейнер «Мой блок», дважды щелкни по его экземпляру в программе или по имени на вкладке **Мои блоки** (My Blocks) страницы **Свойства проекта** (Project Properties). Например, чтобы отредактировать блок Chime, выполни следующие действия:

1. Открой программу *DoorChime*, если она еще не открыта.
2. Дважды щелкни по блоку Chime. Отобразятся два блока **Звук** (Sound), которые ты сможешь отредактировать.
3. Добавь еще два блока **Звук** (Sound). Активируй режим **Воспроизвести ноту** (Play Note) и выбери ноты для воспроизведения (рис. 12.8).

Теперь после загрузки и запуска программа *DoorChime* должна воспроизвести все четыре блока **Звук** (Sound). Другие программы, в которых задействован блок Chime, также будут использовать эти новые блоки **Звук** (Sound). Изменение контейнера «Мой блок» повлияет на все программы, в которых он используется. Исправление ошибки в контейнере «Мой блок» автоматически распространяется на все содержащие его программы. При этом следует убедиться в том, что изменения, которые ты вносишь для улучшения одной программы, не приводят к ухудшению других программ.

ПРАКТИКУМ 12.1

Программа *WallFollower* состоит из трех разделов: один удерживает робота TriBot у стены, другой заставляет робота въезжать в проход, а третий заставляет его поворачивать влево, если он окажется в углу. Преврати каждый из трех разделов в контейнер «Мой блок», чтобы сделать программу более компактной и понятной.

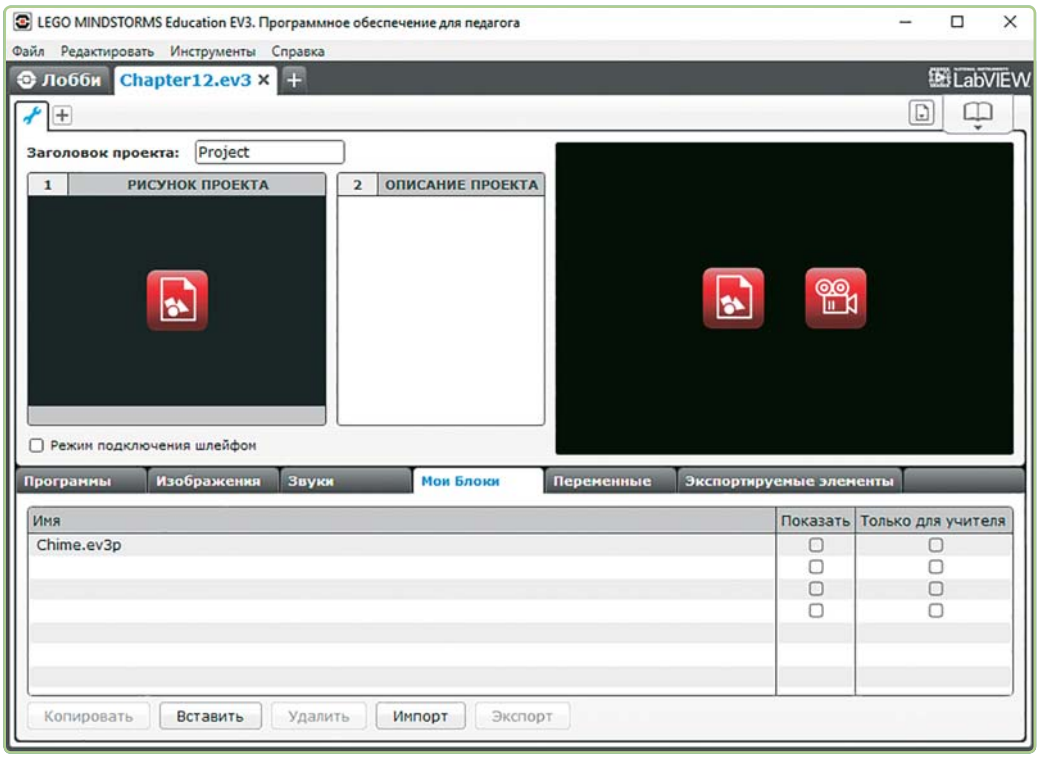


Рис. 12.7. Вкладка ***Мои блоки*** *(My Blocks) на странице* ***Свойства проекта*** *(Project Properties)*

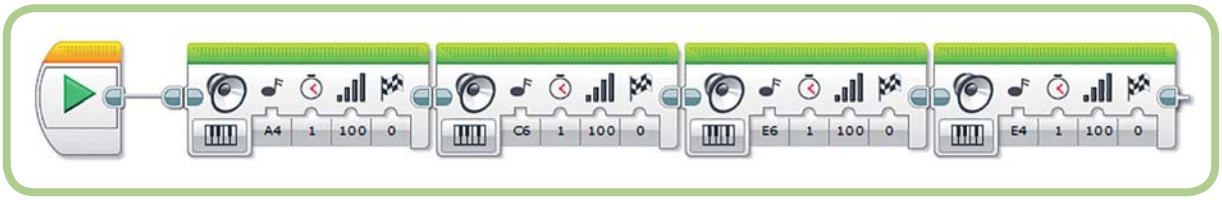


Рис. 12.8. *Редактирование контейнера Chime*

Контейнер LogicToText

В блоке **Chime** отсутствуют параметры, т. е. в нем всегда происходит один и тот же процесс. Для большей части блоков EV3 требуются параметры, определяющие их поведение. Ты уже знаешь, как с помощью шин данных происходит обмен выходными данными между блоками. В этом разделе ты создашь контейнер **LogicToText** с параметрами **Ввод** (Input) и **Вывод** (Output). Для начала давай создадим контейнер «Мой блок» на основе блока **Переключатель** (Switch), используемого в программе *LogicToText* из гл. 9 (рис. 12.9). Этот блок принимает в качестве входных данных логическое значение, а в качестве выходных данных выдает текстовое значение. Затем этот блок можно использовать в любой программе для преобразования логического значения в текст, который можно отобразить на экране модуля EV3.

Выполни следующие действия для создания контейнера **LogicToText**:

1. Открой проект *Chapter9* и скопируй программу *LogicToText* в проект *Chapter12*.

Хочется назвать новый контейнер **LogicToText**, однако имя контейнера не может совпадать с именем программы.

2. Измени имя программы на *LogicToTextBuilder*. Для этого дважды щелкни по вкладке с именем программы и введи новое имя.

3. Выдели блок **Переключатель** (Switch).

4. Выбери команду меню **Инструменты** (Tools) ⇒ **Конструктор Мой блок** (My Block Builder).

В окне **Конструктор мой блок** (My Block Builder) должен отображаться контейнер «Мой блок» с двумя параметрами,

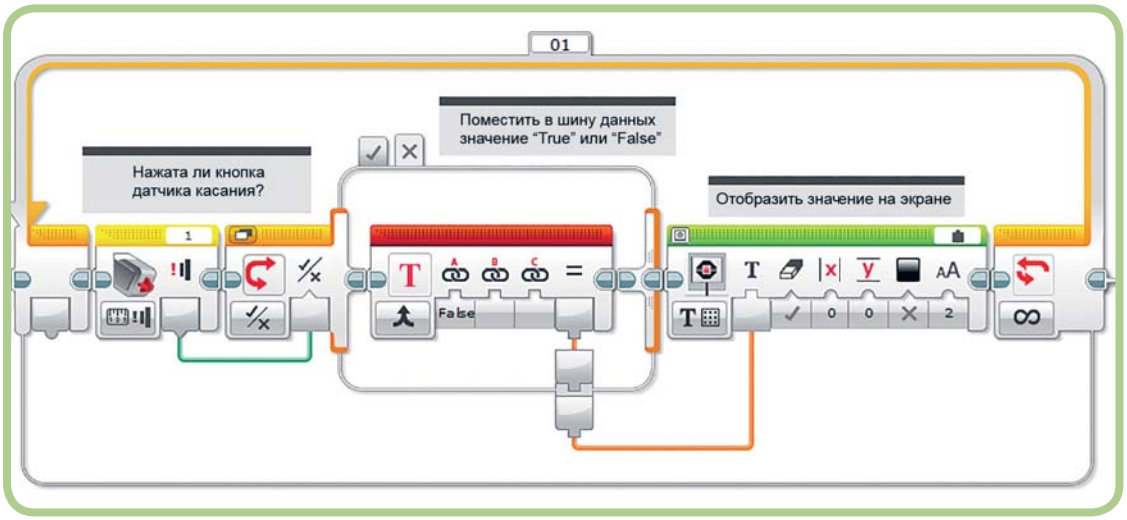


Рис. 12.9. Программа LogicToText

выделенными на рис. 12.10. Обрати внимание на то, что в блоке **Переключатель** (Switch) используются две шины данных: для принятия логического значения в качестве входных данных и для передачи результирующего текстового значения следующим блокам. При создании контейнера «Мой блок» шины данных, которые подключаются к выбранным блокам, превращаются в параметры **Ввод** (Input) и **Вывод** (Output) нового блока. В данном случае шина данных для логического значения, передающегося из блока **Датчик касания** (Touch Sensor), превращается в параметр **Ввод** (Input), а шина данных для текстового значения, подключающаяся к блоку **Экран** (Display), — в параметр **Вывод** (Output).

В нижней части окна ты увидишь две новые вкладки: **Настройка параметров** (Parameter Setup) и **Значки параметров** (Parameter Icons), которые можно использовать для изменения внешнего вида параметров блока. Чуть позже мы применим обе эти вкладки.

5. В окне **Конструктор мой блок** (My Block Builder) введи значение **LogicToText** в поле **Имя** (Name).

6. Введи текст **Преобразование логического значения в текстовое значение «Истина» или «Ложь»** в поле **Описание** (Description).

7. Поскольку данный контейнер «Мой блок» имеет дело с логическими значениями, выбери соответствующий значок ( ).

8. Убедись в том, что выбран первый параметр, а затем щелкни по вкладке **Настройка параметров** (Parameter Setup). Окно должно выглядеть так, как показано на рис. 12.11.

На этой вкладке можно выбрать имя и тип параметра (ввод или вывод), тип данных и значение по умолчанию. Имя отображается при наведении на параметр указателя мыши, как и в случае с другими блоками EV3. Выбранное тобой значение по умолчанию будет использоваться для этого параметра, если ты не подключишь к нему шину данных.

Для этого параметра уже заданы тип параметра и тип данных, их изменить нельзя, потому что данный параметр был создан автоматически на основании содержимого шины данных. Ты также можешь добавить параметры с помощью окна **Конструктор Мой блок** (My Block Builder); при таком способе добавления параметров можно поменять эти настройки.

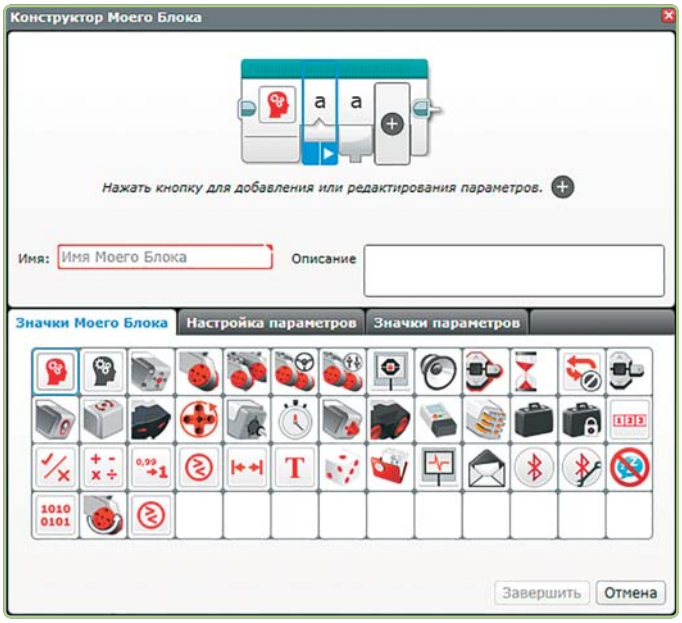


*Рис. 12.10. Окно **Конструктор мой блок** с выделенными вводом и выводом блока*

На этой вкладке поле **Имя** (Name) содержит значение State, а в качестве значения по умолчанию выбран вариант **Ложь** (False). Ты можешь оставить значение по умолчанию (для этого блока оно не так важно), однако данному параметру лучше дать другое имя:

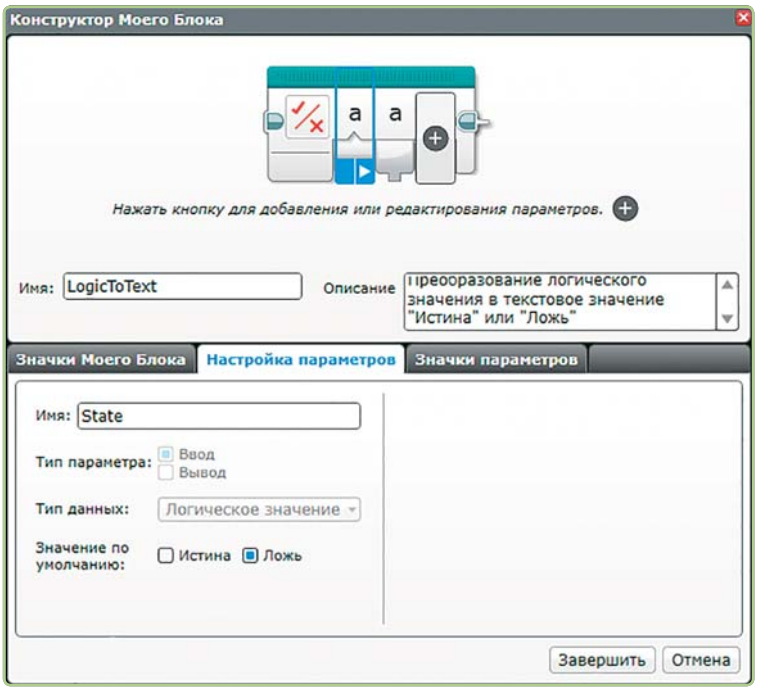


*Рис. 12.11. Вкладка **Настройка параметров** (Parameter Setup) для первого параметра*

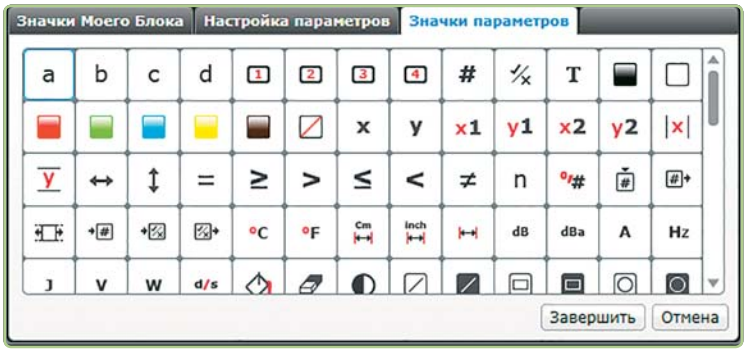


*Рис. 12.12. Вкладка **Значки параметров** (Parameter Icons)*

9. Введи значение **Value** в поле **Имя** (Name).

Теперь необходимо выбрать значок для параметра **Ввод** (Input). На выбор предоставляется множество значков, поэтому ты почти всегда сможешь найти тот, который передает смысл лучше, чем значок, выбранный по умолчанию.

10. Щелкни по вкладке **Значки параметров** (Parameter Icons) (рис. 12.12).

11. Выбери значок для логических значений (). Теперь он должен отображаться в блоке в верхней части окна (рис. 12.13).

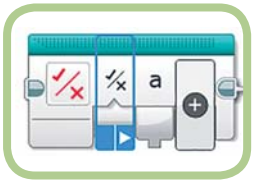


Рис. 12.13. Блок с выбранным значком

Первый параметр готов. Теперь выдели второй параметр, укажи его имя и выбери значок.

12. Выдели второй параметр.

13. Щелкни по вкладке **Настройка параметров** (Parameter Setup).

14. Введи **Result** в поле **Имя** (Name).

15. Щелкни по вкладке **Значки параметров** (Parameter Icons).

16. Выбери значок для текстовых значений (**T**).

Теперь блок должен выглядеть так, как показано на рис. 12.14.

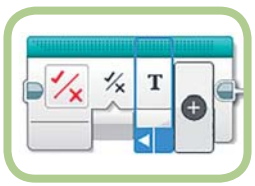


Рис. 12.14. Блок после выбора значков для обоих параметров

ВНИМАНИЕ! Прежде чем щелкнуть по кнопке **Завершить** (Finish), убедись в том, что у тебя есть все необходимые параметры с правильно выбранными настройками и значками. После щелчка по кнопке **Завершить** (Finish) и создания контейнера «Мой блок» ты не сможешь вернуться, чтобы добавить, удалить или изменить его параметры или значки.

17. Щелкни по кнопке **Завершить** (Finish), чтобы создать контейнер **LogicToText**.

После перемещения блоков ближе друг к другу исходная программа должна выглядеть так, как на рис. 12.15.

Дважды щелкни по контейнеру **LogicToText**, чтобы открыть его в области программирования (рис. 12.16). Если параметр **Вывод** (Output) отображается не там, где надо, его довольно легко перетащить вправо (рис. 12.17). Удобнее помещать вводы слева, а выводы справа. Кроме того, не помешает добавить некоторые комментарии. Это позволит тебе или другому программисту легко понять, что и как делает данный контейнер «Мой блок».

Попробуй запустить программу *LogicToTextBuilder*. При этом на экране должно отобразиться слово "True" («Истина»), если кнопка датчика касания нажата, и слово "False" («Ложь»), если нет.

Добавление, удаление и перемещение параметров

Теперь подробнее рассмотрим процесс добавления, удаления и изменения параметров в окне **Конструктор мой блок** (My Block Builder). Выбери блок, открой окно **Конструктор мой блок** (My Block Builder) и добавь несколько параметров. На рис. 12.18 показаны элементы управления для контейнера «Мой блок» с четырьмя параметрами.

- Выбранный параметр выделен синим цветом. Ты можешь изменить его, используя вкладки **Настройка параметров** (Parameter Setup) и **Значки параметров** (Parameter Icons).
- С помощью кнопки **Добавить параметр** (Add Parameter) ты можешь добавить в блок новый параметр.
- Кнопка **Удалить параметр** (Remove Parameter) необходима для удаления выделенного параметра. Удалить можно

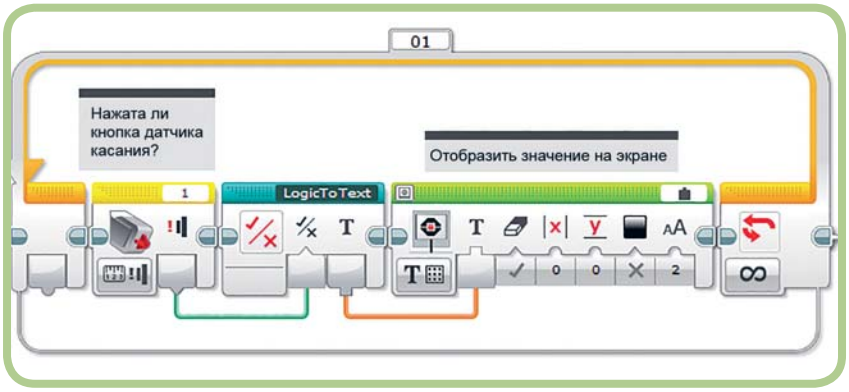


*Рис. 12.15. Использование контейнера **LogicToText***

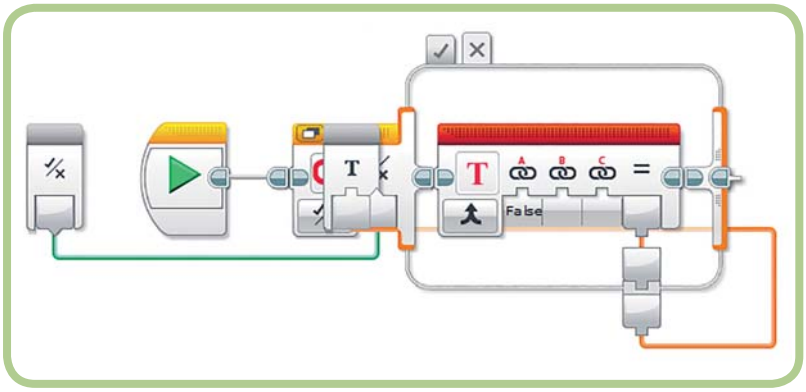


*Рис. 12.16. Контейнер **LogicToText***

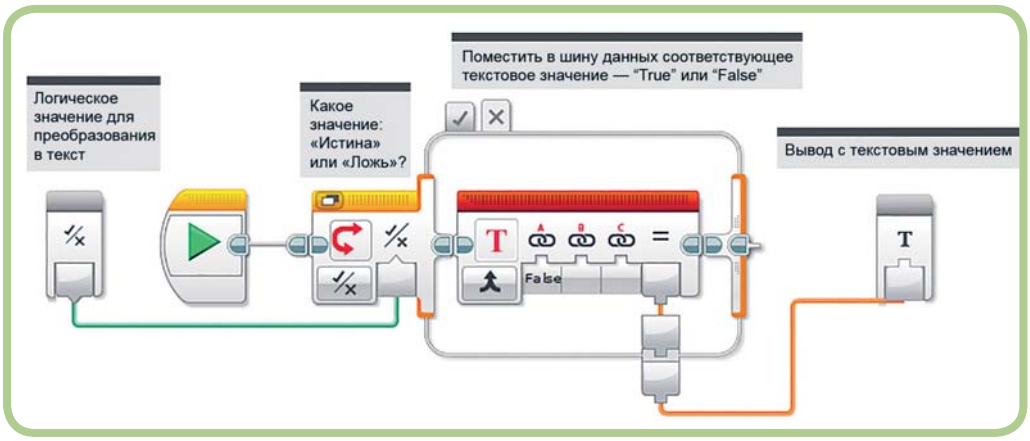


Рис. 12.17. После перемещения вывода вправо и добавления комментариев

только те параметры, которые были добавлены с помощью кнопки **Добавить параметр** (Add Parameter). Параметры, созданные автоматически с помощью шины данных, удалить нельзя.

– С помощью кнопки **Переместить параметр влево** (Move Parameter Left) ты можешь переместить выбранный параметр влево.

– Используя кнопку **Переместить параметр вправо** (Move Parameter Right), выбранный параметр перемещается вправо.

Рис. 12.18. Элементы управления для работы с вводами и выводами блока

Вкладка «Настройка параметров»

Вкладка **Настройка параметров** (Parameter Setup) определяет внешний вид и поведение параметра. Ты уже видел эту вкладку для контейнера **LogicToText** на рис. 12.11. На рис. 12.19 показана вкладка **Настройка параметров** (Parameter Setup) для одного из параметров, добавленных с помощью окна **Конструктор мой блок** (My Block Builder). Для добавленных таким способом параметров ты можешь указать, должны ли они представлять собой ввод или вывод блока, а также выбрать тип данных. Если для настройки **Тип параметра** (Parameter Type) выбран вариант **Ввод** (Input), а для настройки **Тип данных** (Data Type) — **Число** (Number), правая часть вкладки предоставит тебе на выбор ползунковый регулятор для ввода значения, если ты предпочитаешь использовать его вместо обычного ввода. При выборе одного из ползунковых регуляторов ты все равно сможешь ввести значение или использовать шину данных (как в случае с параметром **Мощность** (Power) блока **Рулевое управление** (Move Steering)).

Контейнер DisplayNumber

В этом разделе ты создашь контейнер **DisplayNumber**. Этот блок будет полезен для отображения чисел на экране модуля, предоставления пользователю обратной связи во время выполнения программы, информации, связанной с отладкой программы, или отображения результата эксперимента.

На рис. 12.20 показаны блоки, используемые программой *SoundMachine* для отображения уровня громкости, который контролируется мотором B. С помощью блока **Вращение мотора** (Motor Rotation) считывается значение, блоком **Текст** (Text) добавляются метка и единица измерения, с помощью блока **Экран** (Display) полученный текст выводится на экран модуля EV3. Необходимо, чтобы в контейнере «Мой блок» выполнялась аналогичная функция, но также была возможность передать в него в качестве входных данных любое число. Существуют и другие параметры этого контейнера, которые было бы полезно настраивать в каждом конкретном случае, например, текст для метки и единицы измерения.

Посмотри на остальные вводы блоков **Текст** (Text) и **Экран** (Display), изображенных на рис. 12.20, и подумай, какие из них должны быть настраиваемыми и какими должны быть их значения по умолчанию. Далее приведен список параметров, которые было бы логично включить.

– **Блок Текст (Text)** — **A.** Метка, отображаемая перед числом. Значением по умолчанию является пустая строка.

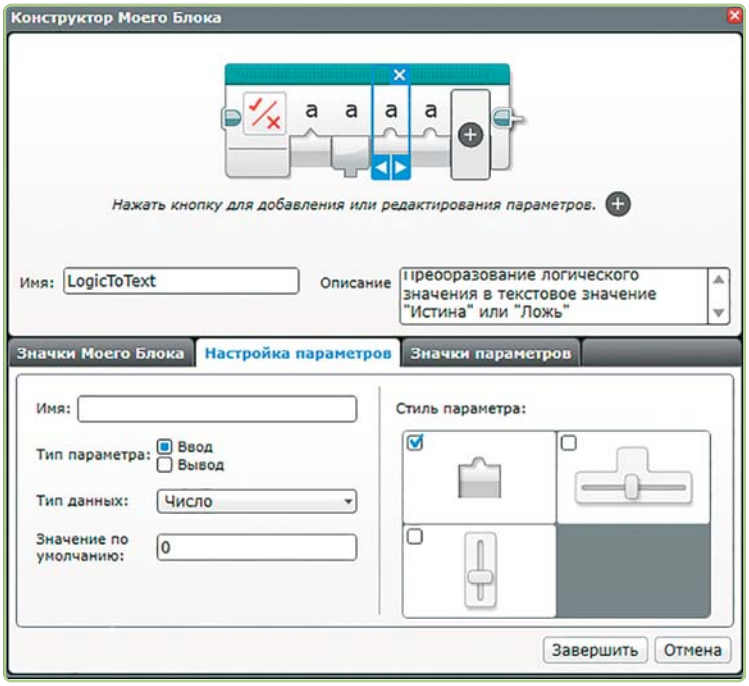


*Рис. 12.19. Вкладка **Настройка параметров** (Parameter Setup) для нового параметра*

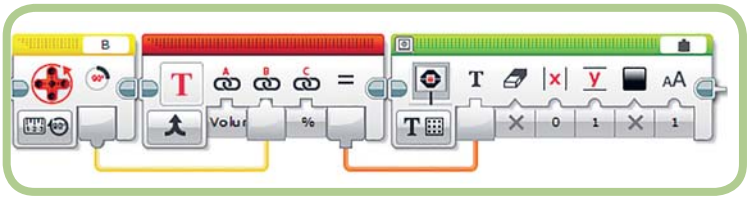


Рис. 12.20. Отображение уровня громкости звука

— **Блок Текст (Text)** — **С.** Единица измерения, отображаемая после числа. Значением по умолчанию является пустая строка.
— **Блок Экран (Display)** — **Очистить экран (Clear Screen).** В зависимости от того, где используется блок, тебе может понадобиться очистить экран перед отображением значения. В качестве значения по умолчанию я выбрал бы вариант **Истина** (True) в соответствии с принципом работы блока **Экран** (Display).
— **Блок Экран (Display)** — **Строка (Row).** Если ты решишь отобразить несколько значений, строку нужно будет изменить, поэтому этот параметр должен быть настраиваемым. В качестве значения по умолчанию разумно задать 0, чтобы текст отображался вверху экрана.

Параметры контейнера «Мой блок» должны быть настраиваемыми. После их определения необходимо подключить к каждому из них шину данных. В качестве источника входных данных для блоков можно применить блок **Константа** (Constant) в начале программы. Для каждого блока **Константа** (Constant) необходимо выбрать режим с учетом типа данных соответствующего параметра (для метки и единицы измерения должно использоваться текстовое значение, для параметра **Очистить экран** (Clear Screen) — логическое, а для параметра **Строка** (Row) — числовое). Для хранения выходных данных блоков удобно использовать блок **Переменная** (Variable) в режиме **Записать** (Write) в конце программы. Блок **DisplayNumber** не предусматривает выводов, поэтому в данном случае будем использовать только блоки **Константа** (Constant).

Далее перечислены действия для создания контейнера **DisplayNumber**:

1. Создай новую программу с именем *DisplayNumberBuilder*.
2. Скопируй три блока, показанные на рис. 12.20, в новую программу.
3. Добавь блок **Константа** (Constant) для каждого настраиваемого параметра. Выбери режим с учетом подходящего типа данных и соедини блок **Константа** (Constant) с параметром, используя шину данных.

На данном этапе программа должна выглядеть так, как показано на рис. 12.21. Теперь ты готов к созданию контейнера «Мой блок».

4. Выдели блоки **Текст** (Text) и **Экран** (Display).
5. Выбери команду меню **Инструменты** (Tools) ⇒ **Конструктор мой блок** (My Block Builder).
6. Введи имя DisplayNumber и добавь описание.
7. Выбери значок для нового блока.
8. Поочередно выбери каждый параметр, задай для него имя, значение по умолчанию и значок. Пример выбора имен и значков показаны на рис. 12.22.
9. Щелкни по кнопке **Завершить** (Finish).

После щелчка по кнопке **Завершить** (Finish) будет создан контейнер **DisplayNumber**, а блоки **Текст** (Text) и **Экран** (Display) в программе *DisplayNumberBuilder* будут заменены новым блоком.

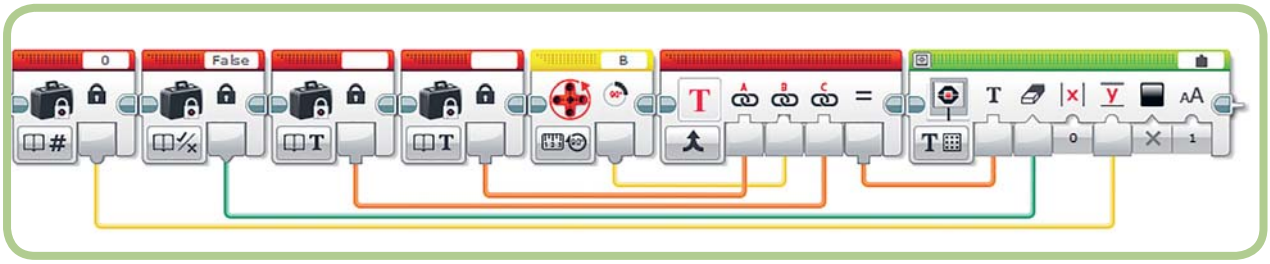


Рис. 12.21. Программа DisplayNumberBuilder

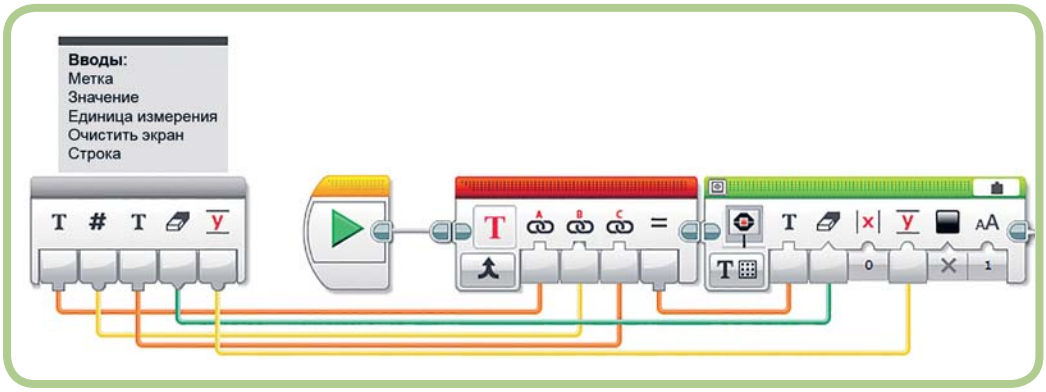


Рис. 12.22. Контейнер ***DisplayNumber***

Если тебе понадобится вернуть программу в исходное состояние, выбери команду меню **Редактировать** (Edit) ⇒ **Отменить** (Undo) или нажми комбинацию клавиш **Ctrl+Z**. Это может быть полезно в случае, если ты решишь изменить любой из параметров в блоке **DisplayNumber**; эта функция позволяет вернуться к версии программы, используемой для создания контейнера «Мой блок».

После создания контейнера «Мой блок» рекомендуется открыть его и убедиться в том, что он выглядит так, как надо. Кроме того, ты также можешь добавить некоторые комментарии с описанием параметров и способа использования блока. На рис. 12.22 показан контейнер **DisplayNumber** после добавления комментариев.

После создания контейнера «Мой блок» ты можешь применять его в любой из своих программ (для использования контейнера в другом проекте тебе придется сначала скопировать его). Например, на рис. 12.23 показана программа *SoundMachine*, в которой блоки **Текст** (Text) и **Экран** (Display) заменены блоками **DisplayNumber**. Теперь программа является более компактной, что облегчает понимание ее логики и проясняет принцип работы блоков **DisplayNumber**.

Изменение параметров контейнера «Мой блок»

Как показано в разделе «Изменение контейнера «Мой блок» этой главы, изменить содержимое контейнера очень просто. Для этого нужно всего лишь открыть его и отредактировать блоки так, как в обычной программе. К сожалению, изменить параметры контейнера «Мой блок» не так легко. Если ты решишь добавить новый параметр, изменить имя существующего параметра или значение по умолчанию, придется создать контейнер заново.

Воссоздать контейнер «Мой блок» будет гораздо проще при наличии программы, например *DisplayNumberBuilder*, которую можно использовать в качестве отправной точки. Если у тебя нет такой программы, ты можешь создать ее, скопировав блоки из контейнера «Мой блок» в новую программу и добавив блоки **Константа** (Constant) или **Переменная** (Variable) для любых необходимых шин данных. После создания программы ты можешь заново создать контейнер «Мой блок», выполнив следующие четыре действия:

1. Открой существующий контейнер «Мой блок» и переименуй его, добавив к его имени слово “Old” (дважды щелкни по имени на вкладке вверху программы и введи новое имя).

2. Создай контейнер «Мой блок» из программы *DisplayNumberBuilder*.

3. Пройдись по всем программам, в которых используется контейнер «Мой блок», и замени старый контейнер новым.

4. Удали старый контейнер «Мой блок».

После переименования контейнера «Мой блок» среда EV3 будет использовать новое имя во всех программах проекта, в которых задействован этот блок. Таким образом, изменение имени с **DisplayNumber** на **DisplayNumberOld** приведет к обновлению всех программ, в результате чего в них будет применяться блок **DisplayNumberOld**, и в результате придется вручную изменить программы так, чтобы они использовали новый блок **DisplayNumber**.

Переменные и мои блоки

В своих контейнерах «Мой блок» ты можешь применять переменные так же, как и в обычной программе. Важно знать о том, что существует один список общих переменных для программы и всех входящих в нее контейнеров «Мой блок». Переменные EV3 называются *глобальными переменными*, поскольку к ним можно получить доступ из любой части программы, включая контейнеры «Мой блок».

МОИ БЛОКИ И ОТЛАДКА ПРОГРАММЫ

Во время работы программы ты можешь использовать программное обеспечение EV3, чтобы увидеть, какой блок выполняется в данный момент и какие значения находятся в шинах данных. Я считаю, что эти две функции невероятно полезны при отладке программы. К сожалению, они работают только для основной программы, поэтому ты не можешь их применять для определения того, что происходит внутри контейнера «Мой блок». Это означает, что, если контейнер «Мой блок» содержит ошибку, ее будет сложнее найти и исправить.

Однако если ты тщательно протестируешь свои контейнеры «Мой блок» и убедишься в их корректной работе, найти ошибки в основной программе станет намного проще. Рекомендуется выполнить тестирование кода программы *DisplayNumberBuilder* перед созданием контейнера «Мой блок», а небольшие тестовые программы помогут убедиться в том, что контейнер «Мой блок» работает так, как запланировано. Использование контейнера «Мой блок» уменьшает размер твоей программы и, следовательно, количество блоков, которые требуется проверить для выявления ошибки. Использование протестированных контейнеров «Мой блок» для решения распространенных задач позволяет значительно сократить время отладки.

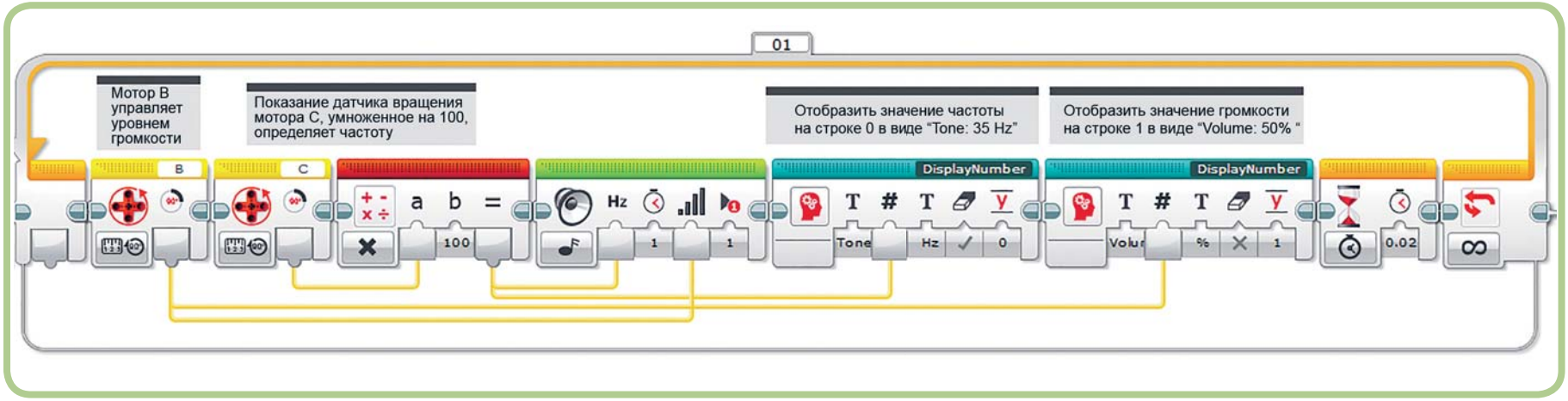


Рис. 12.23. Программа SoundMachine, содержащая контейнер ***DisplayNumber***

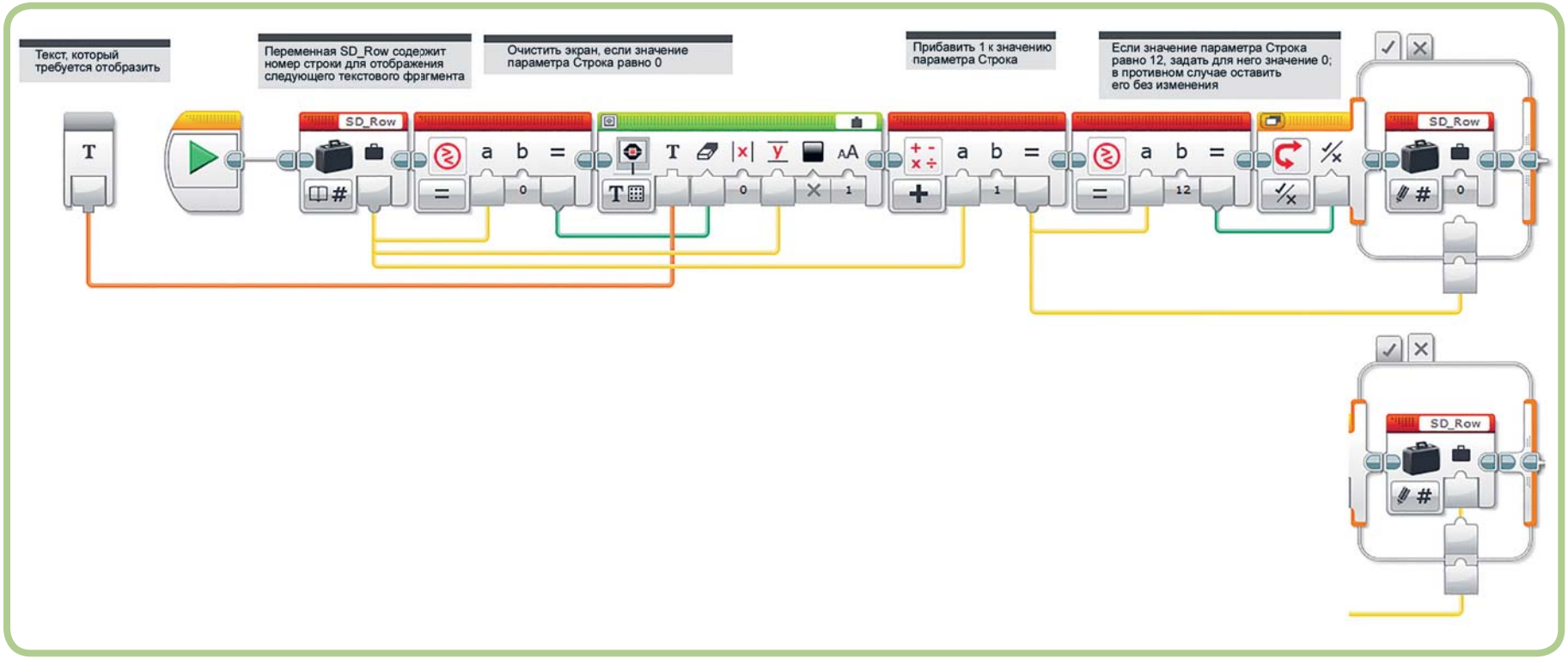


Рис. 12.24. Контейнер ***ScrollDisplay***

Ты можешь выбрать переменные для обмена информацией между основной программой и используемыми в ней контейнерами «Мой блок» или между двумя (или более) контейнерами «Мой блок». Например, если ты разделишь программу *WallFollower* на три контейнера «Мой блок», то сможешь использовать одну переменную для управления параметром **Мощность** (Power) всех блоков **Рулевое управление** (Move Steering), несмотря на то, что эти блоки **Рулевое управление** (Move Steering) будут находиться в трех разных контейнерах «Мой блок».

Переменные также полезны, если тебе нужно, чтобы в контейнере «Мой блок» сохранилось некоторое значение. В качестве примера рассмотрим контейнер **ScrollDisplay**, изображенный на рис. 12.24. Этот контейнер «Мой блок» предназначен для отображения прокручиваемого текста на экране модуля EV3. Текст, который требуется отобразить, передается блоку через шину данных. При первом выполнении с помощью этого блока очищается экран и отображается текст на первой строке. При следующем выполнении текст отображается на второй строке, затем на третьей и т. д., пока текст не отобразится на последней строке. При очередном выполнении с помощью блока очищается экран, а затем снова отображается текст на первой строке.

В этом блоке применяется переменная с именем SD_Row для определения номера строки для отображения следующего текстового фрагмента. Используем префикс SD_, поскольку эта переменная применяется в блоке **ScrollDisplay**. Использование префикса, основанного на имени блока, помогает избежать случайного использования и изменения уже существующей в основной программе или в другом контейнере «Мой блок» переменной.

В этом блоке используется много шин данных, однако его базовый принцип работы довольно прост. На экране модуля EV3 могут отображаться 12 строк текста с номерами строк от 0 до 11. Необходимо, чтобы при каждом выполнении блока текст отображался на следующей строке. По достижении последней строки (11) экран должен очиститься, и при очередном выполнении на нем должен начать появляться текст, начиная с верхней строки (0).

Значение переменной SD_Row передается в блок **Экран** (Display) в качестве параметра **Строка** (Row). В блоке **Сравнение** (Compare) происходит анализ, превышает ли параметр **Строка** (Row) значение 0, чтобы принять решение относительно очистки экрана. После отображения текста значение параметра **Строка** (Row) увеличивается на 1. Затем с помощью блоков **Сравнение** (Compare) и **Переключатель** (Switch) проверяется значение: если оно равно 12, в переменной SD_Row сохраняется значение 0; в противном случае в ней сохраняется новое инкрементированное значение. Это обновленное значение будет использоваться для задания значения параметра **Строка** (Row) при следующем выполнении контейнера «Мой блок».

Дальнейшее исследование

Вот еще два действия, которые ты можешь выполнить, чтобы попрактиковаться в создании контейнеров «Мой блок»:

1. Создай блок **DisplayLogic**, который работает так же, как блок **DisplayNumber**, но в котором используются логические значения. Используй блок **LogicToText** в своем новом блоке для преобразования логического значения в текстовое.

2. Программа *LightPointer*, описанная в гл. 11, состоит из двух частей: одна отвечает за определение направления, в котором расположен источник света, а другая — за разворот робота в этом направлении. Создай два контейнера «Мой блок» для этих двух частей программы. Вместо переменной Position используй шины данных для передачи информации о положении робота из одного раздела программы в другой.

ПРАКТИКУМ 12.2

Добавь параметры громкости и продолжительности в контейнер Chime и примени их к каждому из блоков Звук (Sound). Для каждого параметра необходимо задать разумные значения по умолчанию (например, в качестве значения по умолчанию для параметра громкости не стоит задавать 0). Убедись в том, что программа *DoorChime* использует обновленный блок.

Заключение

Создание собственных блоков — простой, но мощный способ повторного применения кода, который делает программы более понятными и с меньшим количеством ошибок. В этой главе были описаны методы создания контейнеров «Мой блок» разной сложности. Такие простые блоки, как **Chime**, облегчают повторное использование кода и позволяют поддерживать адекватный размер программы. С помощью более сложных блоков с большим количеством параметров, вроде блока **DisplayNumber**, можно избежать без многократного переписывания одного и того же сложного кода.

13

Математика и логика

В этой главе ты познакомишься с режимом **Дополнения** (Advanced) блока **Математика** (Math), благодаря которому программы могут выполнять более сложные вычисления. Ты также узнаешь о тесно связанном с блоком **Математика** (Math) блоке **Логические операции** (Logic Operations), который позволяет объединять логические значения, благодаря чему твои программы могут принимать сложные решения. Кроме того, ты узнаешь о других блоках, работающих с числами, в их числе блоки **Интервал** (Range), **Случайное значение** (Random) и **Округление** (Round).

Режим «Дополнения» блока «Математика»

Режим **Дополнения** (Advanced) блока **Математика** (Math) позволяет выполнять сложные вычисления. Мы уже использовали блок **Математика** (Math) в нескольких программах, — тогда мы выбирали одну операцию (сложение, умножение и т. д.) и задавали два параметра. Режим **Дополнения** (Advanced), приведенный на рис. 13.1, позволяет объединить несколько операций и до четырех входных параметров (с именами от *a* до *d*), чтобы вычислить результат более сложных математических выражений.

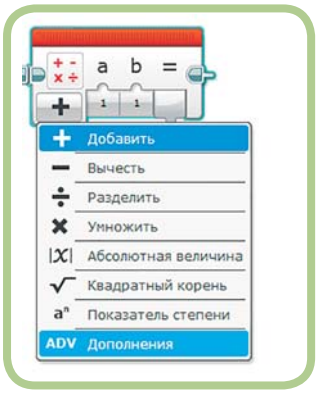


*Рис. 13.1. Блок **Математика** (Math) в режиме **Дополнения** (Advanced)*

Для ввода выражения щелкни по соответствующему полю. Появится небольшое окно с полем ввода в верхней части и списком операторов и функций под ним (рис. 13.2). Ты можешь просто ввести свое выражение, используя общепринятые символы математических операторов, или щелкнуть по одному из операторов или функций в списке, чтобы добавить его в текущее выражение.

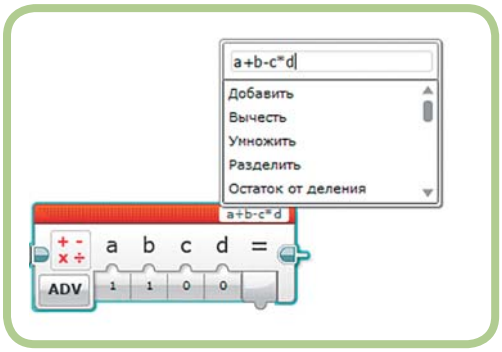


Рис. 13.2. Ввод выражения

Поддерживаемые операторы и функции

В табл. 13.1 приведен список поддерживаемых операторов, большая часть которых тебе, вероятно, знакома. Для использования этих операторов ты можешь просто ввести соответствующий символ или выбрать оператор из списка в окне для ввода выражения.

Табл. 13.1. Поддерживаемые операторы

Оператор	Описание
+	Сложение.
–	Вычитание или обозначение отрицательного числа. 5–3 — это вычитание. *–a* — это отрицательное число.
*	Умножение.
/	Деление.
^	Возведение в степень. 2^3 означает 2 в степени 3 или 2^3.
%	Деление по модулю. Возвращает остаток от деления двух чисел. 5%2 = 1, поскольку число 2 содержится в числе 5 два раза, а остаток равен 1.

В табл. 13.2 приведен список поддерживаемых функций вместе с их кратким описанием. Большинство из них относится к достаточно продвинутому уровню математики, и в этой книге мы будем использовать только некоторые из этих функций.

Табл. 13.2. Поддерживаемые функции

Функция	Описание
floor()	Округляет значение в меньшую сторону до ближайшего целого числа. floor(4.7) = 4, floor(4.1) = 4, floor(–4.4) = –5.
ceil()	Округляет значение в большую сторону до ближайшего целого числа. ceil(4.7) = 5, ceil(4.1) = 5, ceil(–4.4) = –4.
round()	Округляет значение до ближайшего целого числа. round(4.7) = 5, round(4.1) = 4, round(–4.4) = –4, round(–4.7) = –5.
abs()	Возвращает абсолютное значение. abs(5) = 5, abs(–5) = 5.
log()	Возвращает логарифм числа по основанию 10.
ln()	Возвращает натуральный логарифм числа.
sin()	Возвращает синус угла. Все тригонометрические функции работают со значениями в градусах.
cos()	Возвращает косинус угла.
tan()	Возвращает тангенс угла.
asin()	Возвращает арксинус угла.
acos()	Возвращает арккосинус угла.
atan()	Возвращает арктангенс угла.
sqrt()	Возвращает положительный квадратный корень числа. sqrt(16) = 4.

Щелчок по функции в списке добавляет имя функции и открывающую скобку; затем необходимо ввести значение, к которому требуется применить добавленную функцию, и закрывающую скобку. Например, если ты хочешь округлить параметр *a* до ближайшего целого числа, щелкни по элементу списка **Округление** (Round), чтобы добавить фрагмент *round(*. Затем введи ***a)*** (рис. 13.3).

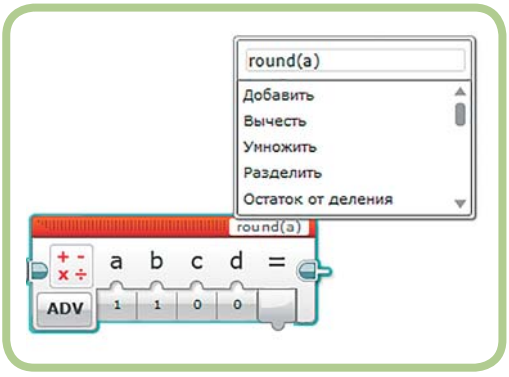


Рис. 13.3. Ввод выражения

Оператор деления по модулю

Оператор деления по модулю (оператор деления с остатком) (%) возвращает остаток от деления одного числа на другое. Например, 7 % 4 = 3, поскольку число 4 содержится в числе 7 один раз с остатком 3. (Выражение 7 % 3 читается как «семь по модулю три».) Этот оператор обладает некоторыми свойствами, которые делают его очень полезным в процессе программирования.

В табл. 13.3 перечислены результаты операции *a* % *3* для возрастающих значений *a*. Обрати внимание на то, что результат начинается с 0, увеличивается до 2, а затем снова возвращается к 0. Оператор деления по модулю полезен, когда тебе нужно, чтобы значение увеличивалось, а всякий раз при достижении определенной точки возвращалось к исходной величине. Такая ситуация рассмотрена в гл. 12, в примере с контейнером **ScrollDisplay**, где номер строки начинался с 0, увеличивался до 11, а затем снова возвращался к 0. На рис. 13.4 показана часть блока **ScrollDisplay**, увеличивающая номер строки, проверяющая, достигло ли оно 12, а затем сохраняющая новое значение. Значение, поступающее в блок **Математика** (Math) слева, представляет собой номер предыдущей строки.

Табл. 13.3. Поведение оператора деления по модулю

A	a % 3
0	0
1	1
2	2
3	0
4	1
5	2
6	0
7	1

Используя блок **Математика** (Math) в режиме **Дополнения** (Advanced) и оператор деления по модулю, можно достичь той же цели с помощью всего двух блоков (рис. 13.5). Выражение (*a* + 1)% 12 прибавляет 1 к значению *a* (номер предыдущей строки), а затем возвращает остаток от деления этого значения на 12. Таким образом, номер строки начинается с 0, увеличивается до 11, а затем возвращается к 0, как в исходном коде.

Ошибки блока «Математика»

Если ты введешь выражение, которое будет невозможно вычислить правильно в блоке **Математика** (Math), на экране отобразится ошибка. Блок **Математика** (Math) по-разному реагирует на различные ошибки, которые могут вызвать проблемы при передаче его результата другим блокам программирования.

Ты можешь проверить блок **Математика** (Math) на наличие ошибок, просмотрев значение, содержащееся в выходной шине данных, или отобразив это значение на экране модуля EV3 с помощью блока **Экран** (Display). Например, если ты попытаешься разделить число на 0, то в шине данных, выходящей из блока, появится значение **Бесконечность** (Infinity), как показано на рис. 13.6, которое отобразится на экране модуля EV3 в виде «Inf».

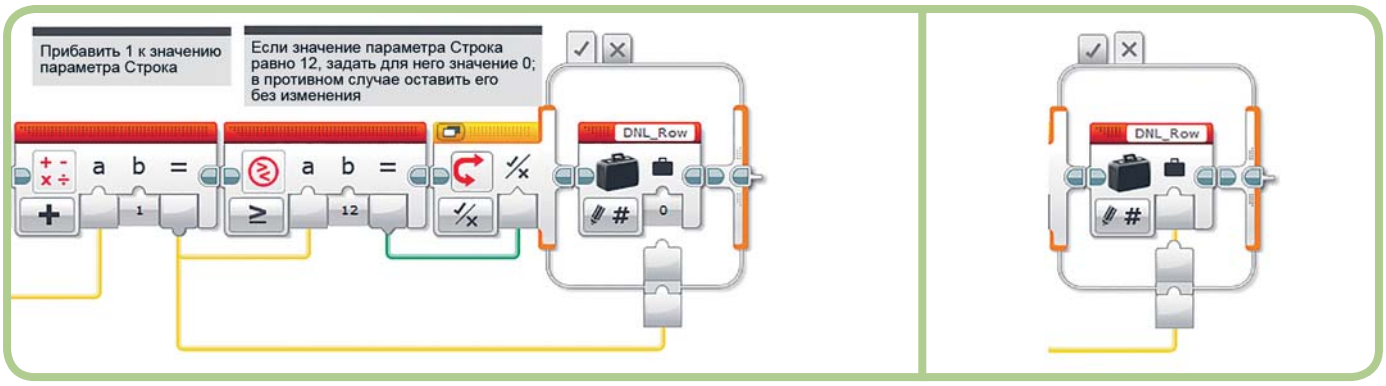


*Рис. 13.4. Блоки **ScrollDisplay** для вычисления и сохранения номера очередной строки*

Рис. 13.5. Вычисление и сохранение номера очередной строки с помощью двух блоков

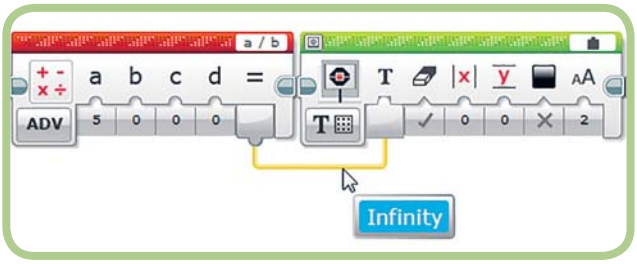


*Рис. 13.6. Ошибка блока **Математика** (Math) в шине данных при попытке деления на 0*

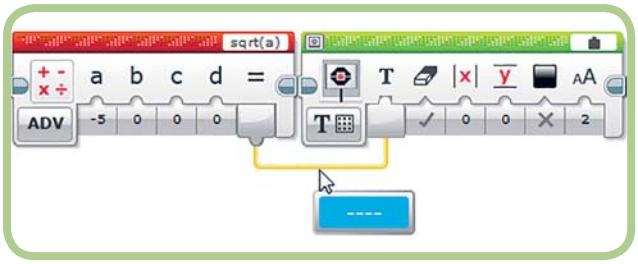


*Рис. 13.7. Ошибка блока **Математика** (Math) в шине данных при попытке извлечь квадратный корень из отрицательного числа*

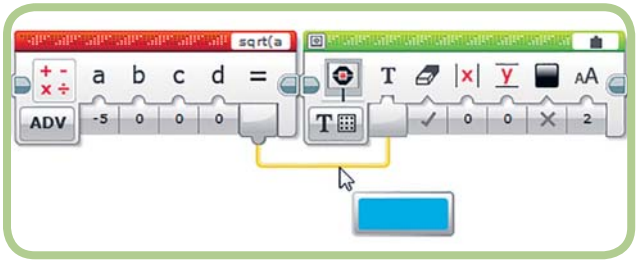


*Рис. 13.8. Ошибка блока **Математика** (Math) в шине данных при вводе недействительного выражения*

Другая распространенная ошибка возникает при попытке извлечь квадратный корень из отрицательного числа. При этом выдается специальное значение, которое отображается в виде ---- в шине данных или на экране модуля EV3 (рис. 13.7). Аналогично, если ты введешь выражение, в котором не хватает значения или скобки и которое не может быть вычислено, например, *sqrt(a* или *a + b*°), в шине данных не будет ничего, как на рис. 13.8, а на экране модуля при этом отобразится «----».

Передача значения в блок в качестве входного параметра часто приводит к странным результатам. Например, если ты используешь квадратный корень из отрицательного числа в качестве параметра **Мощность** (Power) блока **Рулевое управление** (Move Steering), этот блок будет функционировать так, будто для параметра **Мощность** (Power) задано значение 100. То же самое значение, используемое для настройки параметра **Продолжительность** (Duration) в градусах, заставит блок работать так, будто для этого параметра задано значение 0. Использование этого значения в качестве параметра **Рулевое управление** (Steering) приведет к тому, что два мотора будут колебаться вперед-назад, демонстрируя поведение, которое не соответствует ни одному из значений параметра **Рулевое управление** (Steering). Следи за этими обозначениями, поскольку из-за них могут возникать программные ошибки.

Программа LineFollower с пропорциональным управлением

В этом разделе вернемся к нашей программе *LineFollower* и используем режим **Дополнения** (Advanced) блока **Математика** (Math) для усовершенствования механизма настройки параметра **Рулевое управление** (Steering). Часть этой программы, которая корректирует значение параметра **Рулевое управление** (Steering) на основе показаний датчика, называется *алгоритмом управления*. Улучшение этого алгоритма управления позволяет роботу TriBot двигаться более плавно и следовать вдоль линий с более резкими поворотами.

Алгоритм управления, разработанный в гл. 6 и переработанный в гл. 9 (рис. 13.9), называется *трехпозиционным регулятором*, поскольку в зависимости от показания датчика программа заставляет робота выполнять одно из трех действий: двигаться прямо, поворачивать налево или поворачивать направо. Основная проблема в данном случае заключается в том, что при необходимости повернуть робот всегда поворачивает на одно и то же количество градусов. Вне зависимости от того, требуется ли ему совершить резкий или мягкий поворот, он использует одно и то же фиксированное значение параметра **Рулевое управление** (Steering).

Было бы лучше, если бы значение параметра **Рулевое управление** (Steering) зависело от резкости линии и позволяло роботу поворачивать мягко при движении вдоль плавных кривых и выполнять более резкие повороты на острых углах. Так программа могла бы быстрее реагировать на изменения в направлении линии и при этом обеспечивать плавное движение робота вдоль более или менее прямых ее участков. Этот подход называется *пропорциональным регулированием*, поскольку изменение параметра **Рулевое управление** (Steering) пропорционально или напрямую зависит от расстояния между роботом и краем линии.

При пропорциональном регулировании управляющая переменная (в данном случае направление движения) изменяется в зависимости от *целевого* и *входного значения*. В рассматриваемом примере входным значением является показание датчика цвета. Целевое значение — это показание датчика цвета в тот момент, когда он находится непосредственно над краем линии. Это значение определено ранее в гл. 6, где было выбрано среднее значение между показаниями датчика цвета при его нахождении над серединой линии и полностью вне линии (в моем случае оно равно 52).

Разность между целевым и входным значениями называется *значением ошибки*. Ты можешь представить значение ошибки как разницу между желаемым и фактическим положением робота. Для вычисления значения параметра **Рулевое управление** (Steering) умножаем значение ошибки на значение коэффициента усиления. *Значение коэффициента усиления* определяет скорость реакции робота на изменения значения ошибки. Небольшой коэффициент усиления заставляет робота двигаться медленно, а это означает, что он может

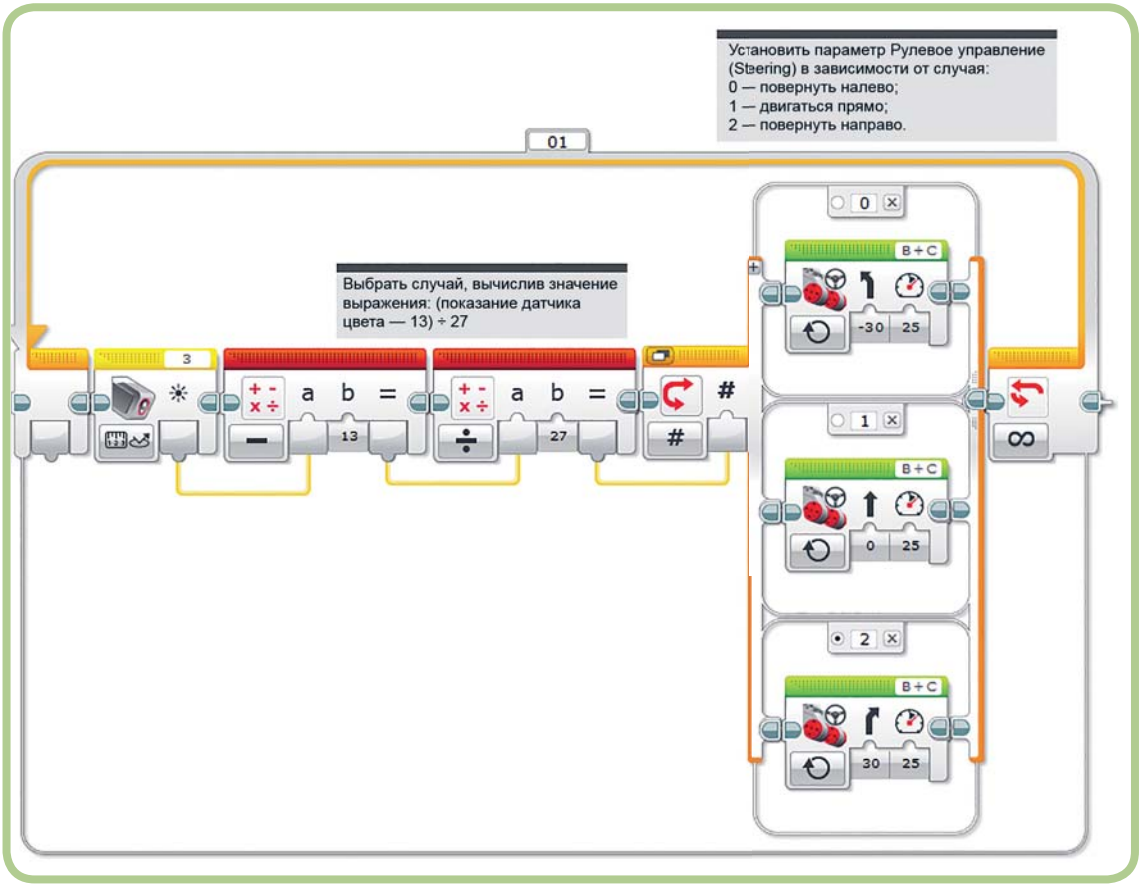


Рис. 13.9. *Трехпозиционный регулятор программы LineFollower из гл. 9*

реагировать недостаточно быстро при необходимости выполнить резкий поворот, но при этом уменьшает его движение из стороны в сторону на более или менее прямых участках линии. Большее значение коэффициента усиления обусловливает более быструю реакцию, но вместе с тем робот может двигаться рывками. Выбор значения коэффициента усиления называется *настройкой* регулятора и обычно осуществляется методом проб и ошибок.

Ниже показаны уравнения для вычисления значения ошибки и значения параметра **Рулевое управление** (Steering):

Значение ошибки = Целевое значение — Показание датчика.
Значение параметра ***Рулевое управление*** *= Значение ошибки × Коэффициент усиления*

Можно объединить эти два выражения в одно: (Целевое значение — Показание датчика) × Коэффициент усиления и использовать единственный блок **Математика** (Math) в режиме **Дополнения** (Advanced) для вычисления значения параметра **Рулевое управление** (Steering). Два значения (целевое и коэффициент усиления) являются константами, поэтому в программе будем использовать блоки **Константа** (Constant) для передачи значений в блок **Математика** (Math), что упростит их корректировку. Изменение значения в блоке **Константа** (Constant) требует меньших усилий, а вероятность возникновения ошибок при этом ниже, чем при изменении выражения в блоке **Математика** (Math).

Готовая программа показана на рис. 13.10. В качестве коэффициента усиления задано значение 0,7, которое хорошо подходит для тестовой линии. Попробуй другие значения, чтобы выбрать самое подходящее. Ниже показаны входные параметры блока **Математика** (Math):

- ***a*** *Показание датчика цвета*
- ***b*** *Целевое значение*
- ***c*** *Значение коэффициента усиления*

Подставив эти параметры в описанное выше выражение для вычисления значения параметра **Рулевое управление** (Steering), получим $(b - a) \times c$. Однако в результате решения этого выражения получаются такие значения, при которых робот движется в противоположном направлении по сравнению с предыдущей версией программы *LineFollower*. В результате робот будет оставаться у правой стороны линии, а не у левой, как в случае предыдущей версии. Для согласования этой программы с предыдущими версиями изменим знак результата выражения следующим образом: $-(b - a) \times c$.

Протестируй данную программу с использованием собственного целевого значения и коэффициента усиления. Эта программа должна обеспечить лучший результат по сравнению с программой *LineFollower* с трехпозиционным регулятором.

Таймеры модуля EV3

В следующей программе мы будем использовать таймеры модуля EV3. Модуль EV3 предусматривает восемь встроенных таймеров, работающих как секундомеры. Ты можешь использовать *таймер модуля EV3* для определения длительности работы программы или количества времени, необходимого роботу для выполнения конкретной задачи. Как правило, перед выполнением действия показание таймера сбрасывается и считывается после выполнения действия. Это похоже на способ применения датчика вращения мотора и гироскопического датчика в некоторых ранее созданных в этой книге программах. Ты можешь рассматривать таймер в качестве простого датчика времени.

Поскольку модуль EV3 предусматривает восемь таймеров, с их помощью ты можешь выполнять несколько задач в рамках одной и той же программы для достижения различных целей. Вот некоторые идеи, касающиеся их применения:

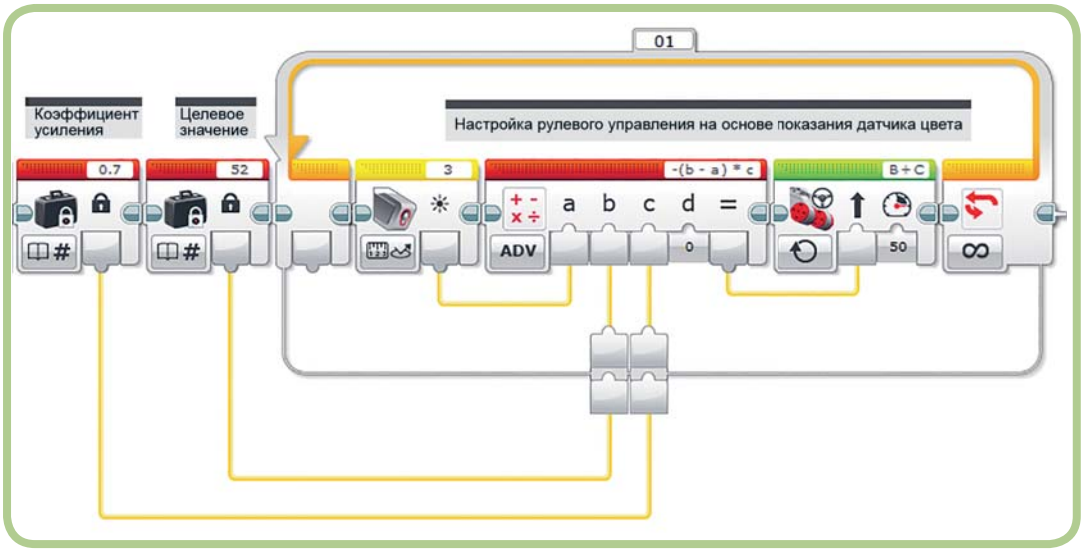


Рис. 13.10. *Программа LineFollower* ***с пропорциональным регулированием***

— Определи длительность выполнения своей программы и используй эту информацию для сравнения эффективности различных подходов. Например, можно узнать, сколько времени требуется твоему роботу для того, чтобы найти выход из лабиринта, используя разные программы, а затем применить эту информацию для выбора более быстрого варианта.
— Определи длительность выполнения разных фрагментов своей программы, чтобы узнать, нельзя ли ускорить их работу.
— Используй таймеры, чтобы заставить программу периодически выполнять определенные действия. Например, при проведении эксперимента ты можешь использовать таймер для того, чтобы считывать показания датчика каждые 10 секунд в течение 5 минут.
— Используй таймер для ограничения периода ожидания достижения показанием датчика целевого значения. Это помогает избежать ситуаций, когда программа полностью перестает работать в случае наступления неожиданного события.

ПРАКТИКУМ 13.1

Программа *Gyroturn*, представленная в гл. 5, была предусмотрена четверть оборота робота TriBot, он двигался до тех пор, пока показание гироскопического датчика не достигнет 90°. Этот подход работает хорошо при медленном движении робота, и не очень хорошо, когда он двигается быстро. Усовершенствуй эту программу, используя показания гироскопического датчика для управления скоростью и заставляя робота двигаться быстрее, когда до окончания поворота еще далеко, и замедляя его по мере приближения к цели. Это позволит роботу TriBot быстро и точно повернуть на 90°, потратив на это одну-две секунды вместо нескольких секунд, которые тратятся при повороте на постоянной, медленной скорости. Предотврати слишком медленную скорость поворота робота, убедившись в том, что блок **Математика** (Math) всегда предоставляет некое минимальное значение.

СОВЕТ Выражение *10 + (Целевое значение — Показание датчика) × Коэффициент усиления* всегда будет равно как минимум 10, если целевое значение больше или равно показанию датчика (а значение коэффициента усиления положительно).

Таймеры можно выбрать из списка датчиков блоков **Ожидание** (Wait), **Переключатель** (Switch) и **Цикл** (Loop). Кроме того, ими можно управлять с помощью блока **Таймер** (Timer), который находится на вкладке палитры программирования с блоками датчиков. В блоке **Таймер** (Timer) (рис. 13.11) предусмотрены три режима: режим **Измерение** (Measure), считывающий текущее значение таймера (в секундах), режим **Сравнение** (Compare), с помощью которого сравнивается значение с пороговым и выходит результат в виде логического значения, а режим **Сброс** (Reset), сбрасывающий показание таймера на 0. Кроме того, в этом блоке предусмотрена возможность выбора одного из восьми таймеров.

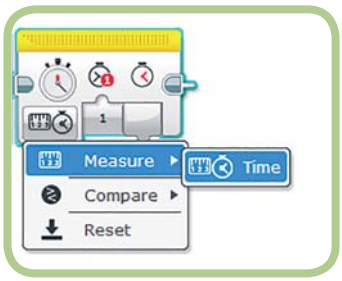


*Рис. 13.11. Блок **Таймер** (Timer)*

Программа DisplayTimer

Программа *DisplayTimer* сочетает в себе мощь блоков **Математика** (Math) и **Таймер** (Timer) для отображения отсчета времени на экране модуля EV3. Эта программа получает показание из блока **Таймер** (Timer), после чего оно отображается в типичном формате «минуты: секунды»; например, три секунды отображаются в виде 0:03, а две минуты и пятнадцать секунд — в виде 2:15. В программе используется блок **Цикл** (Loop), благодаря которому она продолжает работать до тех пор, пока ты ее не остановишь, и при каждом выполнении цикла в блоке **Таймер** (Timer) считывается значение таймера, и на экране отображается новое значение.

Обрати внимание на то, что блоком **Таймер** (Timer) считываются дробные значения, например 7.46 или 11.038, которые отображаются так же: (2:15.947). Однако чтобы не усложнять описание этой программы, в следующих примерах будут использованы целые числа.

Разделение показания таймера на минуты и секунды

В этой программе возьмем показание из блока **Таймер** (Timer) и разделим его на минуты и секунды. Например, если показание блока **Таймер** (Timer) составляет 127 секунд, мы хотим отобразить его как 2 минуты и 7 секунд. Для этого можно применить две простые формулы:

Секунды = Показание таймера % 60
Минуты = (Показание таймера — Секунды) / 60

Для того чтобы вычислить количество секунд (в диапазоне от 0 до 59), которое требуется отобразить, мы берем остаток от деления показания блока **Таймер** (Timer) на 60. Таким образом, при показании блока **Таймер** (Timer), равном 127, получаем 7, поскольку 127% 60 = 7.

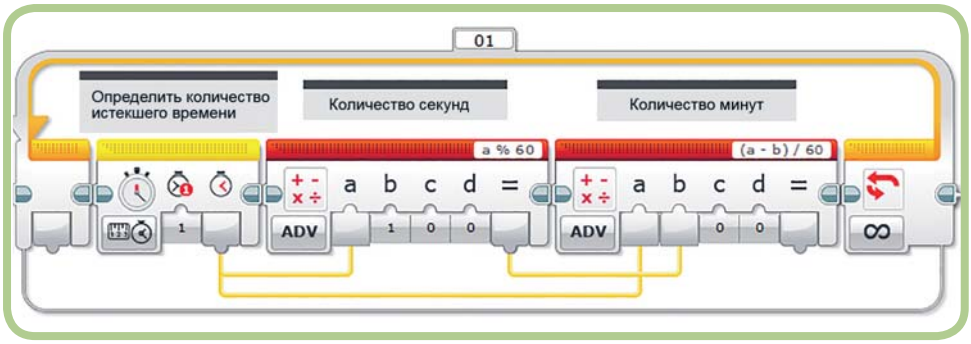


Рис. 13.12. *Преобразование количества истекшего времени в минуты и секунды*

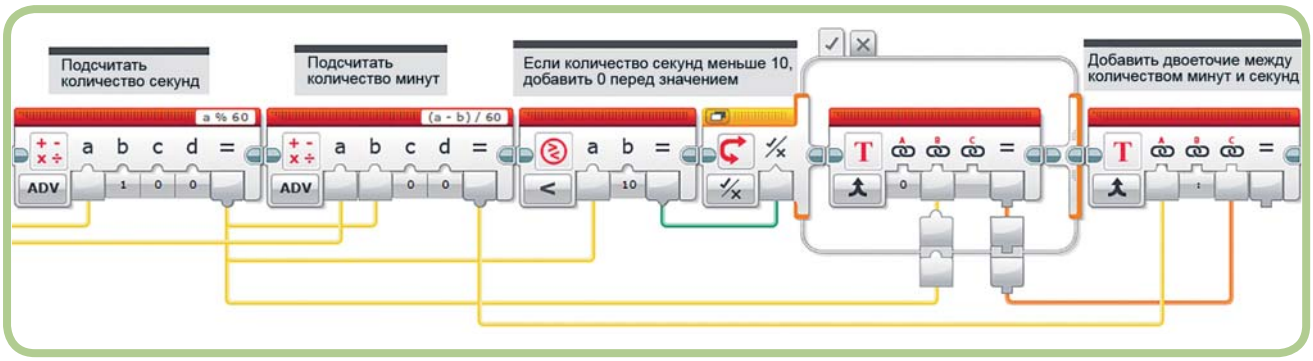


Рис. 13.13. Добавление 0, если количество секунд меньше 10

Вычтя количество секунд из показания блока **Таймер** (Timer), получим значение, кратное 60. В рассматриваемом примере: 127–7 = 120. Деление этого значения на 60 даст нам количество минут, поскольку в одной минуте 60 секунд.

На рис. 13.12 показана первая часть программы. Блок **Таймер** (Timer) определяет истекшее время и передает полученное значение двум блокам **Математика** (Math). В обоих блоках **Математика** (Math) применяется режим **Дополнения** (Advanced) для вычисления количества секунд и минут, которое требуется отобразить. Блок для вычисления количества секунд должен идти первым, поскольку его результат используется другим блоком **Математика** (Math).

Создание текста для отображения

На следующем этапе необходимо взять числовые значения, соответствующие количеству минут и секунд, и объединить их с помощью двоеточия (:) для создания текстового значения в формате «минуты: секунды». Можно отправить количество минут и секунд прямо в блок **Текст** (Text) для их объединения, однако существует одна проблема: если количество секунд меньше 10, результат окажется некорректным. Например, если количество минут равно 2, а количество секунд равно 7, то блок **Текст** (Text) сгенерирует значение 2:7 вместо 2:07. Прежде чем отправлять значения в блок **Текст** (Text), мы должны добавить 0 перед количеством секунд, если оно меньше 10.

На рис. 13.13 показан код, генерирующий текстовое значение в правильном формате, когда количество секунд меньше 10, а также два блока **Математика** (Math), которые выдают количество минут и секунд. Программой используется блок **Сравнение** (Compare), для того чтобы проверить, не превышает ли количество секунд значение 10. В случае «Истина» блока **Переключатель** (Switch) используется блок **Текст** (Text) для добавления 0 к количеству секунд и передачи результата из блока **Переключатель** (Switch) в другой блок **Текст** (Text), объединяющий его с количеством минут и двоеточием для получения отформатированного значения.

Последний блок **Текст** (Text) теперь выдает правильное значение — 2:07, когда количество минут равно 2, а количество секунд — 7.

Что происходит, когда количество секунд превышает 10? Используется случай «Ложь» блока **Переключатель** (Switch), и требуется передать значение последнему блоку **Текст** (Text) в неизмененном виде, поэтому можно просто добавить шину данных между вводом и выводом блока (рис. 13.14).

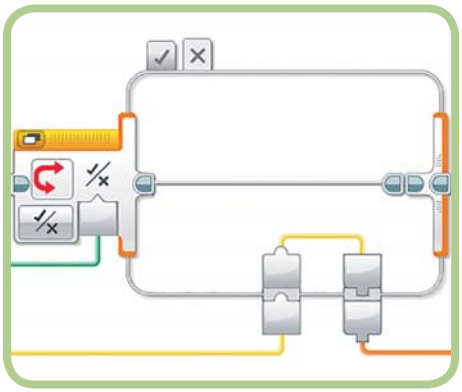


Рис. 13.14. Передача значения из блока ***Переключатель*** *(Switch) в виде текста*

Обрати внимание на то, что шина данных, подключенная к вводу блока **Переключатель** (Switch), содержит числовое значение из блока **Математика** (Math), что объясняет ее

желтый цвет. Выходная шина данных содержит текстовое значение (этот вывод был создан блоком **Текст** (Text), находящимся внутри случая «Истина»), что объясняет ее оранжевый цвет. При создании этой программы сначала обязательно заполни случай «Истина», чтобы выходная шина блока **Переключатель** (Switch) была оранжевого цвета, соответствующего текстовому значению. Если сначала заполнить случай «Ложь» (для передачи числового значения в следующий блок), будет создан вывод для желтой числовой шины данных, и тогда будет невозможно подключить к этому выводу шину данных от блока **Текст** (Text), находящегося в случае «Истина». Помни, что ты можешь передать числовое значение на текстовый ввод (при этом тип данных будет преобразован автоматически), однако передать текстовое значение на числовой ввод нельзя.

Заключительной частью программы является блок **Экран** (Display), отображающий результат выполнения последнего блока **Текст** (Text) на экране модуля EV3 (рис. 13.15). Для параметра **Строка** (Row) задано значение 4, поскольку воспринимать текст удобнее, когда он расположен ближе к середине экрана.

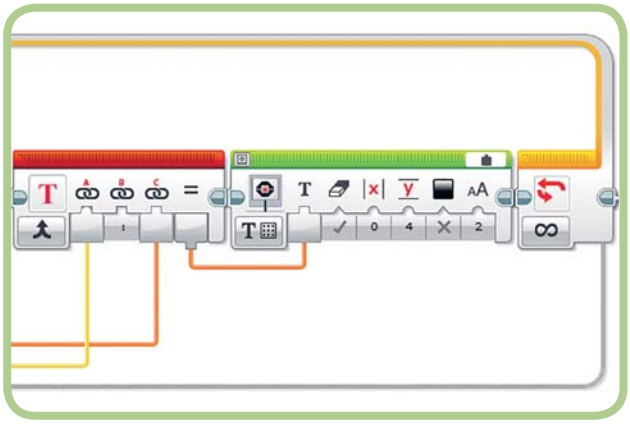


Рис. 13.15. Отображение значения времени

После запуска программы на экране должен отобразиться отсчет времени. Дай таймеру поработать хотя бы минуту, чтобы убедиться в корректном отображении количества секунд, когда оно меньше 10, когда оно больше 10 и когда количество минут меняется с 0 на 1. На экране отобразятся еще три цифры после десятичной точки, которые будут очень быстро меняться. О том, как скрыть или отбросить дробную часть, ты узнаешь в следующем разделе.

Блок «Округление»

Использование блока **Округление** (Round) (рис. 13.16) — это простой способ округления чисел. Четыре предусмотренных в нем режима соответствуют различным способам округления числа. Используя режим **До ближайшего** (To Nearest), можно округлить значение до ближайшего целого числа, с помощью режима **Округлить к большему** (Round Up), — до следующего целого числа, а с помощью режима **Округлить к меньшему** (Round Down) — до предыдущего целого числа. Эти три режима делают то же самое, что и функции round, ceil и floor блока **Математика** (Math) (доступные в режиме **Дополнения** (Advanced)).

*Рис. 13.16. Блок **Округление** (Round)*

В табл. 13.4 приведены результаты применения трех режимов к некоторым входным значениям. Обрати внимание на то, что если входное значение является целым числом, то все три режима возвращают значение без изменений. То, как режимы **Округлить к большему** (Round Up) и **Округлить к меньшему** (Round Down) работают с отрицательными числами, может показаться несколько странным, однако немного разобравшись с ними, ты увидишь, что все функционирует правильно; например, поскольку −4 больше −5, при использовании режима **Округлить к большему** (Round Up) −4,2 округляется до −4, а при применении режима **Округлить к меньшему** (Round Down) — до −5.

Табл. 13.4. Сравнение режимов блока **Округление** (Round)

Входное значение	До ближайшего	Округлить к большему	Округлить к меньшему
4.0	4	4	4
4.2	4	5	4
4.5	5	5	4
4.7	5	5	4
−4.2	−4	−4	−5
−4.5	−5	−4	−5
−4.7	−5	−4	−5

Режим **Отбросить дробную часть** (Truncate) блока **Округление** (Round) предусматривает дополнительный параметр **Число десятичных знаков** (Number of Decimals), позволяющий указать, сколько цифр требуется оставить после десятичной точки (рис. 13.17). Любые цифры, идущие после указанного тобой разряда, отбрасываются (округления при этом не происходит).

*Рис. 13.17. Режим **Отбросить дробную часть** блока **Округление***

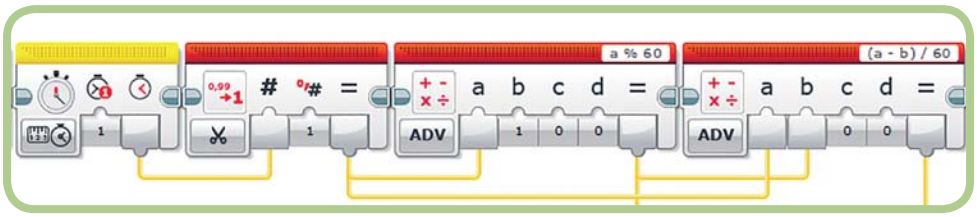


Рис. 13.18. Отбрасывание дробной части значения истекшего времени

Для отображения программой *DisplayTimer* только одного десятичного знака добавь блок **Округление** (Round) с выбранным режимом **Отбросить дробную часть** (Truncate) после блока **Таймер** (Timer). Не забудь подключить вывод блока **Округление** (Round) к вводу *a* обоих блоков **Математика** (Math) (рис. 13.18). Если тебе нужно отобразить целое число секунд, задай значение 0 для параметра **Число десятичных знаков** (Number of Decimals) в блоке **Округление** (Round).

ПРАКТИКУМ 13.2

Создай контейнер **TimeToText** из блоков, показанных на рис. 13.14. Этот контейнер должен принимать значение истекшего времени в качестве входного параметра, а в качестве выходного параметра выдавать отформатированное текстовое значение.

Блок «Случайное значение»

Среди блоков операций с данными содержится еще один блок, имеющий отношение к математике, — **Случайное значение** (Random) (рис. 13.19). На этом блоке изображена игральная кость, и, подобно ей, данный блок генерирует случайные числа, которые можно использовать для создания роботизированных игр или для добавления элемента случайности в поведение робота. Зачастую робот, который ведет себя несколько непредсказуемо, может казаться более интересным и самобытным.

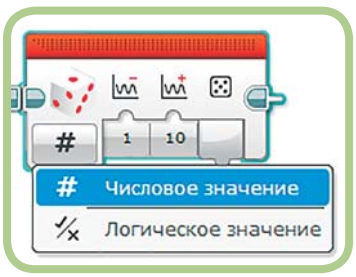


*Рис. 13.19. Блок **Случайное значение***

При использовании блока **Случайное значение** (Random) в режиме **Числовое значение** (Numeric) ты можешь задать значения параметров **Нижняя граница** (Lower Bound) и **Верхняя граница** (Upper Bound), чтобы блок сгенерировал целое число в заданном диапазоне. По умолчанию для этих параметров задаются числа 1 и 10, что дает выходное значение от 1 до 10 включительно. Ты можешь изменить диапазон в соответствии со своей программой; например, для создания виртуальной игральной кости задай значение 1 для параметра **Нижняя граница** (Lower Bound) и значение 6 для параметра **Верхняя граница** (Upper Bound).

ПРАКТИКУМ 13.3

С помощью контейнера **DisplayNumber** (из гл. 12) отображаются числа, содержащие до трех десятичных знаков. Добавь параметр, чтобы указать, до какого десятичного разряда требуется округлить значение перед его отображением. Режим **Отбросить дробную часть** (Truncate) блока **Округление** (Round) в данном случае не сработает, поскольку нам нужно округлить число, а не просто отбросить дробную часть. С помощью режима **До ближайшего** (To Nearest) — тоже, поскольку в этом режиме можно округлить значение только до ближайшего целого числа.

СОВЕТ Допустим, есть число 34,567, и его необходимо округлить до двух десятичных знаков, чтобы на экране модуля EV3 отображалось число 34,57. Вот один из способов решения этой задачи:

1. Умножь 34,567 на 10^2, чтобы получить 3456,7. Обрати внимание на то, что показатель степени соответствует количеству нужных нам десятичных знаков.
2. Округли значение до 3457.
3. Раздели 3457 на 10^2, чтобы получить 34,57.

*Рис. 13.20. Блок **Случайное значение** (Random) в режиме **Логическое значение** (Logic)*

В режиме **Логическое значение** (Logic) (рис. 13.20) в этом блоке генерируется случайное логическое значение (**Истина**

(True) или **Ложь** (False)), исходя из указанной степени вероятности. Значение параметра **Вероятность значения «Истина»** (Probability of True) в диапазоне от 0 до 100 определяет вероятность выпадения значения **Истина** (True). Например, значение 80 говорит о том, что существует 80%-ная вероятность того, что результатом будет **Истина** (True) и 20%-ная вероятность того, что значением будет **Ложь** (False).

Добавление случайного поворота в программу BumperBot

В этом разделе мы внесем небольшое изменение в программу *BumperBot*, чтобы сделать ее интереснее. Напомню, что при столкновении с препятствием робот TriBot откатывается назад и поворачивает в другом направлении. Робот не обязательно должен поворачивать на конкретное количество градусов; нам просто нужно, чтобы он двигался в другом направлении. Для задания количества градусов можно использовать блок **Случайное значение** (Random), чтобы сделать программу менее предсказуемой.

Ты можешь начать с копирования последней версии программы *BumperBot* из проекта *Chapter6* в проект *Chapter13*. На рис. 13.21 показана часть программы *BumperBot*, которую тебе нужно изменить. Этот код выполняется после нажатия кнопки датчика касания. С помощью первой группы блоков робот откатывается назад, а с помощью последнего блока **Рулевое управление** (Move Steering) поворачивается.

На данном этапе блок **Рулевое управление** (Move Steering) настроен на поворот на 225°. Чтобы сделать поворот менее предсказуемым, добавь блок **Случайное значение** (Random) перед блоком **Рулевое управление** (Move Steering). Затем задай диапазон значений, которые могут быть сгенерированы блоком **Случайное значение** (Random). В исходной программе использовалось значение 225°, что позволяло роботу повернуть чуть больше, чем на четверть оборота. Я буду использовать 200° в качестве значения параметра **Нижняя граница** (Lower Bound) и 2000° в качестве значения параметра **Верхняя граница** (Upper Bound). При таких значениях робот иногда будет быстро поворачивать и продолжать двигаться вперед, а иногда будет в течение некоторого времени крутиться на месте перед возобновлением своего

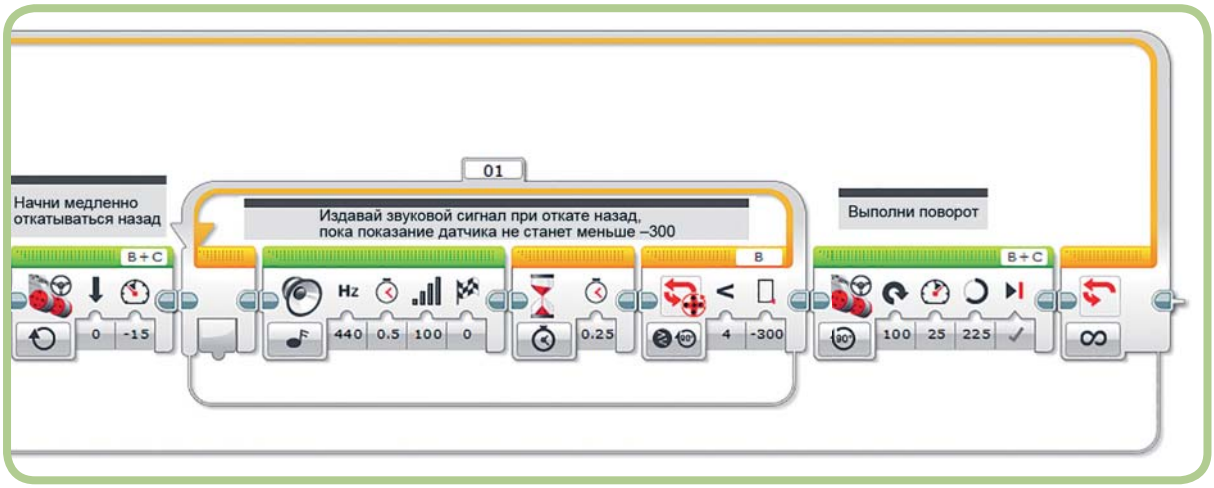


Рис. 13.21. Откат назад и поворот

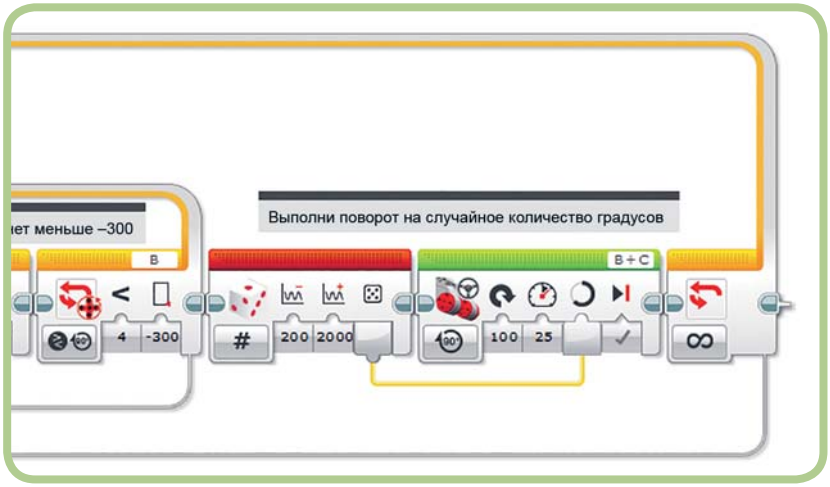


Рис. 13.22. Поворот на случайное количество градусов

путешествия по комнате. На рис. 13.22 показана эта часть программы с внесенными изменениями.

После запуска измененной таким образом программы робот TriBot должен варьировать количество градусов, на которое он поворачивает после столкновения с препятствием.

Блок «Логические операции»

Во многих из представленных до сих пор программ решения принимаются на основе одного условия, которое обычно сравнивает показание датчика с целевым значением, а полученный результат («Истина» или «Ложь») используется в блоке **Переключатель** (Switch) или **Цикл** (Loop). Другими словами, программы задают простые вопросы, например, «Нажата ли кнопка датчика касания?» или «Показание датчика цвета меньше 50?».

Блок **Логические операции** (Logic) позволяет объединить несколько условий, благодаря чему программа может принимать более сложные решения. При наличии этого блока программа может задавать такие вопросы: «Нажата ли кнопка датчика касания *и* превышает ли показание датчика цвета значение 50?» Ты можешь найти блок **Логические операции** (Logic), показанный на рис. 13.23, среди других имеющих отношение к математике блоков операций с данными.

*Рис. 13.23. Блок **Логические операции** (Logic)*

В блоке **Логические операции** (Logic) предусмотрены четыре режима: **И** (AND), **ИЛИ** (OR), **Исключающее ИЛИ** (XOR) и **Исключение** (NOT). Далее описан принцип работы каждого из них:

- **И** (AND) Результатом использования данного режима будет **Истина** (True) только в том случае, если истинны оба входных значения. Если одно их входных значений ложно, — результатом будет **Ложь** (False).
- **ИЛИ** (OR) Результатом использования данного режима будет **Истина** (True), если одно или оба входных значения истинны. Результатом будет **Ложь** (False), только если оба входных значения ложны.
- **Исключающее ИЛИ** (XOR) Этот режим похож на режим **ИЛИ** (OR) за исключением того, что результат является ложным, если оба входных значения истинны. Его действие похоже на то, как слово *или* используется в языке: если твоя мама говорит, что ты можешь съесть мороженое или конфету, это, вероятно, не означает, что тебе позволили съесть и то и другое; она ожидает, что ты выберешь что-то одно.
- **Исключение** (NOT) Этот режим предполагает использование только одного входного значения и генерирует противоположное значение. Если входное значение истинно, то выходное значение будет ложным, а если входное значение ложно, выходное значение будет истинным.

В табл. 13.5 перечислены все возможные входные значения и результат применения каждого из режимов. Такая таблица называется *таблицей истинности*. Обрати внимание на то, что результат использования режима **Исключение** (NOT) зависит только от входного значения *a*.

Табл. 13.5. Таблица истинности для блока **Логические операции**

Входное значение a	Входное значение b	ИЛИ (OR)	И (AND)	Исключающее ИЛИ (XOR)	Исключение (NOT)
Ложь	Ложь	Ложь	Ложь	Ложь	Истина
Ложь	Истина	Истина	Ложь	Истина	Истина
Истина	Ложь	Истина	Ложь	Истина	Ложь
Истина	Истина	Истина	Истина	Ложь	Ложь

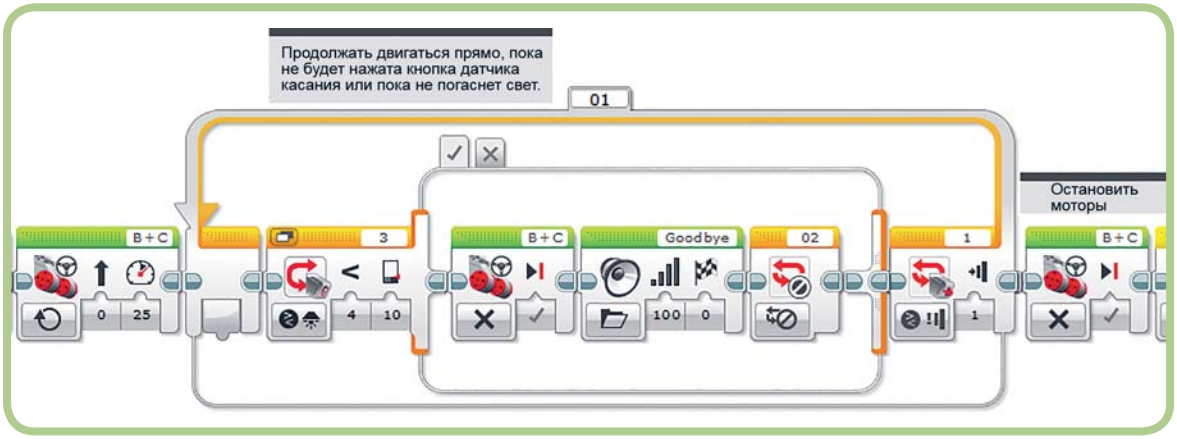


Рис. 13.24. Движение робота вперед вплоть до нажатия кнопки датчика касания

Добавление логики в программу BumperBot

В этом разделе изменим программу *BumperBot*, добавив в нее блок **Логические операции** (Logic). Напомню, что эта программа заставляет робота TriBot двигаться вперед до столкновения с каким-нибудь препятствием. Что, если тебе требуется ограничить период времени, в течение которого робот движется вперед, например, чтобы он не уезжал слишком далеко? Тебе предстоит изменить эту программу так, чтобы робот TriBot останавливался и поворачивал при столкновении с препятствием *или* после 20 секунд движения вперед.

На рис. 13.24 показан код, который заставляет робота двигаться вперед. С помощью блока **Рулевое управление** (Move Steering) робот начинает движение и продолжает движение до тех пор, пока блок **Цикл** (Loop) не завершится в результате нажатия кнопки датчика касания.

Как узнать, что робот двигался более 20 секунд? Ты можешь использовать блок **Таймер** (Timer) для сброса показания таймера перед запуском робота, а затем использовать другой блок **Таймер** (Timer), который сообщит, когда истекут 20 секунд.

Блок **Цикл** (Loop) можно настроить на считывание показания датчика касания или таймера, но не их обоих. Чтобы выйти из цикла при истинности одного из условий, тебе необходимо проверить оба условия, объединить результаты с помощью блока **Логические операции** (Logic) и использовать полученный от него результат для повтора цикла или выхода из него. Рассмотрим процесс внесения этих изменений шаг за шагом.

1. Добавь блок Таймер (Timer) слева от блока Рулевое управление (Move Steering) и выбери режим Сброс (Reset) (рис. 13.25).

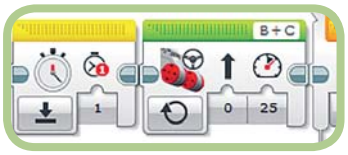


*Рис. 13.25. Расположение блока **Таймер** (Timer)*

2. Добавь блок **Таймер** (Timer) после блока **Переключатель** (Switch). Выбери режим **Сравнение** (Compare) ⇒ **Время** (Time) и задай значение **20** для параметра **Пороговое значение** (Threshold parameter).

3. Добавь блок **Датчик касания** (Touch Sensor) после блока **Таймер** (Timer) и выбери режим **Сравнение** (Compare) ⇒ **Состояние** (State).

4. Добавь блок **Логические операции** (Logic) справа от блока **Датчик касания** (Touch Sensor) и выбери режим **ИЛИ** (OR).

5. Добавь шину данных между выводом **Результат сравнения** (Compare Result) блока **Датчик касания** (Touch Sensor) и вводом *a* блока **Логические операции** (Logic).

6. Добавь шину данных между выводом **Результат сравнения** (Compare Result) блока **Таймер** (Timer) и вводом *b* блока **Логические операции** (Logic).

7. Выдели блок **Цикл** (Loop) и измени режим на **Логическое значение** (Logic).

8. Добавь шину данных между выводом **Результат** (Result) блока **Логические операции** (Logic) и вводом **Пока не будет Истина** (Until True) блока **Цикл** (Loop).

На рис. 13.26 показана программа после внесения этих изменений.

Теперь, после запуска программы, робот TriBot должен двигаться вперед в течение максимум 20 секунд. Если за это время он не столкнется с препятствием, он должен повернуть и начать двигаться в другом направлении.

Рис. 13.26. Выход из цикла по истечении 20 секунд или при нажатии кнопки датчика касания

Блок «Интервал»

Последним блоком, имеющим отношение к математике, является блок **Интервал** (Range), с помощью которого можно определить, находится ли число внутри или вне заданного диапазона чисел. Этот блок имеет три параметра: **Тестовое**

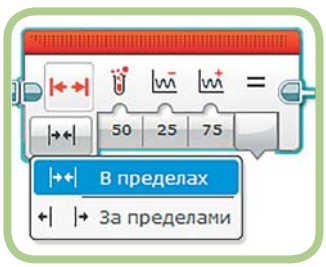


*Рис. 13.27. Блок **Интервал** (Range)*

значение (Test Value), которое обычно передается с помощью шины данных, **Нижняя граница** (Lower Bound) и **Верхняя граница** (Upper Bound) рассматриваемого тебя диапазона.

В блоке **Интервал** (Range) предусмотрены два режима: **В пределах** (Inside) и **За пределами** (Outside) (рис. 13.27). В режиме **В пределах** (Inside) в блоке появляется вопрос: «Находится ли тестовое значение внутри диапазона (между нижним и верхним пределами)?» В режиме **За пределами** (Outside) в блоке возникает следующий вопрос: «Находится ли тестовое значение за пределами диапазона (меньше нижнего предела или больше верхнего)?» Если тестовое значение равно верхнему или нижнему пределу, то оно считается находящимся внутри диапазона.

Программа TagAlong

Программа*TagAlong* использует блок **Интервал** (Range) для того, чтобы робот TriBot следовал за тобой по комнате, держась на небольшом расстоянии позади. Эта простая программа перемещает робота только вперед или назад. Он не будет следовать за тобой, если ты будешь двигаться в сторону. В программе применяются блок **Инфракрасный датчик** (Infrared Sensor) и блок **Интервал** (Range), чтобы определить, находится ли робот в пределах заданного диапазона. Если робот находится вне допустимого диапазона, с помощью блока **Рулевое управление** (Move Steering) он переместится вперед или назад, используя показание инфракрасного датчика для настройки параметра **Мощность** (Power). Если робот находится в пределах требуемого диапазона, моторы останавливаются.

Данная программа показана на рис. 13.28. Рассмотрим все блоки по очереди, чтобы разобраться с принципом работы программы.

Блок **Цикл** (Loop) обеспечивает работу программы до тех пор, пока ты ее не остановишь.

Блок **Инфракрасный датчик** (Infrared Sensor) использует режим **Измерение** (Measure) ⇒ **Приближение** (Proximity) для определения расстояния до объекта, находящегося перед роботом (то есть до тебя).

Блок **Интервал** (Range) проверяет, находится ли показание инфракрасного датчика вне диапазона. Ограничим диапазон значениями 40 и 60, что позволит роботу находиться достаточно близко. Результатом будет **Истина** (True), если показание датчика меньше 40 или больше 60, в противном случае результатом будет **Ложь** (False).

Блок **Переключатель** (Switch) случай «Истина» используется, когда робот оказывается за пределами допустимого диапазона и его необходимо передвинуть. Значение параметра **Мощность** (Power) блока **Рулевое управление** (Move Steering) рассчитывается путем вычитания 50 из показания инфракрасного датчика. Число 50 используется потому, что оно представляет собой среднее значение между 40 и 60. Таким образом, если показание датчика превышает 60, значение параметра **Мощность** (Power) оказывается равным как минимум 10, и робот перемещается вперед. Если показание датчика меньше 40, значение параметра **Мощность** (Power) оказывается равным –10 или меньше, и робот перемещается назад.

Случай «Ложь» блока **Переключатель** (Switch) применяется, когда робот находится в пределах допустимого диапазона, а показание датчика составляет от 40 до 60. При этом блок **Рулевое управление** (Move Steering) используется для остановки моторов.

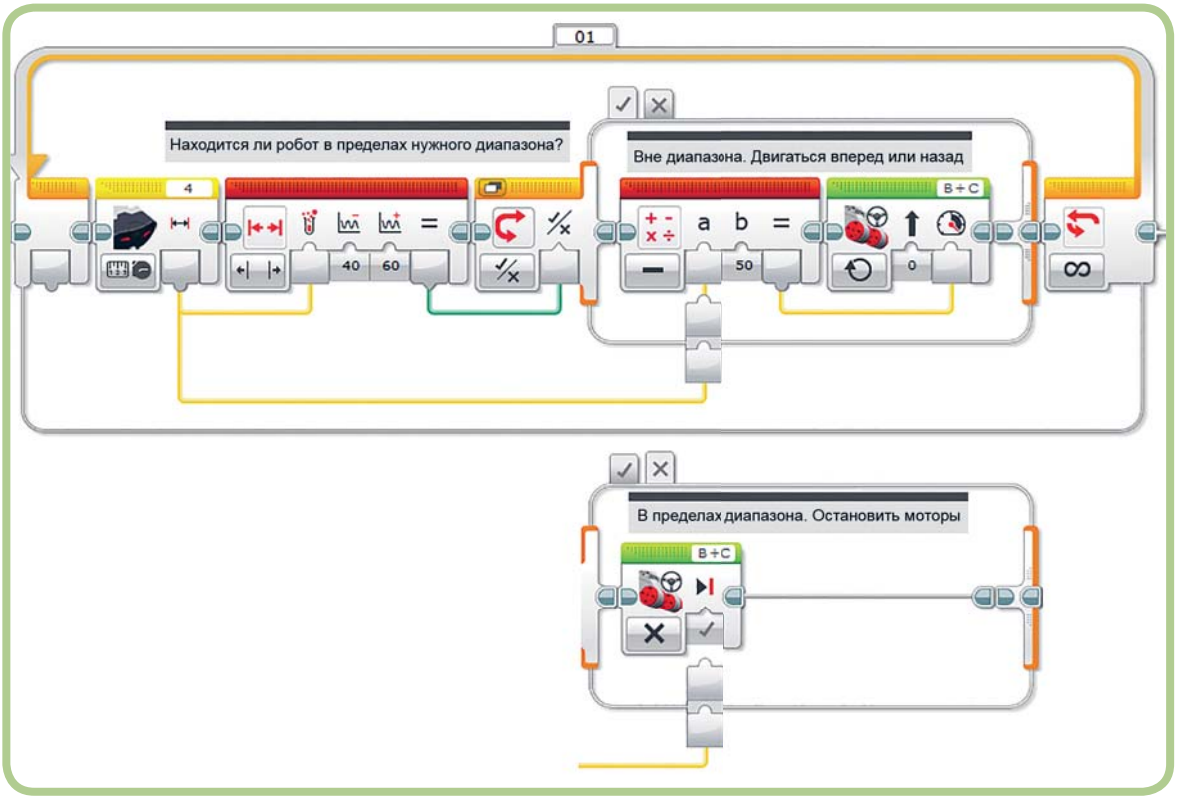


Рис. 13.28. Программа TagAlong

ПРИМЕЧАНИЕ В случае применения ультразвукового датчика используй блок **Ультразвуковой датчик** (Ultrasonic Sensor) в режиме **Измерение** (Measure) ⇒ **Расстояние в дюймах** (Distance Inches). В качестве границ диапазона в блоке **Интервал** (Range) используй значения 12 и 24, а для параметра *b* блока **Математика** (Math) задай значение **18** (среднее между 12 и 24).

После запуска этой программы робот TriBot должен двигаться вперед и назад при твоем отдалении и приближении к нему.

Программа GyroPointer

Программа *GyroPointer* представляет собой вариацию программы *TagAlong*, благодаря которой робот TriBot указывает в одном и том же направлении во время его вращения на вращающейся платформе. Если ты используешь домашнюю версию набора EV3 и у тебя нет гироскопического датчика, прочитай этот раздел и попробуй применить описанные в нем концепции при выполнении практикума 13.4.

Для тестирования этой программы помести робота TriBot на поворотный стол (вращающийся стул или другую вращающуюся поверхность) и медленно поворачивай этот стол во время работы программы. При этом робот будет поворачиваться, чтобы продолжать указывать в заданном направлении.

Эта программа показана на рис. 13.29. Ее основная структура аналогична структуре программы *TagAlong* за исключением некоторых изменений:

- Вместо блока **Инфракрасный датчик** (Infrared Sensor) используется блок **Гироскопический датчик** (Gyro Sensor).
- Параметры блока **Интервал** (Range) позволяют убедиться в том, что показания датчика находятся в диапазоне от –10 до 10. В начале программы показание датчика будет равно 0, при этом диапазоне робот не будет слишком сильно отклоняться от исходного направления.
- Вывод блока **Математика** (Math) передается на ввод **Рулевое управление** (Steering) блока **Рулевое управление** (Move Steering), а не на ввод **Мощность** (Power). Чтобы робот вращался на месте, этот параметр должен быть равен –100 или 100; в противном случае робот будет двигаться вперед и поворачивать.
- В блоке **Математика** (Math) показание датчика умножается на –10. Если показание датчика меньше –10, результат превысит значение 100, но блок **Рулевое управление** (Move Steering) обработает его как 100. Если показание датчика превышает 10, результат будет меньше –100, но блок **Рулевое управление** (Move Steering) обработает его как –100.

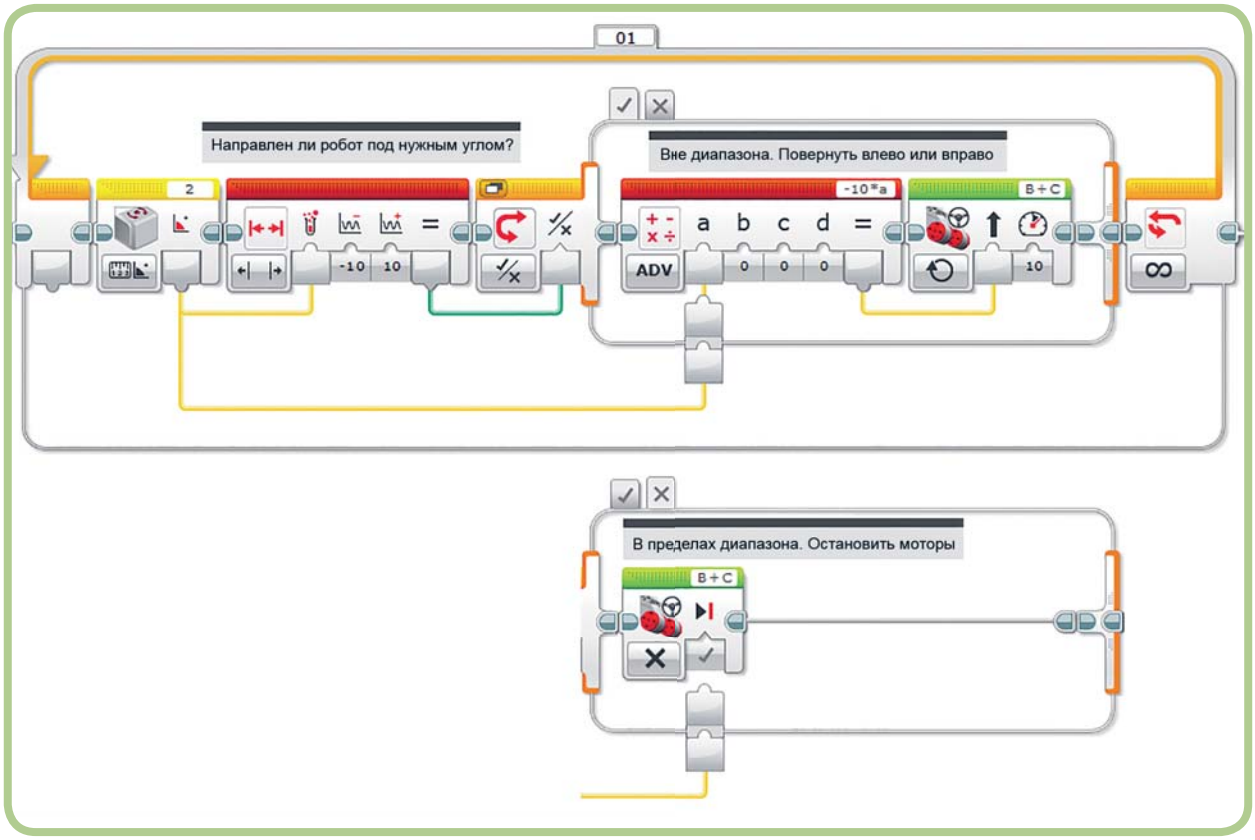


Рис. 13.29. Программа GyroPointer

После запуска этой программы робот будет оставаться неподвижным, пока ты не повернешь поворотный стол по крайней мере на 10° (в любом направлении). Затем робот начнет поворачиваться для сохранения изначально заданного направления. При вращении поворотного стола в разных направлениях робот TriBot должен корректировать свое положение, чтобы указывать в одном и том же направлении.

ПРАКТИКУМ 13.4

При работе с домашней версией конструктора EV3 используй инфракрасный датчик и удаленный инфракрасный маяк для создания программы *RemotePointer*, которая работает так же, как программа *GyroPointer*. При использовании этой программы робот TriBot должен поворачиваться так, чтобы он все время указывал в направлении инфракрасного маяка, используя режим **Направление маяка** (Beacon Heading) для управления блоком **Рулевое управление** (Move Steering).

Дальнейшее исследование

Попробуй выполнить следующие действия, чтобы попрактиковаться в использовании блоков, имеющих отношение к математике:

1. Напиши программу *CountDown*, с помощью которой на экране модуля EV3 отобразится обратный отсчет времени, начиная с двух минут. После окончания отсчета должен произойти выход из блока **Цикл** (Loop).

2. Объедини программы *TagAlong* и *RemotePointer* для создания программы, которая будет поддерживать необходимое расстояние и направление робота. В результате получится программа, с которой робот будет следовать по комнате за удаленным инфракрасным маяком.

Используй блок **Случайное значение** (Random) для создания программы *MagicEightBall*. Когда ты задаешь вопрос и запускаешь робота (например, нажатием кнопки датчика касания), робот должен выбирать случайный ответ среди нескольких возможных вариантов. Применяй блок **Экран** (Display) для отображения ответа на экране модуля. Кроме того, ты можешь использовать инструмент **Редактор звука** (Sound Editor) для записи собственных ответов и блок **Звук** (Sound) для их воспроизведения.

Для выполнения данного задания требуются некоторые знания по тригонометрии; ты можешь пропустить его, если пока не разбираешься в этой области математики. Функция синуса начинается с 0 и колеблется между 1 и –1, поэтому ее график напоминает змеящуюся кривую. Ты можешь заставить робота следовать по извилистой, змееподобной траектории, используя функцию синуса для управления движением робота. Создай программу *Slither*, применив таймер и блок **Математика** (Math), в котором функция синуса используется для управления параметром **Рулевое управление** (Steering) в блоке **Рулевое управление** (Move Steering). Подсказка: результат использования функции sin (Истекшее время) не будет очень интересным, поскольку для перехода между крайними значениями –1 и 1 потребуется 6 минут (360 секунд). Однако если умножить значение истекшего времени на 10, весь диапазон будет преодолен за 36 секунд, а если умножить результат на 50, значение параметра **Рулевое управление** (Steering) будет колебаться между 50 и –50.

Заключение

Из этой главы ты узнал об использовании блоков, которые работают с числовыми и логическими значениями. Режим **Дополнения** (Advanced) блока **Математика** (Math) позволяет вычислять сложные уравнения, благодаря чему удалось усовершенствовать программу *LineFollower*, добавив в нее пропорциональный регулятор. Ты также узнал об операторе деления с остатком и посмотрел на него в действии в контейнере **DisplayNumberNextLine** и в программе *DisplayTimer*.

С помощью блока **Логические операции** (Logic) можно создавать программы, способные принимать сложные решения, например с использованием разных комбинаций входных данных от нескольких датчиков. Применяя блок **Интервал** (Range), можно легко проверить, находится ли значение в заданном диапазоне. В этой главе также был рассмотрен блок **Случайное значение** (Random), который вносит в программу элемент непредсказуемости и помогает сделать робота более самобытным.

14

Индикаторы, кнопки и экран модуля EV3

Модуль EV3 имеет пять кнопок и экран, который ты можешь использовать для взаимодействия со своими программами так же, как ты используешь клавиатуру для взаимодействия с компьютером. Эти кнопки подсвечиваются расположенным вокруг них индикатором состояния, и в этой главе ты научишься управлять подсветкой с помощью блока **Индикатор состояния модуля** (Brick Status Light). Ты также узнаешь о некоторых новых функциях блока **Экран** (Display), которые обеспечивают бóльший контроль над экраном модуля EV3.

Кнопки модуля EV3

Используй пять больших кнопок на передней панели модуля EV3 (рис. 14.1) для управления своей программой. Например, можно заставить программу дождаться нажатия кнопки или выбрать действие в зависимости от того, какая кнопка была нажата. Как и в случае с датчиком касания, твои программы могут определить, когда кнопка нажата, отпущена или щелкнута (нажата и сразу отпущена). Кнопка «Назад» не может быть использована программой: нажатие этой кнопки во время выполнения программы прервет ее работу.

Блоки **Ожидание** (Wait), **Переключатель** (Switch) и **Цикл** (Loop) предусматривают режим **Кнопки управления модулем** (Brick Buttons) (рис. 14.2). Кроме того, ты можешь работать с кнопками модуля, используя блок **Кнопки управления модулем** (Brick Buttons), который похож на другие блоки датчиков. У каждой кнопки есть три режима: **Сравнение** (Compare), **Измерение** (Measure) и **Изменить** (Change).

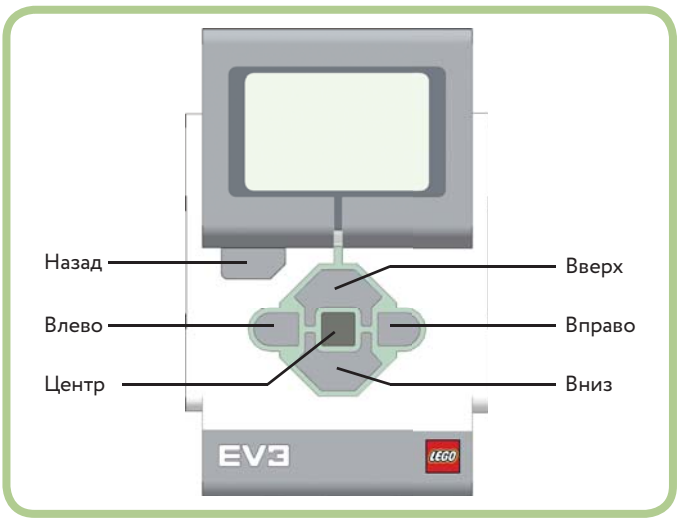


Рис. 14.1. *Кнопки модуля EV3*

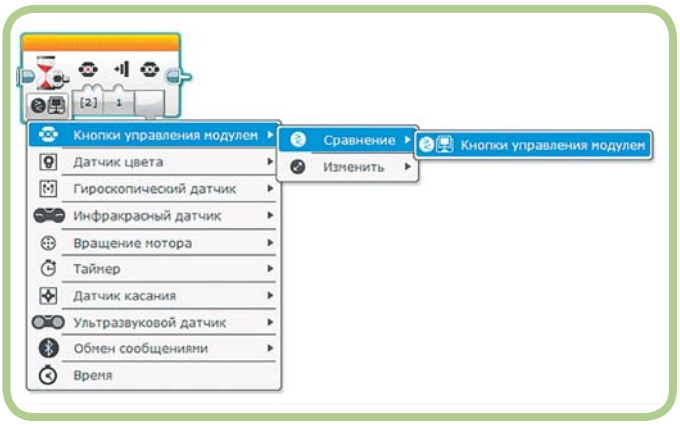


Рис. 14.2. Выбор режима ***Кнопки управления модулем*** *(Brick Buttons)*

С помощью режима **Сравнение** (Compare) можно проверить состояние одной или нескольких кнопок (нажата, отпущена или щелкнута). На рис. 14.3 показан раскрывающийся список выбора кнопок с номерами, соответствующими

каждой из них, — *идентификаторами кнопок*. Если ты выберешь больше одной кнопки, тест будет пройден, когда любая из этих кнопок окажется в нужном состоянии. Когда для блока **Кнопки управления модулем** (Brick Buttons) выбран режим **Сравнение** (Compare) (рис. 14.3), в нем генерируются два выходных значения: логическое значение, показывающее, находится ли одна из выбранных кнопок в указанном состоянии, и числовое значение, содержащее идентификатор соответствующей кнопки.

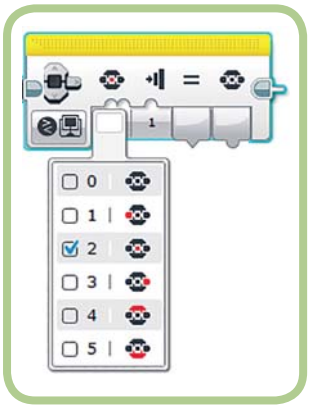


*Рис. 14.3. Выбор кнопок в режиме **Сравнение** (Compare)*

Благодаря режиму **Измерение** (Measure) можно определить, какая кнопка нажата в настоящий момент. Режим **Изменить** (Change) поддерживается только в блоке **Ожидание** (Wait) и позволяет ему дождаться изменения состояния любой из кнопок.

ПРИМЕЧАНИЕ **Если одновременно нажато несколько кнопок, существует вероятность, что датчик распознает нажатие только одной из них.**

Программа PowerSetting

Ранее (см. гл. 11) ты узнал о том, что программу *WallFollower* можно улучшить, добавив переменную для управления параметром **Мощность** (Power), используемым во всех семи блоках **Рулевое управление** (Move Steering). Это облегчает корректировку значения, поскольку этот параметр требуется изменить только в одном месте. Ты можешь управлять им с помощью кнопок модуля для еще большего удобства при тестировании различных значений. В этом разделе мы рассмотрим данный подход на примере программы *PowerSetting*. Внеся лишь несколько простых изменений, ты сможешь использовать этот код всегда, когда тебе потребуется задать значение с помощью кнопок модуля в начале программы.

В программе *PowerSetting* используется переменная *Power*, предназначенная для сохранения текущего значения и отображения его на экране модуля EV3. Нажатие кнопки «Вправо» увеличивает значение на единицу, а нажатие кнопки «Влево» на единицу уменьшает его. Нажатие кнопки «Центр» задает текущее значение. В листинге 14.1 приведен псевдокод для этой программы.

Листинг 14.1. Программа PowerSetting

```
set Power to 50
begin loop
display the current value
   if the Right button is bumped then
      Power = Power + 1
   end if
   if the Left button is bumped then
      Power = Power – 1
   end if
loop until the Center button is bumped
```

Для некоторых из этих строк псевдокода надо использовать несколько блоков программирования. Например, строка `Power = Power + 1` — это краткая запись команды: «Возьми текущее значение переменной Power, прибавь к нему единицу и сохрани результат в переменной Power», для выполнения которой требуются три блока программирования.

Начальное значение переменной и цикл

Первым действием программы является использование блока **Переменная** (Variable) для задания начального значения переменной Power, за которым следует блок **Цикл** (Loop), содержащий остальную часть программы (рис. 14.4). В качестве начального значения переменной Power задано 50, что соответствует среднему значению диапазона мощности от 0 до 100. В блоке **Цикл** (Loop) используется режим **Кнопки управления модулем** (Brick Buttons) ⇒ **Сравнение** (Compare), что обеспечивает его повторение вплоть до нажатия кнопки «Центр».

Рис. 14.4. Инициализация переменной Power и настройка параметров цикла

Настройка цикла на ожидание щелчка кнопкой, а не просто ее нажатия, часто предпочтительнее, поскольку цикл не перейдет к следующей части программы до тех пор, пока кнопка будет нажата *и* отпущена. Таким образом, кнопки

возвращаются в свое нормальное состояние (отпущена), с вероятностью возникновения проблем при использовании кнопок в следующей части программы.

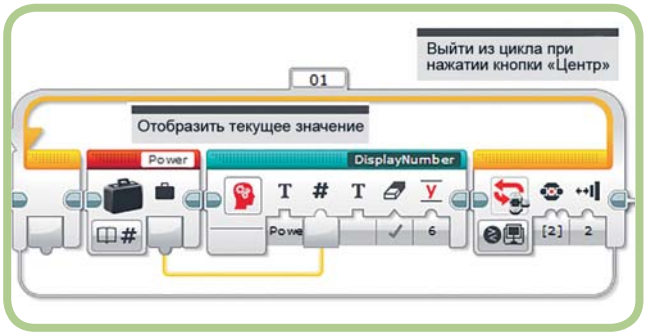


Рис. 14.5. *Отображение значения с меткой*

Отображение текущего значения переменной

При каждом выполнении цикла программа считывает и отображает текущее значение с помощью блока **Переменная** (Variable) и контейнера **DisplayNumber** (созданного в гл. 12), как показано на рис. 14.5. Для параметра **Строка** (Row) блока **DisplayNumber** задано значение 6, чтобы значение переменной отображалось ближе к центру экрана модуля EV3. Для параметра **Метка** (Label) задано значение “Power:”.

Корректировка значения переменной Power

При отображении текущего значения переменной Power ты можешь скорректировать его, используя кнопки модуля EV3 «Влево» и «Вправо». При использовании кода, показанного на рис. 14.6, необходима кнопка «Вправо». Щелчок правой

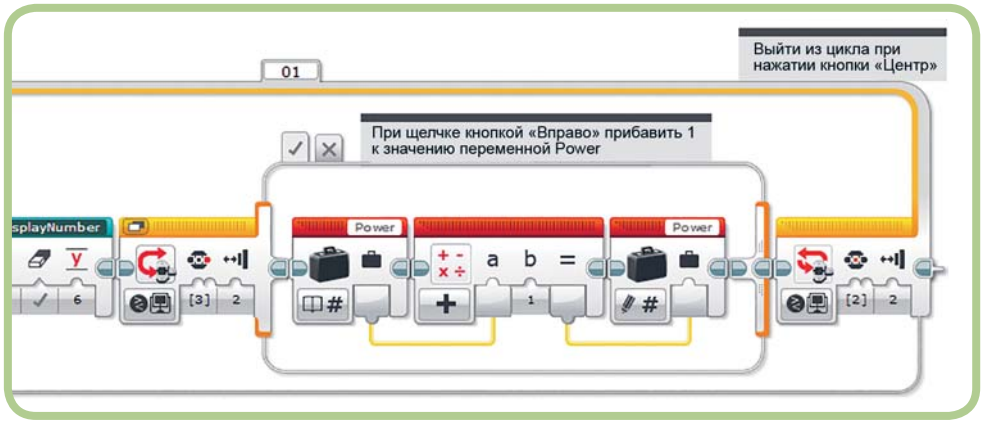


Рис. 14.6. Прибавление единицы к значению переменной Power при щелчке кнопкой «Вправо»

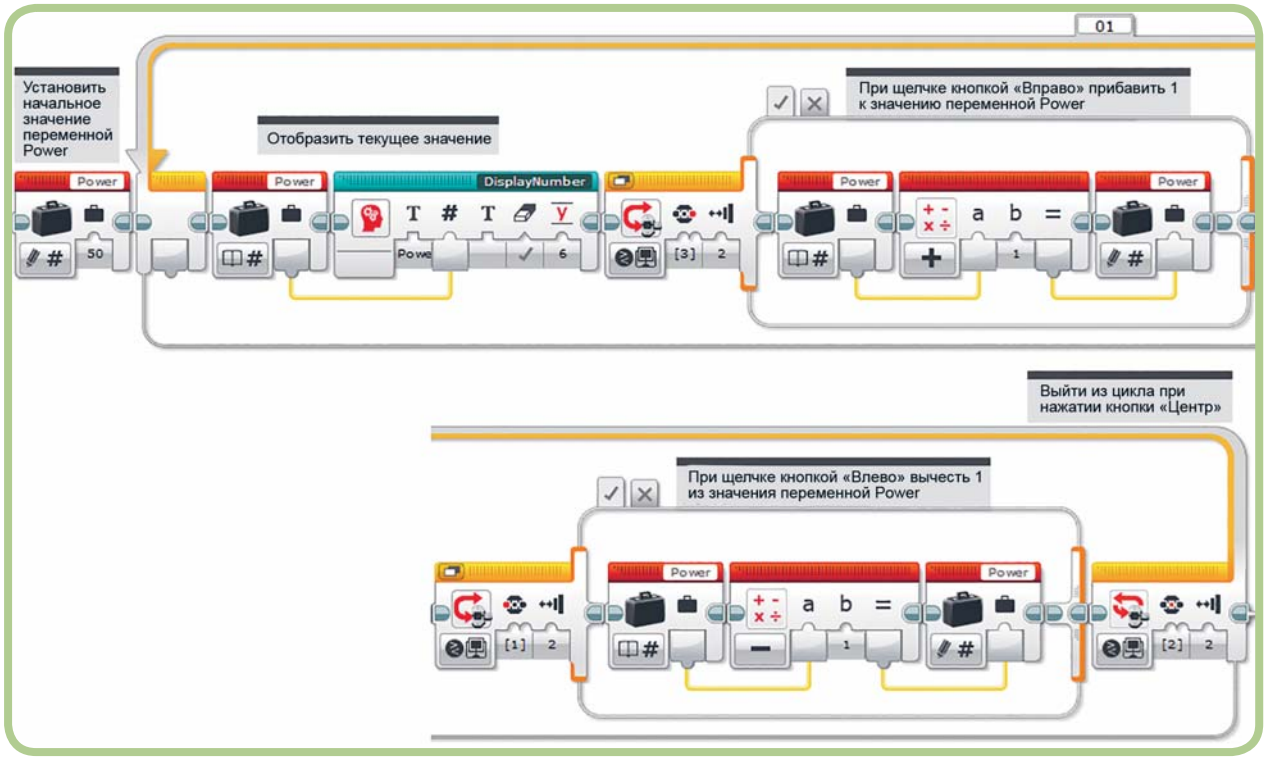


Рис. 14.7. Готовая программа PowerSetting

кнопкой запускает выполнение случая «Истина» блока **Переключатель** (Switch), который прибавляет единицу к значению переменной Power, используя блоки **Математика** (Math) и **Переменная** (Variable). Случай «Ложь» не содержит никаких блоков, поскольку, если щелчка кнопкой не было, в программе не должны совершаться никакие действия.

Код для работы с кнопкой «Влево» почти идентичен этому за исключением того, что при щелчке единица вычитается из значения переменной Power. На рис. 14.7 показана вся программа после добавления новых блоков.

Тестирование программы

После запуска эта программа должна сначала отобразить на экране модуля «Power: 50». Нажми кнопку «Вправо» и «Влево», чтобы изменить это значение. Нажми кнопку «Центр», чтобы завершить выполнение программы.

После завершения программы для переменной Power будет задано выбранное значение. Ты можешь поместить блоки этой программы в начало другой, более крупной программы для задания значения переменной. После добавления дополнительных блоков, следующих за блоком **Цикл** (Loop), по нажатию кнопки «Центр» будет запускаться оставшаяся часть программы (вместо прерывания).

Более быстрый способ изменения значения переменной

Когда ты нажимаешь и отпускаешь кнопку, программа *PowerSetting* изменяет значение переменной лишь на единицу, поэтому для внесения значительных изменений тебе может потребоваться некоторое время. Как можно ускорить этот процесс?

В настоящий момент два блока **Переключатель** (Switch) в цикле настроены на щелчок кнопки, это означает, что для изменения значения переменной кнопку требуется нажать и отпустить. Программа будет реагировать быстрее, если просто проверять, не нажата ли кнопка, вместо того, чтобы ждать, пока она будет нажата и отпущена. Что произойдет, если в обоих блоках **Переключатель** (Switch) для параметра **Состояние** (State) выбрать вариант **Нажатие** (Pressed), как показано на рис. 14.8?

Значение, безусловно, меняется быстрее, однако теперь это происходит слишком быстро, поскольку цикл повторяется несколько раз, даже если кнопка нажата лишь на мгновение. Чтобы сделать программу более удобной, тебе необходимо немного замедлить выполнение цикла, добавив в конец его тела блок **Ожидание** (Wait) в режиме **Время** (Time) (рис. 14.9). Ожидание в течение 0,2 секунды обеспечивает хороший баланс между скоростью изменения значения переменной и возможностью задать необходимое значение. Поэкспериментируй с разными значениями, чтобы определить наиболее подходящее.

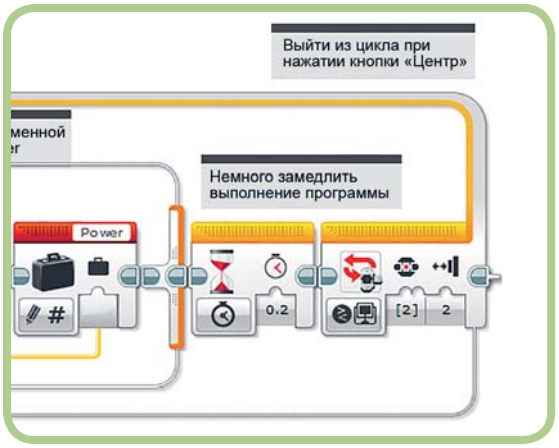


*Рис. 14.9. Добавление блока **Ожидание** (Wait) в режиме **Время** (Time) в конец тела цикла*

Когда программа *PowerSetting* будет работать так, как задумано, ты сможешь применить ее код в программе *WallFollower*. Для того чтобы легче применить его повторно, сначала преобразуй всю программу *PowerSetting* в контейнер «Мой блок». Затем добавь новый блок в начало программы *WallFollower* и используй переменную Power для управления блоками **Рулевое управление** (Move Steering).

ПРАКТИКУМ 14.1

Для еще большего удобства организации процесса внесения значительных изменений в значение переменной усовершенствуй программу *PowerSetting* так, чтобы можно было использовать кнопку «Вверх» для прибавления 10 к значению переменной и кнопку «Вниз», для вычитания из него 10. Какое состояние кнопки в этом случае использовать уместнее: «нажата» или «щелкнута»?

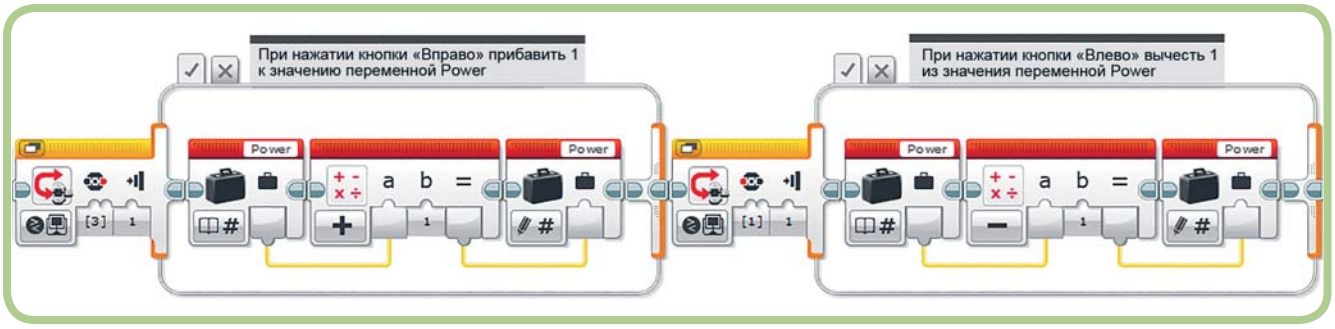


Рис. 14.8. Изменение состояния кнопки с «щелкнута» на «нажата»

Индикатор состояния модуля

Индикатор состояния модуля подсвечивает область вокруг кнопок модуля EV3. Когда модуль включен, этот индикатор непрерывно горит зеленым цветом, а во время выполнения программы мигает зеленым цветом, если программа не предусматривает изменения его поведения. С помощью блока **Индикатор состояния модуля** (Brick Status Light), который находится на вкладке с блоками действий палитры программирования, можно управлять поведением подсветки с помощью своих программ (рис. 14.10).

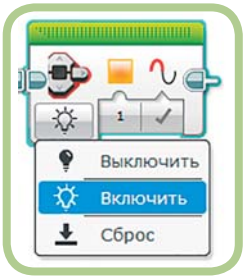


*Рис. 14.10. Блок **Индикатор состояния модуля** (Brick Status Light)*

В блоке **Индикатор состояния модуля** (Brick Status Light) предусмотрено три режима: **Выключить** (Off), **Включить** (On) и **Сброс** (Reset). При режиме **Выключить** (Off) отключается подсветка. При режиме **Включить** (On) включается подсветка и можно выбрать ее цвет (зеленый, оранжевый или красный), а также указать, должен ли свет гореть непрерывно или мигать. При режиме **Сброс** (Reset) включается мигающая зеленая подсветка по умолчанию. В данном случае принцип ее действия несколько отличается от того, который обеспечивается параметром **Импульсный** (Pulse).

Подсветку можно использовать при отладке программы. Например, подсветку можно применять для визуального представления процесса работы программы, зеленый цвет может означать, что все идет хорошо, а красный — возникла проблема. Кроме того, с ее помощью можно определить выполняемую в данный момент часть программы, а также узнать, истинно ли некоторое условие, не влияя на нормальную работу программы.

Программа ColorCopy

С помощью программы *ColorCopy* изменяется цвет подсветки индикатора в соответствии с цветом, обнаруженным датчиком цвета. Если перед датчиком находится красный объект, цвет подсветки становится красным; если датчик распознает зеленый цвет, цвет подсветки становится зеленым; а если он обнаруживает желтый цвет, цвет подсветки становится оранжевым (датчик цвета не распознает оранжевый цвет, поэтому выбирается наиболее близкий, т. е. желтый цвет). При распознавании другого цвета подсветка отключается.

На рис. 14.11 показана готовая программа. В блоке **Переключатель** (Switch) используется режим **Датчик цвета** (Color Sensor) ⇒ **Измерение** (Measure) ⇒ **Цвет** (Color) для определения обнаруженного цвета. Первые три случая соответствуют красному, зеленому и желтому цветам и включают соответствующую подсветку. Параметр **Импульсный** (Pulse) отключен, чтобы свет горел непрерывно, пока датчик фиксирует один и тот же цвет. Нижний случай — значение по умолчанию, при котором подсветка отключается, если датчиком обнаружен любой цвет, кроме красного, зеленого или желтого.

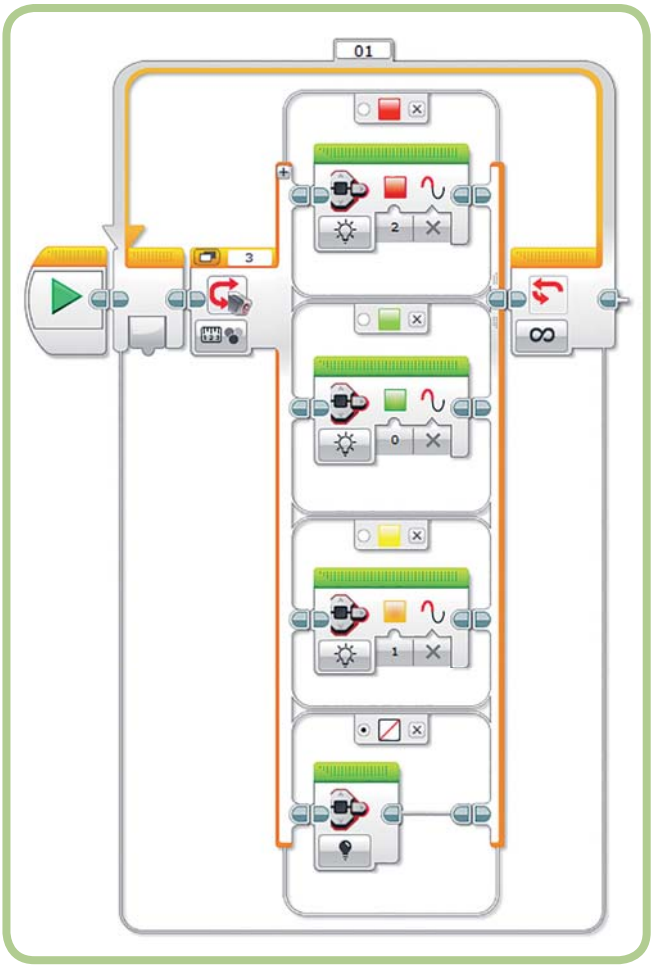


Рис. 14.11. Программа ColorCopy

ПРАКТИКУМ 14.2

Напиши программу *ProximityAlarm*, благодаря которой меняется подсветка индикатора состояния модуля в зависимости от расстояния между роботом и объектом. Используй инфракрасный или ультразвуковой датчик, чтобы определить расстояние и включить мигающий красный свет, если это расстояние меньше порогового значения, в противном случае индикатор должен непрерывно гореть зеленым светом.

Запусти программу. Цвет подсветки должен соответствовать цвету объекта, находящегося перед датчиком цвета. Когда перед датчиком ничего нет или цвет находящегося перед ним объекта отличается от красного, зеленого или желтого, подсветка должна отключиться.

Блок Экран

В блоке **Экран** (Display) предусмотрены четыре основных режима: **Текст** (Text), **Фигуры** (Shapes), **Изображение** (Image) и **Окно сброса настроек** (Reset Screen). Ты уже знаком с режимом **Текст** (Text). Режим **Окно сброса настроек** (Reset Screen) просто выводит на экран информацию, которая обычно отображается при выполнении программы. О режимах **Изображение** (Image) и **Фигуры** (Shapes) стоит рассказать подробнее.

Вывод изображения на экран

С помощью режима **Изображение** (Image) на экран выводится модуль EV3 изображение. В программном обеспечении EV3 существует широкий выбор изображений, среди которых есть несколько лиц, стрелки, циферблаты и другие объекты. Для создания собственных файлов изображений ты можешь использовать Редактор изображения EV3. Для этого выбери команду меню **Инструменты** (Tools) ⇒ **Редактор изображения** (Image Editor).

На рис. 14.12 показан блок **Экран** (Display) в режиме **Изображение** (Image). Щелкни по полю **Имя файла** (File Name), чтобы выбрать изображение. Нажми кнопку **Предварительный просмотр** (Display Preview), чтобы отобразить или скрыть окно предварительного просмотра, с помощью него можно увидеть, как выбранное изображение будет выглядеть на экране модуля EV3 (рис. 14.13).

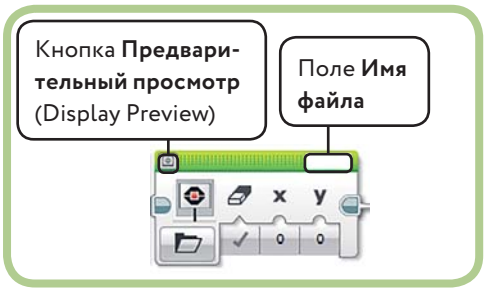


*Рис. 14.12. Блок **Экран** в режиме **Изображение***

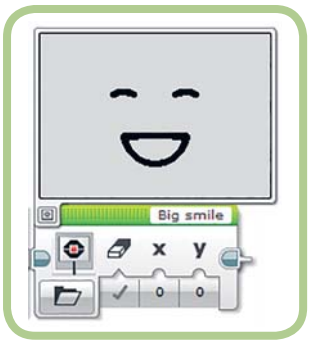


Рис. 14.13. Окно предварительного просмотра

Параметры **X** и **Y** определяют позицию верхнего левого угла изображения. Экран EV3 представляет собой сетку из точек, называемых *пикселами*. Слово *pixel* — это сокращение от *picture element*, которое переводится как элемент изображения. Размер **экрана составляет 178 пикселов** в **ширину** и 128 **пикселов** в высоту, а местоположение каждого пиксела определено значениями **X** и **Y**. Значение **X** определяет положение по горизонтали, при этом значения идут слева направо от 0 до 177. Значение **Y** указывает положение по вертикали, при этом отсчет ведется сверху вниз с 0 до 127 (рис. 14.14).

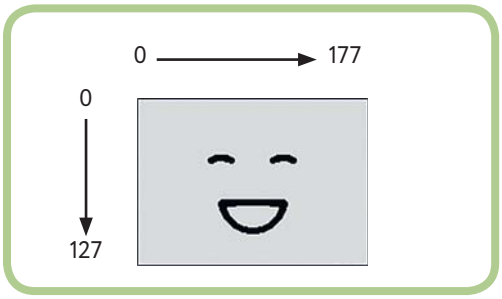


*Рис. 14.14. Значения **X** и **Y** для экрана модуля EV3*

Задай значения параметров **X** и **Y**, чтобы изменить позицию изображения на экране модуля EV3. Например, на рис. 14.15 показан большой смайлик (также используемый на рис. 14.14), перемещенный к самому верху экрана. Это стало возможным после того, как было задано значение –41 для параметра **Y**. В результате изображение удалось переместить на 41 пиксел вверх. В зависимости от значений, заданных для параметров **X** и **Y**, часть изображения может быть потеряна.

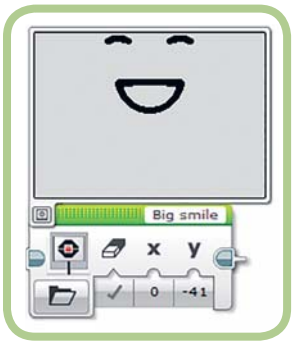


Рис. 14.15. Большой смайлик вверху экрана

Программа Eyes

С помощью изображения на экране модуля у робота появляется индивидуальность. Простым примером этого может послужить программа *Eyes*, в которой используется блок **Случайное значение** (Random) и шесть изображений подобных тому приведенному на рис. 14.16,

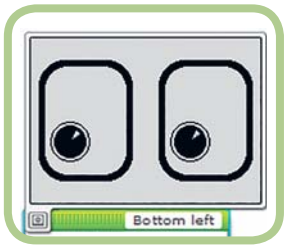


Рис. 14.17. Программа Eyes

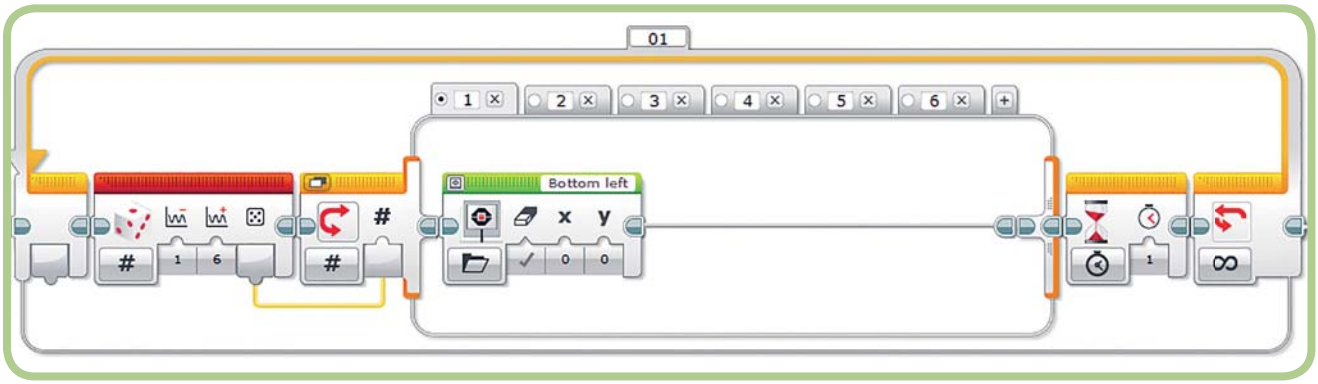


Рис. 14.16. Изображение Bottom left

чтобы придать экрану модуля EV3 скучающее выражение, как будто он ожидает от пользователя каких-то действий. Единственная разница между этими шестью изображениями заключается в направлении взгляда.

Данная программа показана на рис. 14.17. В блоке **Случайное значение** (Random) генерируется число от 1 до 6, которое затем передается в блок **Переключатель** (Switch). Каждый из шести случаев блока **Переключатель** (Switch) содержит блок **Экран** (Display), выводящий на экран модуля разные изображения. Файлы изображений для каждого случая показаны в табл. 14.1. Благодаря блоку **Ожидание** (Wait) перед сменой изображения возникает небольшая пауза.

Табл. 14.1. Файлы изображений для программы Eyes

Случай	Имя файла
1	Bottom leftBottom right
2	Middle leftMiddle right
3	Up
4	Down
5	
6	

После запуска программы ты увидишь на экране модуля блуждающий взгляд.

Рисование на экране модуля EV3

С помощью режима **Фигуры** (Shapes) блока **Экран** (Display) ты можешь рисовать точки, круги, прямоугольники и линии. На рис. 14.18 показан блок **Экран** (Display) в режиме **Фигуры** (Shapes) ⇒ **Точка** (Point). Определи местоположение точки, задав значения параметров **X** и **Y**. Параметр **Цвет** (Color), обозначенный значком в виде черного квадрата, отражает информацию о цвете точки: черный (пиксел включен) или белый (пиксел выключен). Если ты хочешь очистить экран, не рисуя на нем ничего нового, можешь просто нарисовать на экране один белый пиксел, задав для параметра **Очистить экран** (Clear Screen) значение **Истина** (True). Нарисованный пиксел не отобразится, поскольку все остальные пикселы тоже белые.

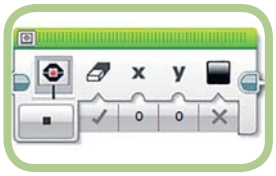


Рис. 14.18. Рисование точки

На рис. 14.19 показан блок **Экран** (Display) в режиме **Фигуры** (Shapes) ⇒ **Круг** (Circle). В данном случае параметры **X** и **Y** определяют местоположение центра круга, а параметр **Радиус** (Radius) — его размер. С помощью параметра **Заполнить** (Fill), который обозначен значком в виде ведра с краской, определяется, будет ли заполнена внутренняя часть круга, или будет нарисован только его контур. В окне предварительного просмотра показано, как круг будет выглядеть на экране.

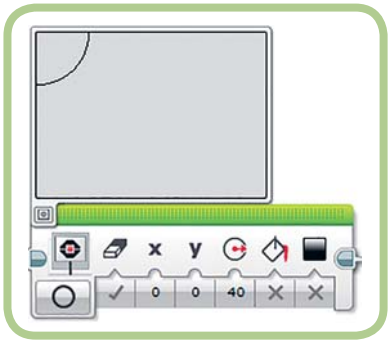


Рис. 14.19. Рисование круга

На рис. 14.20 показан блок **Экран** (Display) в режиме **Фигуры** (Shapes) ⇒ **Прямоугольник** (Rectangle). Параметры **X** и **Y** задают положение верхнего левого угла прямоугольника, а его размер определяется параметрами **Ширина** (Width) и **Высота** (Height).

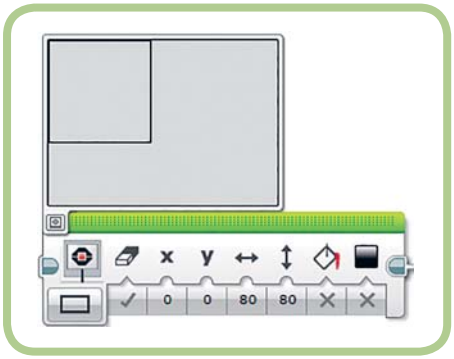


Рис. 14.20. Рисование прямоугольника

На рис. 14.21 показан блок **Экран** (Display) в режиме **Фигуры** (Shapes) ⇒ **Прямая** (Line). Чтобы нарисовать прямую линию, требуется указать положение ее начальной и конечной точек. Параметры **X1** и **Y1** определяют местоположение одного конца линии, параметры **X2** и **Y2** — местоположение другого ее конца.

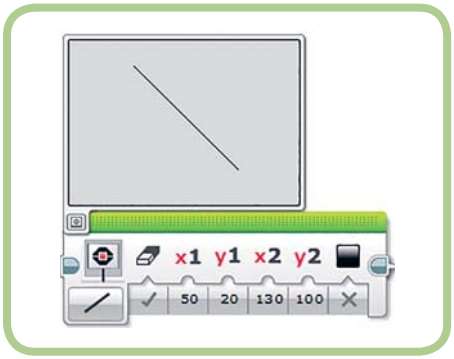


Рис. 14.21. Рисование прямой линии

Программа EV3Sketch

В этом разделе ты создашь программу *EV3Sketch*, в которой используется функция рисования прямой линии блока **Экран** (Display) для превращения робота TriBot в доску для рисования. Два колеса робота будем использовать для задания координат точек. Основная идея довольно проста: программа многократно проводит прямую линию от предыдущей точки до местоположения, определяемого текущим показанием двух датчиков вращения мотора.

В этой программе используются две переменные **X** и **Y** для хранения координат последней точки. В начале программы экран модуля EV3 очищается, переменные инициализируются нулем, а показания датчиков вращения моторов сбрасываются. Затем программа входит в цикл, в котором сначала считывается показание датчиков вращения для задания нового местоположения (мотор B задает новое значение **X**, а мотор C задает новое значение **Y**). После этого между старым и новым местоположением проводится прямая линия, а координаты нового местоположения сохраняются в переменных **X** и **Y**, чтобы их можно было использовать при следующем выполнении цикла.

Помимо рисования прямой линии в этой программе должна быть функция очистки экрана модуля EV3 перед началом создания нового рисунка. Ты можешь добавить ее, сделав так, чтобы при щелчке кнопкой «Центр» для параметра **Очистить экран** (Clear Screen) блока **Экран** (Display) задавалось значение **Истина** (True). В листинге 14.2 приведен псевдокод для этой программы.

Листинг 14.2. Программа EV3Sketch

```
clear the EV3 screen
set X to 0
set Y to 0
reset the Rotation Sensors for motors B and C
begin loop
   read the Rotation Sensor for motor B
   read the Rotation Sensor for motor C
   draw a line from X, Y to the point defined by
      the motor B and C positions; if the Center
      button is bumped then set the Clear option
   set X to the motor B position
   set Y to the motor C position
loop forever
```

Первый раздел программы показан на рис. 14.22. Он очищает экран модуля EV3, инициализирует переменные и сбрасывает показания датчиков вращения моторов.

На рис. 14.23 показана основная часть программы, в которой блок **Экран** (Display) рисует прямую линию. Для этого данному блоку нужно предоставить координаты двух точек: начальной, определяемой значениями переменных **X** и **Y**, и конечной, определяемой показаниями датчиков вращения моторов B и C. Все эти значения передаются блоку **Экран** (Display) с помощью шины данных.

Рис. 14.22. Инициализация экрана, переменных и датчиков

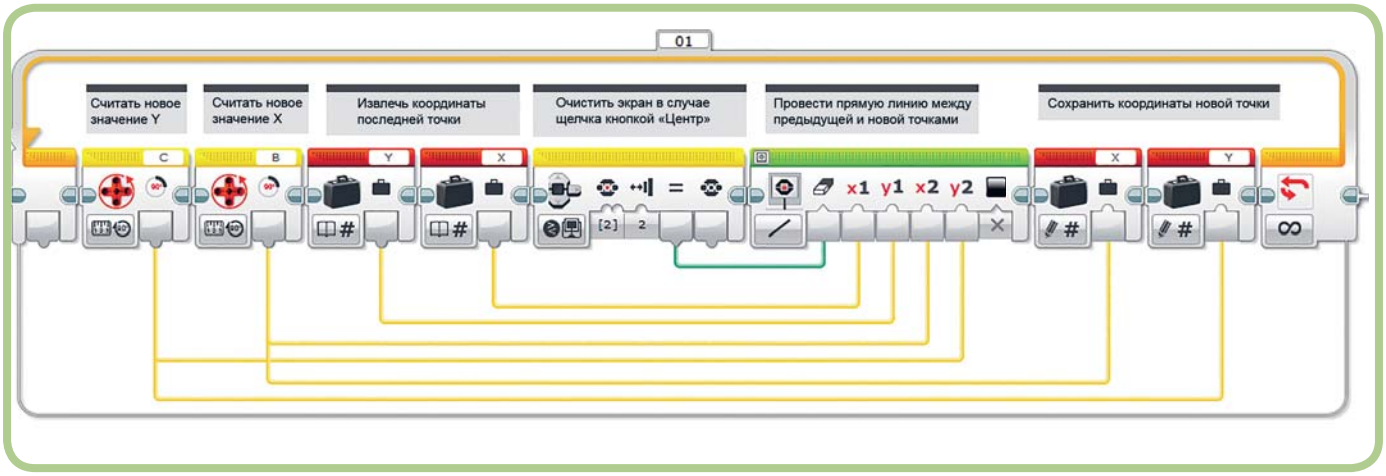


Рис. 14.23. Чтение показаний датчиков, рисование линии и сохранение координат новой точки

С помощью блока **Кнопки управления модулем** (Brick Buttons) осуществляется проверка, был ли сделан щелчок кнопкой «Центр», и в соответствии с результатом на экран выводится значение «Истина» или «Ложь». Это значение передается в блок **Экран** (Display) и используется для управления параметром **Очистить экран** (Clear Screen), поэтому, когда ты нажимаешь (и отпускаешь) эту кнопку, блок **Экран** (Display) очищает экран модуля EV3, чтобы на нем можно было нарисовать новую линию.

В последних двух блоках сохраняются показания датчиков вращения моторов в переменных **X** и **Y**, чтобы их можно было применять при следующем выполнении цикла.

В пяти блоках содержатся параметры для блока **Экран** (Display), и поскольку порядок следования этих блоков не имеет значения, они расположены так, чтобы шины данных не пересекались слишком часто. Делай это по мере возможности, чтобы облегчить восприятие программы.

После запуска программы сначала необходимо очистить экран модуля EV3. Создай рисунок, вращая колесо B для перемещения виртуального пера влево-вправо, а колесо C — для его перемещения вверх-вниз. Нажми кнопку «Центр», чтобы очистить экран модуля.

Первая линия начинается в верхнем левом углу экрана модуля EV3. Чтобы начать линию с какой-то другой точки, перемести виртуальное перо в необходимое место, а затем нажми кнопку «Центр» для очистки экрана, после чего ты сможешь начать создавать новый рисунок.

Дальнейшее исследование

Вот еще два упражнения, в которых используются идеи, рассмотренные в этой главе:

1. Программа *PowerSetting* имеет один весьма раздражающий недостаток: она не ограничивает диапазон допустимых значений мощности числами 0 и 100. Измени эту программу так, чтобы значение никогда не было меньше 0 или больше 100 (другими словами, программа должна игнорировать любое изменение, из-за которого значение оказывается вне диапазона). Для предупреждения пользователей о недопустимом значении сделай так, чтобы при попытке задать значение меньше 0 или больше 100 индикатор состояния модуля в течение одной секунды мигал красным цветом.

2. Адаптируй программу *CountDown* из гл. 13 так, чтобы на экране модуля отображалась серия изображений, пока идет отсчет времени. Ты можешь использовать одну из групп изображений, предусмотренных в программном обеспечении EV3 для отображения циферблата, индикатора выполнения или таймера, а можешь использовать инструмент **Редактор изображения** (Image Editor) для создания собственной последовательности изображений.

Заключение

С помощью кнопок модуля EV3 удобно взаимодействовать с программой. В блоке **Индикатор состояния модуля** (Brick Status Light) можно контролировать подсветку кнопок — еще один способ передачи информации пользователю и делает программы более интересными.

На примере программы *PowerSetting* было продемонстрировано использование кнопок «Влево» и «Вправо» для задания значения переменной. В других программах этой главы было показано, что с помощью блока **Экран** (Display) на экране модуля можно отобразить не только текст, но и изображения, и рисовать на нем рисунки. Ты можешь включить эти функции в свои программы, чтобы в полной мере использовать экран модуля EV3.

15

Массивы

Из этой главы ты узнаешь о массивах, используя которые, можно хранить списки числовых и логических значений. До сих пор все значения, которые мы применяли в своих программах (в качестве параметров, в шинах данных и переменных), представляли собой отдельные фрагменты данных. Использование массива позволяет хранить в одной переменной список значений.

После краткого обзора массивов мы создадим простую тестовую программу, чтобы разобраться в принципе их работы. Затем напишем три довольно сложные программы, содержащие массивы: в одной сможем создавать простые программы для робота TriBot с помощью кнопок модуля; в другой — подсчитать количество объектов того или иного цвета; в третьей будет реализована игра на запоминание.

Обзор и терминология

Массив EV3 — это упорядоченный список значений, доступ к которым осуществляется по их позиции в списке. Каждое значение в массиве называется *элементом*, а позиция элемента — его *индексом*. В массивах EV3 индексы начинаются с 0 (как это часто бывает в мире компьютерного программирования). Таким образом, индексом первого элемента массива является 0, индексом второго элемента — 1 и т. д. Количество значений в массиве — это его *длина*. Используя программное обеспечение EV3, можно работать с числовым и логическим массивами. В конкретном массиве может содержаться либо группа числовых, либо логических значений, но никак не их сочетание.

В качестве примера рассмотрим массив с именем SampleValues, в котором содержатся цвета объектов, обнаруженных программой (с помощью датчика цвета). Скажем, программа распознала синий объект, красный объект, белый объект и еще один красный объект; таким образом, данный массив содержит числа 3, 5, 6 и 5. Значения этого массива и их индексы перечислены в табл. 15.1.

Табл. 15.1. Индексы и значения элементов массива SampleValues

Индекс	0	1	2	3
Значение	3	5	6	5

Длина этого массива равна 4, а индексы идут от 0 до 3. Поскольку индексы начинаются с 0, индекс последнего элемента массива всегда будет на единицу меньше его длины. Каждый элемент имеет уникальный индекс, однако хранящиеся в массиве значения не обязательно должны быть уникальными. Например, в массиве SampleValues содержатся два элемента со значением 5, поскольку программа распознала два красных объекта.

При обозначении элемента массива обычно используются сокращения. Так, выражение «значение элемента массива SampleValues с индексом 1» можно записать в виде “SampleValues [1]”.

Создание массива

Во всех программах этой главы для создания массива используется блок **Переменная** (Variable). С помощью блоков **Константа** (Constant) и **Операции над массивом** (Array Operations) также можно создавать массивы; тем не менее, блок **Переменная** (Variable) используется с этой целью гораздо чаще, поскольку в любом случае массив необходимо сохранить в качестве переменной.

Например, для создания массива SampleValues с помощью блока **Переменная** (Variable) выбери режим **Записать** (Write) ⇒ **Числовой массив** (Numeric Array) и задай **SampleValues** в качестве имени переменной (рис. 15.1). По умолчанию для параметра **Значение** (Value) выбран вариант [], соответствующий *пустому массиву*. В нем не содержится никаких элементов, следовательно, его длина равна 0. В программах переменная довольно часто инициализируется пустым массивом, в который по мере выполнения программы добавляются элементы.

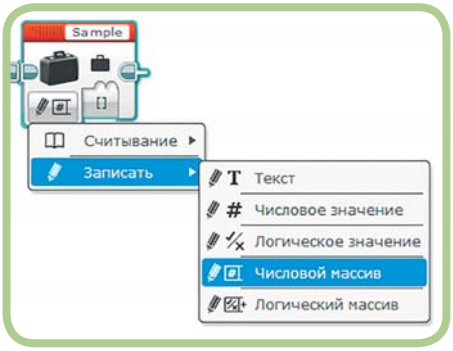


*Рис. 15.1. Создание массива с помощью блока **Переменная** (Variable)*

Обрати внимание на то, что поле параметра **Значение** (Value) данного блока имеет вверху два полукруга: [зн]. Значит, что значением данного параметра является числовой массив. В случае логического массива верхняя граница поля отмечается двумя треугольниками.

Для того чтобы добавить в массив SampleValues четыре значения из табл. 15.1, щелкни по параметру **Значение** (Value) блока **Переменная** (Variable). В результате на экране отобразится поле, в котором можно добавлять, удалять или редактировать элементы. Щелкни по кнопке +, чтобы добавить новый элемент, а затем задай его значение. На рис. 15.2 показан список после добавления первых двух элементов. Для удаления элемента щелкни по значку **X** справа от элемента.

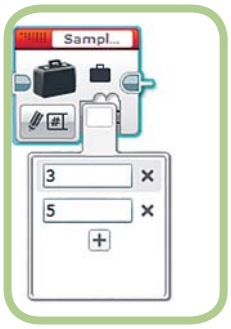


Рис. 15.2. Поле для задания значений элементов

Теперь в поле **Значение** (Value) отображается начало массива, однако большинство его элементов скрыто, поскольку массив превышает ширину поля данного параметра. Чтобы увидеть все элементы, наведи указатель мыши на маленький значок в виде чемодана, как показано на рис. 15.3.

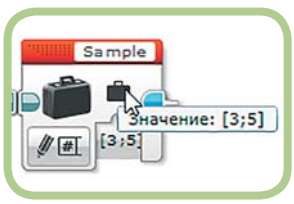


Рис. 15.3. Отображение всех элементов массива

Теперь, когда мы создали массив, посмотрим, что с ним можно делать, используя блок **Операции над массивом** (Array Operations).

Блок «Операции над массивом»

Для блока **Операции над массивом** (Array Operations) массив является входным параметром, при его использовании можно добавить новый элемент в конец массива, считать или записать значения элементов массива или выяснить его длину (количество элементов). В этом режиме можно определить выполняемую операцию, а также тип данных массива (числовой или логический) (рис. 15.4).

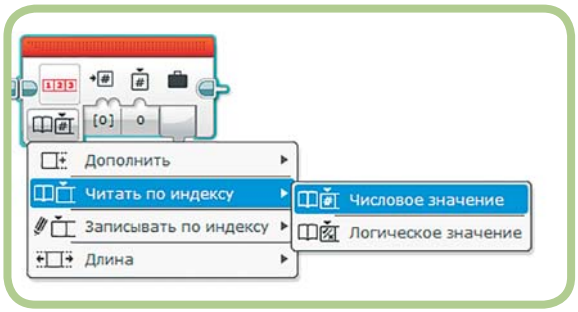


*Рис. 15.4. Блок **Операции над массивом** (Array Operations)*

Режим «Длина»

В режиме **Длина** (Length) можно узнать, сколько элементов содержится в массиве. На рис. 15.5 показана программа, отображающая длину массива SampleValues. Для этого режима в блоке **Операции над массивом** (Array Operations) предусмотрены два параметра: ввод (обычно значение передается с помощью шины данных) и вывод, в котором отображается длина массива. Обрати внимание на то, что шина данных, идущая от блока **Переменная** (Variable) к блоку **Операции над массивом** (Array Operations), толще, чем другие шины данных. Значит, в ней содержится массив значений.

С помощью первых двух блоков программы создается массив SampleValues, который затем помещается в шину данных для передачи в блок **Операции над массивом** (Array Operations). Блок **Операции над массивом** (Array Operations) предназначен для определения длины массива и передачи этого значения в блок **Экран** (Display), с помощью которого оно отобразится на экране модуля EV3. Используя блок **Ожидание** (Wait), ты сможешь прочитать содержимое экрана

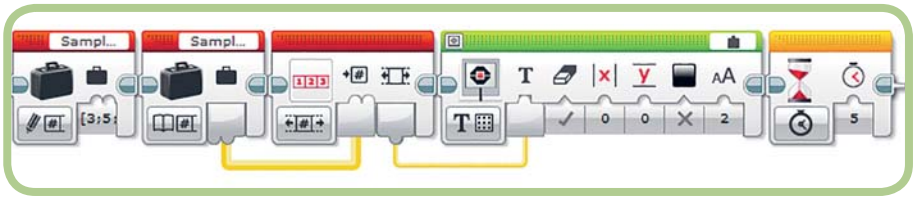


Рис. 15.5. Отображение длины массива SampleValues

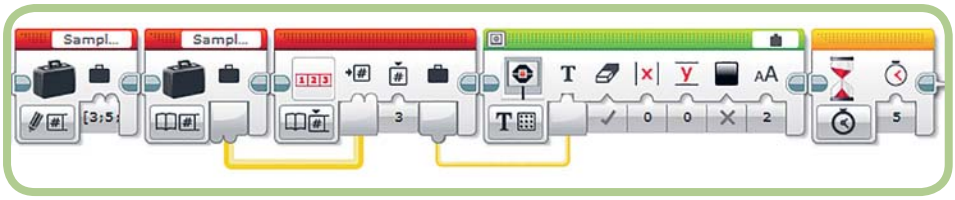


Рис. 15.6. Отображение значения элемента SampleValues [3]

перед завершением программы. В данной программе применяются элементы массива SampleValues (см. табл. 15.1), поэтому на экране должно отобразиться значение 4.

Режим «Читать по индексу»

В режиме **Читать по индексу** (Read At Index) в качестве входных параметров принимаются массив и индекс, блок выдает значение элемента с указанным индексом. Например, программа, показанная на рис. 15.6, отображает значение SampleValues [3] (элемент массива с индексом 3).

В процессе выполнения программы в блок **Операции над массивом** (Array Operations) принимается массив SampleValues в качестве входных данных, в блоке считывается значение элемента с индексом 3 и помещается выходное значение в шину данных. Затем с помощью блока **Экран** (Display) отображается значение, которое должно быть равно 5. Помни о том, что отсчет элементов начинается с 0, поэтому индекс 3 соответствует четвертому элементу массива.

Попытка считать значение несуществующего элемента приводит к ошибке. Если индекс, переданный блоку **Операции над массивом** (Array Operations), больше или равен длине массива, то выполнение программы немедленно прервется, а на экране модуля EV3 появится треугольный значок, часто используемый для обозначения ошибки или предупреждения.

Режим «Записывать по индексу»

В режиме **Записывать по индексу** (Write At Index) можно изменить значение элемента массива. На рис. 15.7 показана программа, с помощью которой можно изменить значение элемента SampleValues [3] на 4. В этом режиме в блоке **Операции над массивом** (Array Operations) принимаются три входных параметра: исходный массив; индекс изменяемого элемента; новое значение этого элемента.

Важно отметить, что при использовании одного лишь блока **Операции над массивом** (Array Operations) значение переменной SampleValues не меняется. Для того чтобы это сделать, необходим блок **Переменная** (Variable) для сохранения обновленного массива значений. На рис. 15.8 показан обновленный массив, переданный по второй шине данных. В результате значение 5, которое находилось в конце массива, заменено на 4.

При записи значения по еще не существующему индексу с помощью программного обеспечения EV3 можно расширить массив до необходимой длины, однако все остальные индексы, добавленные таким способом, заполняются случайными значениями.

Режим «Дополнить»

Используя режим **Дополнить** (Append), можно добавить новый элемент в конец массива, при этом его длина увеличивается на 1. Программой на рис. 15.8 в массив добавляется новый элемент со значением 7.

После запуска этой программы ты увидишь, что в конец массива было добавлено значение 7. Теперь длина массива равна 5.

ПРИМЕЧАНИЕ **При проверке содержимого шины данных отображаются только первые пять элементов массива.**

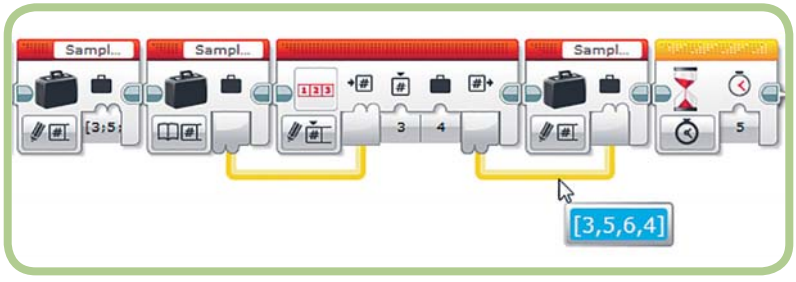


Рис. 15.7. Изменение значения элемента SampleValues [3] на 4

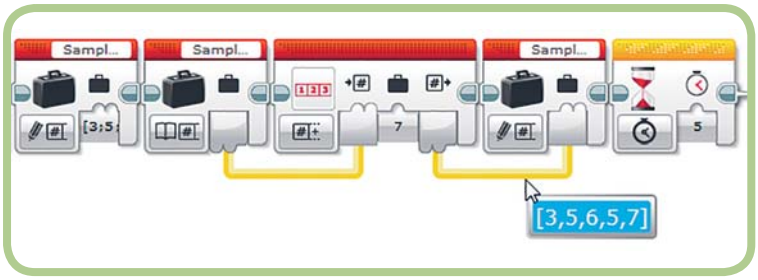


Рис. 15.8. Добавление нового элемента в массив SampleValues

Программа ArrayTest

Рассмотрим программу *ArrayTest*, которая может сгенерировать некоторые типичные операции над массивом. Эта программа создает пустой массив, добавляет в него первые пять чисел, кратных 2 (начиная с 0), а затем отображает значения на экране модуля EV3.

В первой части программы, показанной на рис. 15.9, с помощью блока **Переменная** (Variable) создается пустой числовой массив с именем ArrayValue. Блок **Цикл** (Loop) выполняется пять раз. Каждый раз, когда программа считывает текущее значение массива (с помощью блока **Переменная** (Variable)), в массив добавляется новое число, которое в два раза больше значения **Параметр цикла** (Loop Index) (при этом используются блоки **Математика** (Math) и **Операции над массивом** (Array Operations)). После этого обновленный массив сохраняется во втором блоке **Переменная** (Variable). После выхода из цикла переменная ArrayValue должна содержать массив со значениями: 0, 2, 4, 6 и 8.

Примечание. Для того чтобы наблюдать за построением массива, добавь паузу в 1 секунду перед окончанием цикла и используй программное обеспечение EV3 для проверки значения, которое содержится в шине данных, выходящей из блока **Операции над массивом** (Array Operations). Благодаря этой паузе у тебя появится время оценить изменения в массиве после каждого цикла.

Вторая половина программы, показанная на рис. 15.10, является более сложной. Однако ее основная идея проста: перебрать элементы массива и отобразить значение каждого из них на экране модуля. Далее рассмотрим принцип работы всех блоков слева направо.

1. С помощью первого блока **Экран** (Display) очищается экран модуля.

2. В блоке **Переменная** (Variable) считывается значение в массиве ArrayValue и передается его по шине данных в блоки **Операции над массивом** (Array Operations).

3. В первом блоке **Операции над массивом** (Array Operations) проверяется длина массива, это значение используется для задания количества повторов блока **Цикл** (Loop).

4. Во втором блоке **Операции над массивом** (Array Operations) считывается значение одного элемента массива, при этом в качестве индекса используется значение **Параметр цикла** (Loop Index), которое затем передается в блок **Текст** (Text). Таким образом, при первом выполнении цикла считывается значение ArrayValue[0], при втором — ArrayValue [1] и т. д.

5. С помощью блока **Текст** (Text) каждое значение отображается на отдельной строке, при этом используется значение **Параметр цикла** (Loop Index) для настройки параметра **Строка** (Row). Для параметра **Очистить экран**

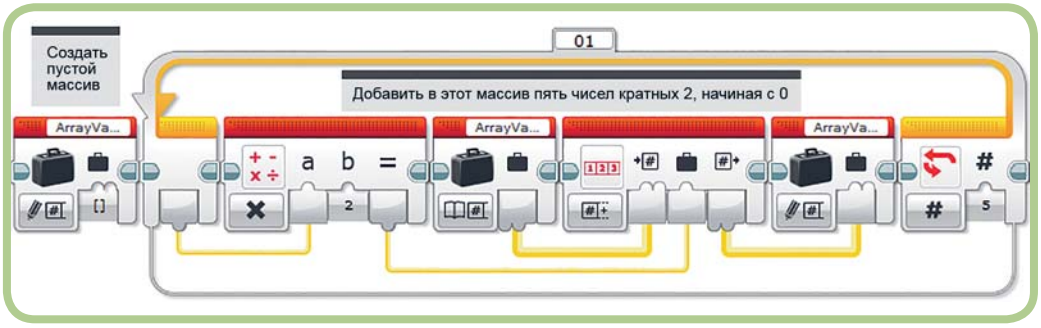


Рис. 15.9. Программа ArrayTest, часть 1

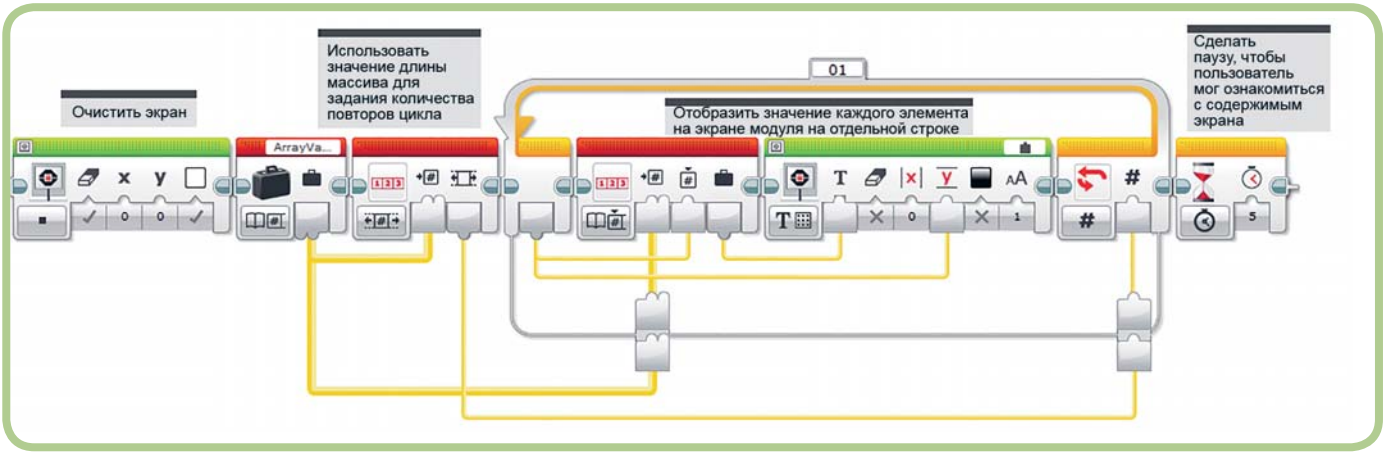


Рис. 15.10. Программа ArrayTest, часть 2

(Clear Screen) выбран вариант **Ложь** (False), чтобы экран не очищался при каждом выполнении цикла.

6. Блоком **Ожидание** (Wait) выполнение программы приостанавливается для того, чтобы у тебя была возможность ознакомиться с результатом перед очищением экрана.

Обрати внимание на то, что значение переменной ArrayValue не надо считывать внутри цикла, поскольку ее значение никогда не меняется. При каждом выполнении цикла в шине данных, идущей к блоку **Операции над массивом** (Array Operations), будет содержаться одно и то же значение (весь массив). Изменяется только значение, используемое в качестве индекса массива, поскольку оно зависит от значения **Параметр цикла** (Loop Index).

Еще один важный момент: эта программа работает благодаря тому, что индекс массива, **Параметр цикла** (Loop Index) и значение параметра **Строка** (Row) начинаются с 0. То, что подсчет начинается с 0, может показаться странным, однако при соблюдении согласованности это оказывается весьма удобным.

ПРАКТИКУМ 15.1

Отображение значений массива часто оказывается полезным при отладке программы. Создай контейнер «Мой блок», с помощью которого можно решить эту задачу, на основе кода программы *ArrayTest*. В этом новом блоке массив должен приниматься в качестве входного параметра, а на экране модуля должно отображаться значение каждого его элемента. Помни о том, что на экране EV3 можно разместить только 12 строк, поэтому тебе необходимо отобразить первые 12 элементов, дождаться, пока пользователь нажмет кнопку, отобразить следующие 12 элементов и так далее вплоть до окончания массива. Должны отображаться индекс и значение каждого элемента, иначе в случае длинных массивов легко сбиться со счета. Для этого можно использовать контейнер **DisplayNumber**.

СОВЕТ Если в массиве содержится более 12 элементов, то необходимо «обнулить» параметр **Строка** (Row) блока **Экран** (Display), чтобы 13-й элемент отобразился в строке 0. В данном случае можно выполнить над индексом массива операцию деления по модулю для задания значения параметра **Строка** (Row).

Программа ButtonCommand

В программе *ButtonCommand* для передачи роботу TriBot списка команд используются кнопки модуля. В Первой части программы создается массив команд в зависимости от нажимаемых кнопок, а во второй эти команды выполняются, в результате робот двигается в соответствии с твоими инструкциями.

Будем использовать все пять кнопок модуля EV3. Кнопки «Влево» и «Вправо» заставляют робота повернуть на 90° влево или вправо, а кнопки «Вверх» и «Вниз» — переместиться вперед или назад на один оборот мотора. Нажатие кнопки «Центр» будет означать, что мы закончили ввод команд и готовы запустить робота.

Создание массива команд

В этой программе для хранения команд, которые робот должен выполнить, применяется числовой массив CommandList. Необходимо записать каждую команду, используя числа, соответствующие номерам кнопок модуля, как показано в табл. 15.2.

Табл. 15.2. Номера кнопок и соответствующие им команды

Кнопка	Номер	Команда
Влево	1	Поверни на 90° влево.
Вправо	3	Поверни на 90° вправо.
Вверх	4	Переместись вперед на один оборот мотора.
Вниз	5	Переместись назад на один оборот мотора.

Для составления списка команд программа сначала создает пустой массив. Затем входит в цикл и находится в режиме ожидания щелчка кнопкой, и либо добавляет номер нажатой кнопки в список, либо выходит из цикла при щелчке кнопкой «Центр». В процессе создания списка команд они отображаются на экране модуля EV3.

Начнем с кода, отвечающего за добавление команд в массив, а в следующем разделе добавим блоки для отображения каждого значения. С помощью кода на рис. 15.11 очищается экран модуля, создается пустой массив, в который добавляются команды вплоть до щелчка кнопкой «Центр».

Принцип работы первых двух блоков очевиден, однако следует пояснить блоки, находящиеся внутри цикла.

1. В блоке **Ожидание** (Wait) используется режим **Кнопки управления модулем** (Brick Buttons) ⇒ **Сравнение** (Compare), чтобы дождаться щелчка любой из пяти кнопок. (На рис. 15.12 видно, что выбраны все пять кнопок модуля.) С помощью выходного параметра

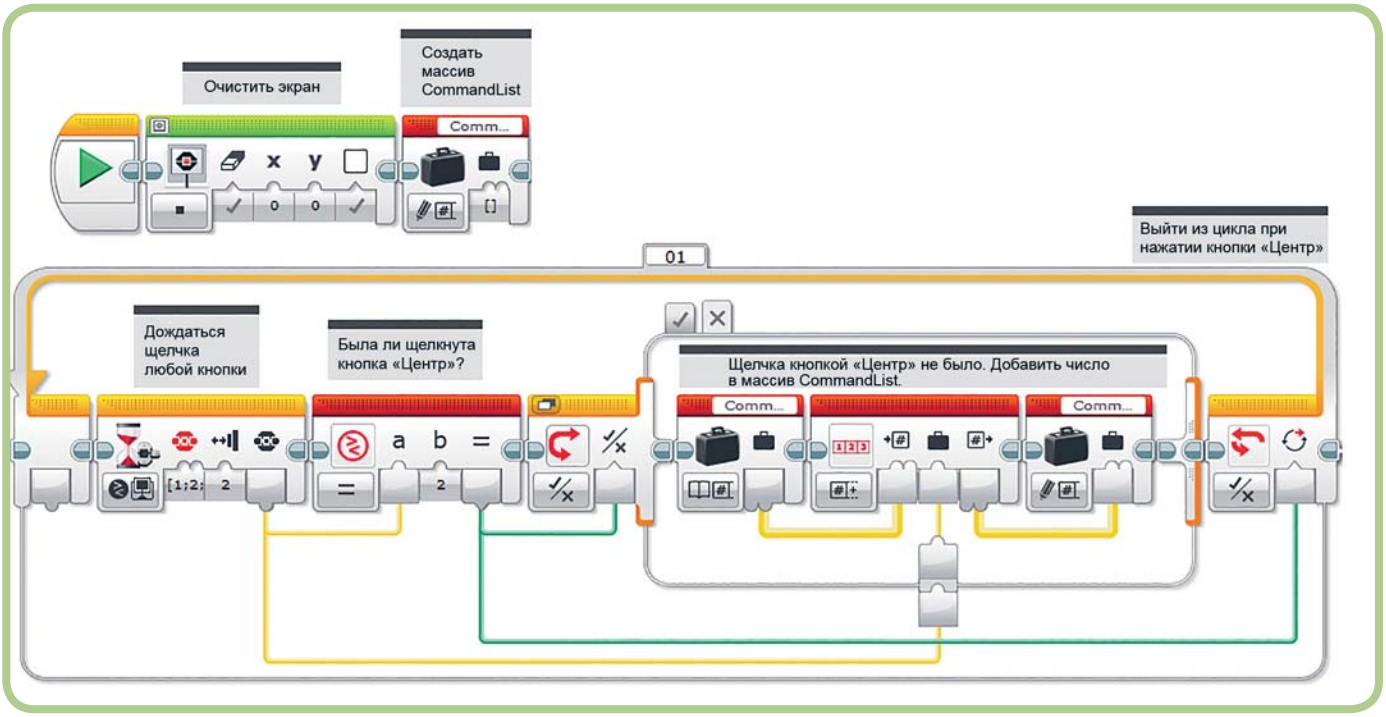


Рис. 15.11. *Добавление команд в массив CommandList*

Идентификатор кнопки (Button ID) будет указано, какая именно кнопка была щелкнута.

2. С помощью блока **Сравнение** (Compare) осуществляется проверка, была ли щелкнута кнопка «Центр» (номер 2). Результат передается блоку **Переключатель** (Switch) и блоку **Цикл** (Loop).

3. Случай «Истина» блока **Переключатель** (Switch) пуст, поэтому в случае щелчка кнопкой «Центр» переходим в конец цикла. Если была щелкнута другая кнопка, — переходим в случай «Ложь» блока **Переключатель** (Switch) и выполняем три находящиеся там блока.

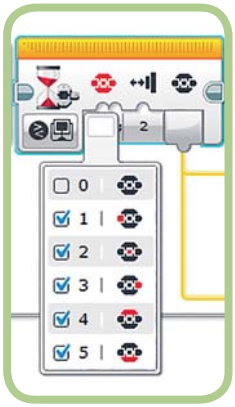


Рис. 15.12. *Дождись щелчка любой из пяти кнопок*

4. В первом блоке **Переменная** (Variable) (внутри блока **Переключатель** (Switch)) текущий массив CommandList помещается в шину данных.

5. В блоке **Операции над массивом** (Array Operations) в конец массива добавляется идентификатор кнопки из блока **Ожидание** (Wait).

6. Во втором блоке **Переменная** (Variable) сохраняется обновленный массив в переменной CommandList.

7. В блоке **Цикл** (Loop) используется результат блока **Сравнение** (Compare) для определения момента выхода из цикла, чтобы осуществить его при щелчке кнопкой «Центр».

Отображение команд

По мере добавления в массив новых команд программа должна отображать соответствующий номер на экране модуля, поэтому после нажатия кнопок «Вверх», «Вверх», «Вверх», «Вправо» на экране должно отображаться «4443». Кроме того, необходимо, чтобы программа выводила на экран как можно больше значений. Если мы просто отобразим каждое значение в новой строке, это будет лишь 12 значений. Аналогично, если для каждого значения вводить новый столбец, получится только 22 значения. Исходя из этого, в этой программе мы будем использовать строки и столбцы: первое значение появится в верхнем левом углу (строка 0, столбец 0), следующее значение — в положении (0,1), т. е. строка 0, столбец 1, и т. д. До тех пор пока мы не дойдем до конца строки. Затем значения начнут появляться в следующей строке: строка 1, столбец 0. В результате мы сможем отобразить на экране модуля до 264 значений (22 × 12). Этого количества должно быть достаточно для данной программы.

Для определения строки и столбца для каждого значения применим две простые формулы, основанные на значении **Параметр цикла** (Loop Index). Помни о том, что этот параметр будет соответствовать индексу каждого элемента массива. Для определения строки делим значение Параметр цикла на 22 и округляем его в меньшую сторону (с помощью функции floor) до ближайшего целого числа: floor (Параметр цикла / 22). Когда значение **Параметр цикла** (Loop Index) находится между 0 и 21, результатом данного выражения является 0, а число, соответствующее новой команде, отображается на первой строке. Когда значение этого параметра находится между 22 и 43, результат выражения равен 1, а число, соответствующее команде, отображается на второй строке и т. д.

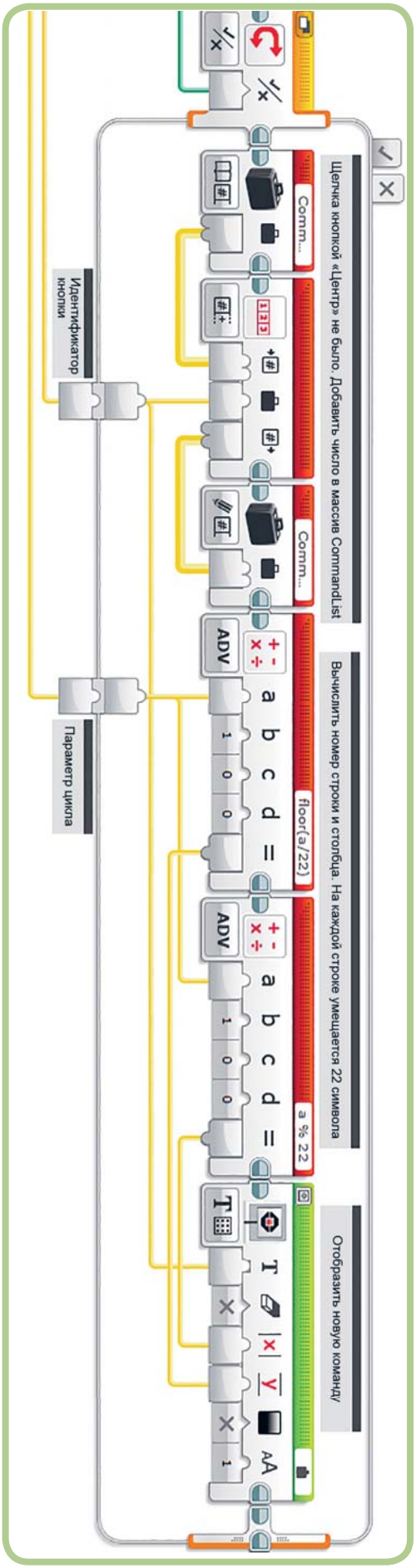


Рис. 15.13. Отображение команд

Чтобы определить, какой столбец следует использовать, необходимо найти остаток от деления значения **Параметр цикла** (Loop Index) на 22: Параметр цикла % 22. Так, мы определим количество пробелов, оставшихся после выбора строки для отображения команды. Когда значение **Параметр цикла** (Loop Index) находится в диапазоне от 0 до 21, результат выражения совпадает со значением этого параметра. Когда значение этого параметра достигает 22, результат выражения равен 0, поэтому номер команды отображается в первом столбце слева. При следующем выполнении цикла значение **Параметр цикла** (Loop Index) равно 23, а поскольку остаток от деления 23 на 22 равен 1, команда отображается в столбце 1. Это еще один способ применения оператора деления по модулю для осуществления циклического возврата, в данном случае с использованием ширины экрана.

На рис. 15.13 показан блок **Переключатель** (Switch), приведенный ранее на рис. 15.11, с двумя блоками **Математика** (Math) и блоком **Экран** (Display), добавленным для отображения каждой команды в подходящих строке и столбце. С помощью блоков **Математика** (Math) вычисляются номер столбца и строки в зависимости от значения **Параметр цикла** (Loop Index), которое подается на ввод *a* обоих блоков. Затем эти значения используются в блоке **Экран** (Display) для отображения новой команды на экране модуля. В блоке **Экран** (Display) для параметра **Шрифт** (Font) выбран вариант **1**, чтобы каждый символ занимал одну строку и один столбец, а для параметра **Очистить экран** (Clear Screen) выбран вариант **Ложь** (False).

Вероятно, на этом этапе ты захочешь протестировать программу и убедиться в том, что список команд хранится и отображается так, как нужно. Запустив эту программу из среды EV3, ты сможешь проверить значение в шине данных, выходящей из блока **Операции над массивом** (Array Operations), и убедиться, что программа работает правильно.

Выполнение команд

С помощью второй части программы (рис. 15.14) считывается каждый элемент массива и выполняется соответствующая команда. Структура этого цикла аналогична структуре цикла, используемого в программе *ArrayTest*. С помощью блока **Операции над массивом** (Array Operations) считывается длина массива, затем это значение используется для задания количества повторов блока **Цикл** (Loop). Внутри блока **Цикл** (Loop) для считывания значения одного элемента массива применяется **Параметр цикла** (Loop Index). При работе с массивами ты будешь часто использовать такую схему.

Затем значение элемента передается блоку **Переключатель** (Switch). Случаи блока **Переключатель** (Switch) соответствуют идентификаторам кнопок, связанных с каждой командой, и в каждом случае используется блок **Рулевое управление** (Move Steering) для перемещения робота TriBot влево, вправо, вперед или назад.

После запуска программы ты сможешь вводить команды с помощью четырех кнопок модуля «Вверх», «Вниз», «Влево» и «Вправо», при этом на экране модуля EV3 будут отображаться соответствующие числа (идентификаторы кнопок). По щелчку кнопки «Центр» робот начнет движение и будет продолжать двигаться до тех пор, пока не выполнит все введенные команды.

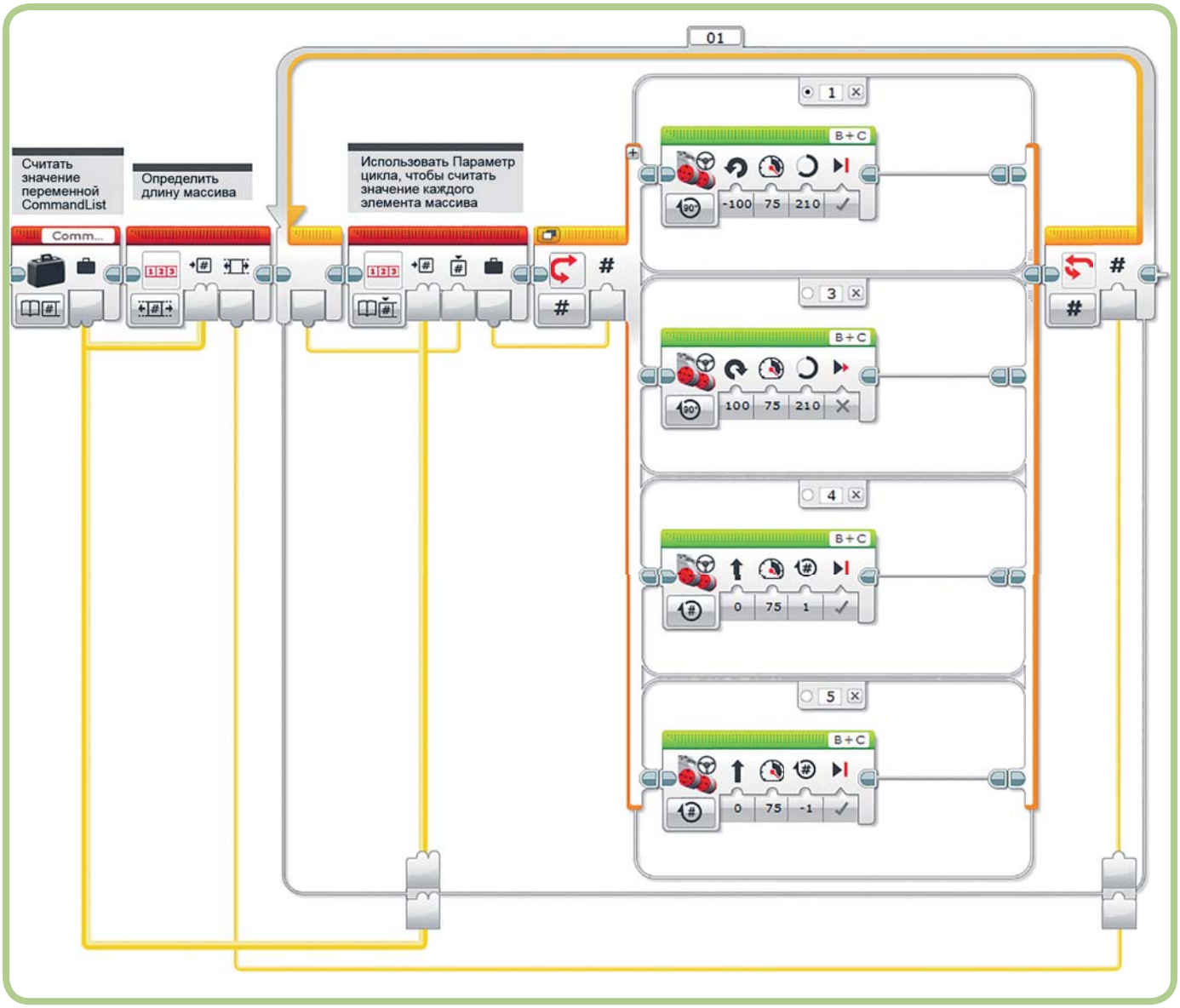


Рис. 15.14. Выполнение команд

ПРАКТИКУМ 15.2

Измени действия программы, чтобы сделать ее более интересной. В каждом случае блока **Переключатель** (Switch) может содержаться любое количество блоков или использоваться контейнеры «Мой блок» для выполнения еще более сложных действий. Например, с помощью следующих команд можно воссоздать поведение программы *BumperBot:*

1. Двигайся вперед (используй режим **Включить** (On)).

2. Дождись нажатия кнопки датчика касания.

3. Двигайся назад в течение 1 оборота мотора.

4. Повернись на случайное количество градусов.

Программа ColorCount

Программа *ColorCount* определяет число объектов каждого цвета, обнаруженных датчиком цвета. Этот процесс похож на действие программы *RedOrBlue*. Программа находится в режиме ожидания до тех пор, пока ты не поместишь перед датчиком цвета объект, а затем сообщает его цвет и подсчитывает общее количество обнаруженных объектов каждого цвета. Однако с помощью программы *ColorCount* можно отследить все восемь цветовых значений, которые может распознать датчик цвета (семь цветов и значение **Нет цвета** (No Color)). Напомним, что программа *RedOrBlueCount* из гл. 11, могла подсчитать количество только красных и синих объектов, используя две переменные. Программу *RedOrBlueCount* можно расширить и использовать в ней восемь переменных

для подсчета всех восьми цветовых значений, возвращаемых датчиком цвета. Однако в массиве лучше хранить восемь значений. Восемь цветовых значений — это семь цветов и значение **Нет цвета** (No Color).

Для отслеживания количества распознанных программой объектов каждого цвета нам понадобится числовой массив с восемью элементами, назовем его ColorCounts. После каждого измерения датчиком цвета отображается распознанное значение от 0 до 7, соответствующее цвету (табл. 15.3). Поскольку нумерация цветов и элементов массива начинается с 0, мы будем хранить количество объектов каждого цвета в позиции массива с соответствующим индексом. Например, в позиции ColorCounts [2] будет содержаться много синих объектов, а в позиции ColorCounts [4] — желтых объектов. Таким образом, в случае с данным массивом индекс отображает не только позицию элемента, но и цвет объектов, количество которых хранится в этой позиции.

Табл. 15.3. Значения датчика цвета

Номер	Цвет
0	**Нет цвета** (No Color)
1	**Черный** (Black)
2	**Синий** (Blue)
3	**Зеленый** (Green)
4	**Желтый** (Yellow)
5	**Красный** (Red)
6	**Белый** (White)
7	**Коричневый** (Brown)

В листинге 15.1 приведено высокоуровневое описание программы. Чтобы сделать процесс написания этой программы проще, создадим пару контейнеров «Мой блок». В первом номер цвета должен приниматься в качестве входного параметра, название цвета должно быть выдано в виде текстового значения. Это может быть полезно как для отображения названия цвета на экране модуля, так и для произнесения его названия с помощью блока **Звук** (Sound). Во втором контейнере «Мой блок» к количеству объектов определенного цвета прибавляется единица. Этот шаг довольно прост, однако следует использовать пять блоков, занимающих довольно много места в области программирования. Таким образом, использование контейнера «Мой блок» сделает основную программу более компактной и простой для восприятия.

Листинг 15.1. *Высокоуровневое описание программы* ColorCount

```
create the ColorCounts variable as an array with
   eight 0s
display the eight color names and the starting
   count (0)
begin loop
   wait for the Center button to be bumped
   read the Color Sensor
   use a Sound block to say the name of the color
   add one to the count for the color
   display the new count for the color
loop forever
```

Контейнер ColorToText

Используя блок **Переключатель** (Switch), можно преобразовать числовое показание датчика цвета в соответствующее название цвета. Этот блок довольно прост, но довольно большой, поскольку для обработки всех возможных цветовых значений потребуется восемь случаев. Кроме того, необходимо будет дважды повторить в программе этот код — еще один повод превратить его в контейнер «Мой блок».

Для создания контейнера **ColorToText** сначала была написана программа *ColorToTextBuilder*, показанная на рис. 15.15. Контейнер «Мой блок» создается из блока **Переключатель** (Switch) и блоков, содержащихся внутри него. Блок **Константа** (Constant) в начале и блок **Переменная** (Variable) в конце присутствуют только для добавления шин данных, которые превратятся в параметры контейнера **ColorToText**.

В блоке **Переключатель** (Switch) число принимается в качестве входного параметра, а также выбирается соответствующий случай. В каждом случае содержится блок **Константа** (Constant), с помощью которого соответствующее (текстовое) название цвета помещается в шину данных; единственное различие между этими случаями — текстовое значение, заданное в блоке **Константа** (Constant).

Для создания контейнера «Мой блок», выдели блок **Переключатель** (Switch) и выбери команду меню

Рис. 15.15. Программа ColorToTextBuilder

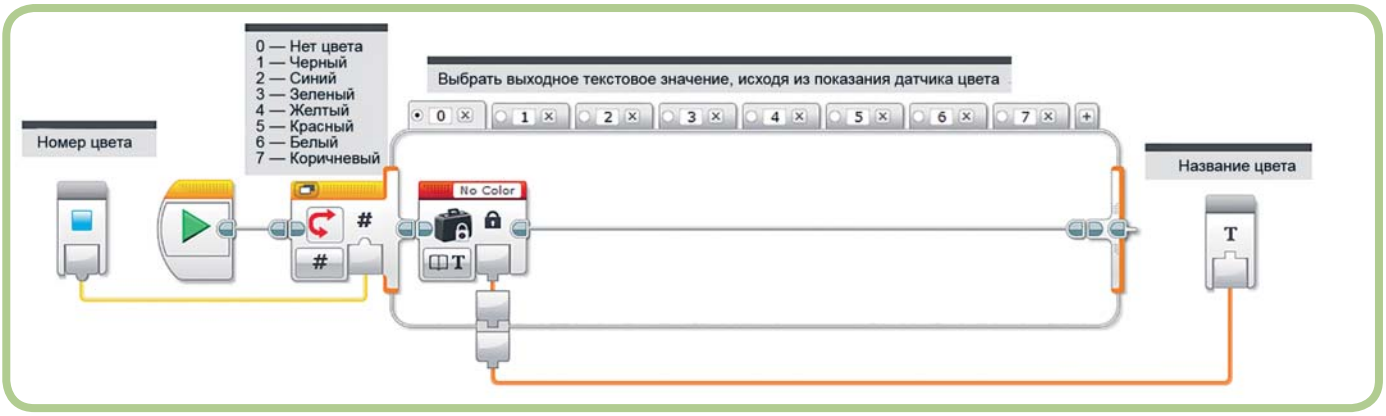


Рис. 15.16. Контейнер ***ColorToText***

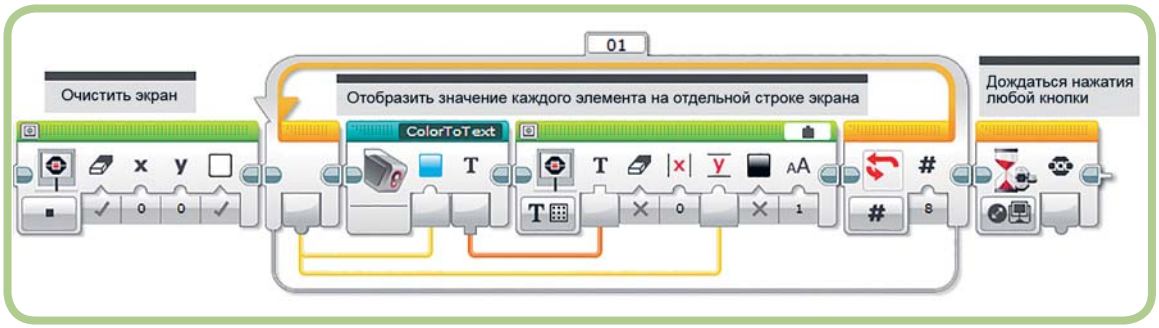


Рис. 15.17. Программа ColorToTextTest

Инструменты (Tools) ⇒ **Конструктор Мой блок** (My Block Builder). Задай имена и значки для двух параметров (например, *Color Number* и *Text*). На рис. 15.16 показан готовый контейнер с несколькими комментариями.

После создания этого контейнера тщательно протестируй его, чтобы убедиться в правильном подключении шин данных и в том, что блок выдает правильное название для каждого цветового значения.

Программа *ColorToTextTest* (рис. 15.17) очищает экран модуля EV3, после чего на нем отображается название каждого цвета на отдельной строке. Блок **Цикл** (Loop) повторяется восемь раз, при этом значение **Параметр цикла** (Loop Index) передается на ввод блока **ColorToText** для получения названий всех восьми цветов, а также на ввод блока **Экран** (Display) для задания номера строки для каждого цвета. При первом выполнении цикла значением **Параметр цикла** (Loop Index) будет 0, поэтому контейнер **ColorToText** должен выдать значение No Color, отображаемое на нулевой строке (в самом верху экрана). При следующем выполнении цикла значением **Параметр цикла** (Loop Index) будет 1, и на первой строке должно отобразиться значение Black. Цикл будет повторяться до тех пор, пока названия всех восьми цветов не отобразятся на экране модуля. В конце в блоке **Ожидание** (Wait) включается режим **Кнопки управления модулем** (Brick Buttons) ⇒ **Изменить** (Change), чтобы дождаться нажатия любой кнопки. Благодаря этому ты сможешь ознакомиться с содержимым экрана перед завершением программы.

После запуска программы на экране модуля EV3 должно отобразиться следующее:

```
No Color
Black
Blue
Green
Yellow
Red
White
Brown
```

Если какой-либо цвет отсутствует или цвета идут в неправильном порядке, для устранения проблемы отредактируй контейнер **ColorToText**.

Контейнер AddColorCount

В переменной ColorCounts хранится массив, содержащий количество обнаруженных объектов каждого цвета. При обнаружении очередного объекта того или иного цвета мы должны прибавить 1 к значению соответствующего элемента массива. Чтобы прочитать и обновить значение переменной, понадобится пять блоков, которые мы объединим в контейнер «Мой блок». В результате программа будет более компактной и удобной для восприятия (процесс создания контейнера «Мой блок» описан в гл. 12).

*Рис. 15.18. Контейнер **AddColorCount***

На рис. 15.18 показан готовый контейнер **AddColorCount**, в котором в качестве входного параметра принимается номер цвета, а также к значению соответствующего элемента массива ColorCounts прибавляется 1. Мы должны будем выбрать новое значение в основной программе для обновления содержимого экрана, поэтому данный контейнер передает это значение в качестве выходного параметра.

В первом блоке **Переменная** (Variable) считывается значение переменной ColorCounts и весь массив помещается в шину данных. В блоке **Операции над массивом** (Array Operations) в качестве входных данных принимается массив и номер цвета, а выдается текущее количество объектов данного цвета. Блоком **Математика** (Math) к этому количеству прибавляется 1, а следующим блоком **Операции над массивом** (Array Operations) в массив записывается новое значение. Затем новое значение массива в переменной ColorCounts сохраняется в последнем блоке **Переменная** (Variable).

Использование шины данных для выбора звукового файла

При распознавании цвета программой используется блок **Звук** (Sound) для того, чтобы произнести название этого цвета. Мы можем выбрать соответствующий каждому цвету звуковой файл для воспроизведения, передав название цвета (из контейнера **ColorToText**) блоку **Звук** (Sound) с помощью шины данных (как показано на рис. 15.19). Для этого необходимо выбрать вариант **Проводной** (Wired) в поле для имени файла (рис. 15.20), а затем подключить шину данных к параметру **Имя файла** (File Name).

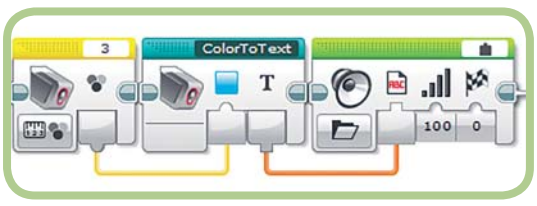


Рис. 15.19. Использование шины данных для выбора звукового файла

При выборе звукового файла с помощью шины данных необходимо убедиться в том, что он хранится в модуле EV3. Когда ты выбираешь звуковой файл вручную, в программном обеспечении EV3 эта задача решается автоматически. Это означает, что мы должны добавить восемь необходимых для данного проекта файлов и убедиться в том, что имена звуковых файлов соответствуют названиям цветов, переданных через шину данных из блока **ColorToText**.

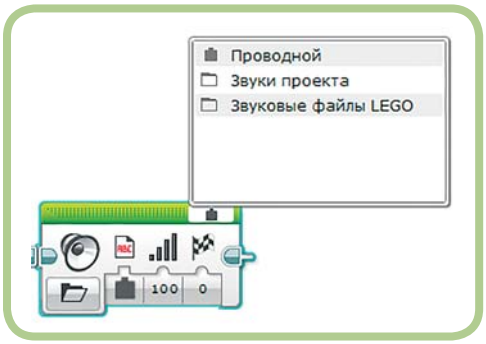


*Рис. 15.20. Блок **Звук** (Sound) с выбранным вариантом **Проводной** (Wired) в поле для имени файла*

Список включенных в проект звуковых файлов можно увидеть на вкладке **Звуки** (Sounds) страницы **Свойства проекта** (Project Properties) (рис. 15.21). Для добавления звукового файла в этот список достаточно включить в программу блок **Звук** (Sound) и выбрать нужный файл. Этот звуковой файл сохранится в списке звуков твоего проекта даже после выбора другого звукового файла или удаления блока **Звук** (Sound).

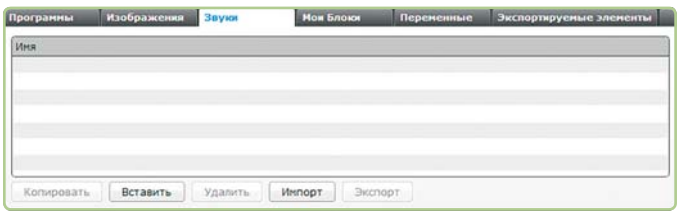


*Рис. 15.21. Список звуковых файлов на странице **Свойства проекта** (Project Properties) остается пустым до тех пор, пока ты не используешь для выбора файла блок **Звук** (Sound)*

Выполни следующие действия, чтобы добавить в проект названия семи цветов:

1. Создай новую программу с именем *ColorCount*.

2. Добавь в программу блок **Звук** (Sound).

3. Выбери вариант **Black** в поле для имени файла (**Звуковые файлы LEGO** (LEGO Sound Files) ⇒ **Цвета** (Colors)).

4. Выбери вариант **Blue** в поле для имени файла (в том же блоке **Звук** (Sound)).

5. Продолжай менять имя файла, пока не будут выбраны все семь цветов.

Список звуковых файлов теперь должен выглядеть так, как показано на рис. 15.22. Убедись в том, что у тебя в списке присутствуют все семь файлов.

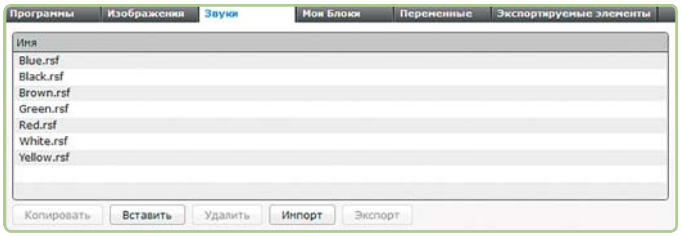


Рис. 15.22. Добавление в список звуковых файлов с названиями семи цветов

Теперь рассмотрим случай **Нет цвета** (No Color). Звукового файла No Color не существует, поэтому мы его создадим. В программе *RedOrBlue* мы использовали звук Uh-oh, когда датчик цвета не мог распознать цвет. Для использования этого звукового файла в программе *ColorCount* можно переименовать его в *No Color*, чтобы имя файла соответствовало тексту, переданному из блока **ColorToText**. Далее перечислены действия для создания звукового файла No Color из файла Uh-oh:

1. Выбери вариант **Uh-oh** в поле для имени файла блока **Звук** (Sound) (**Звуковые файлы LEGO** (LEGO Sound Files) ⇒ **Выражения** (Expressions)).

2. Открой страницу **Свойства проекта** (Project Properties).

3. Выбери пункт *Uh-oh.rsf* в списке звуковых файлов.

4. Щелкни по кнопке **Экспорт** (Export). Откроется диалоговое окно, позволяющее сохранить звуковой файл. Измени имя с *Uh-oh.rsf* на *No Color.rsf* (не забудь поставить пробел).

5. Щелкни по кнопке **Импорт** (Import). Откроется диалоговое окно, в котором можно выбрать файл для добавления в проект. Выбери *No Color.rsf*.

6. На странице **Свойства проекта** (Project Properties) выбери *Uh-oh.rsf* и щелкни по кнопке **Удалить** (Delete).

7. Удали из программы блок **Звук** (Sound).

Страница **Свойства проекта** (Project Properties) теперь должна выглядеть так, как показано на рис. 15.23. Конечно, ты можешь использовать другой звуковой файл, например Boo, Sorry, No или один из звуковых сигналов. Кроме того, с помощью инструмента **Редактор звука** (Sound Editor) ты можешь создать собственный звуковой файл. При этом важно задать для файла имя No Color, которое соответствует текстовому значению, используемому контейнером **ColorToText**.

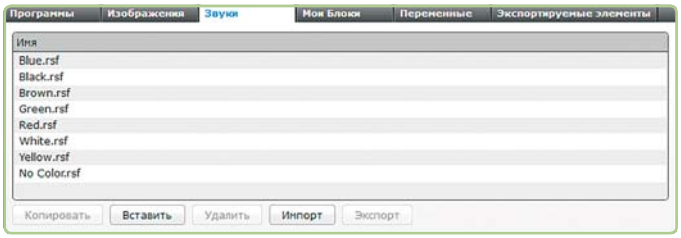


Рис. 15.23. Полный список звуковых файлов, необходимых программе ColorCount

Инициализация

Теперь мы можем приступить к написанию программы. Сначала необходимо создать массив ColorCounts, содержащий восемь элементов, которые инициированы 0, а также обеспечить отображение названий цветов и начальных значений.

Блоки, отвечающие за эту инициализацию, показаны на рис. 15.24. Вот как работает этот раздел кода:

1. В блоке **Переменная** (Variable) в режиме **Записать** (Write) ⇒ **Числовой массив** (Numeric Array) создается переменная ColorCount. В качестве значения параметра **Значение** (Value) задано **[0;0;0;0;0;0;0;0]**. В результате создается массив с восемью элементами, значения которых изначально равны нулю. Процесс задания начального значения массива описан в разделе «Создание массива».

2. С помощью блока **Экран** (Display) очищается экран модуля.

3. Блок **Цикл** (Loop) повторяется восемь раз, по одному разу для каждого цвета.

4. Блоком **ColorToText** в качестве входных данных принимается значение **Параметр цикла** (Loop Index), при этом выдается соответствующее название цвета.

5. С помощью первого блока **Экран** (Display) отображается название цвета, полученное от блока **ColorToText** с использованием значения **Параметр цикла** (Loop Index) для указания номера строки. Для параметра **Шрифт** (Font) задано значение **1**, чтобы на экране модуля отобразились все восемь цветов.

6. Вторым блоком **Экран** (Display) отображается **0**, при этом также используется значение **Параметр цикла** (Loop Index) для управления параметром **Строка** (Row). Для параметра **Столбец** (Column) задано значение **10**, чтобы значения выстроились в аккуратный столбец.

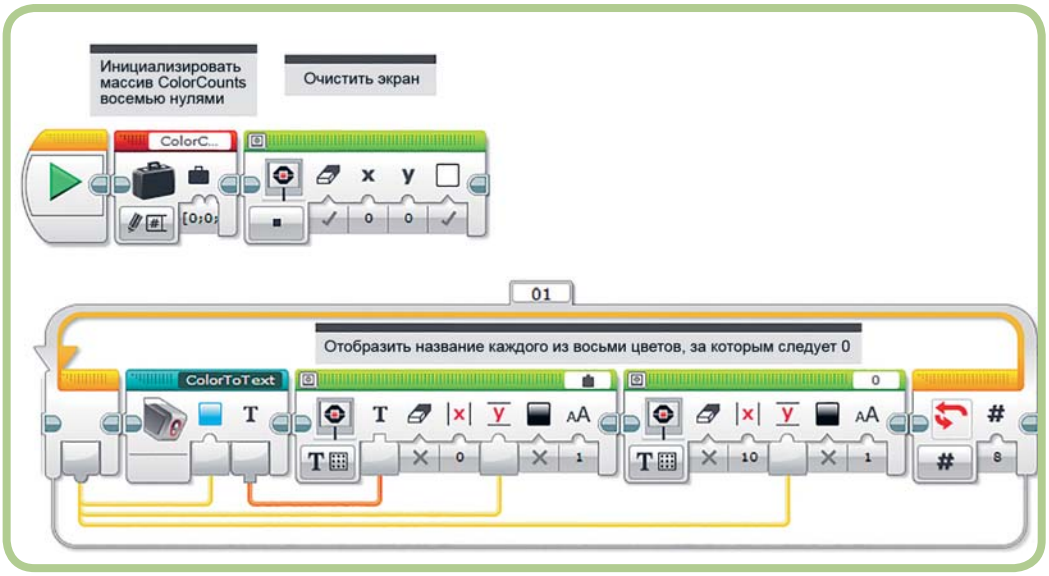


Рис. 15.24. Программа ColorCount, часть 1-я

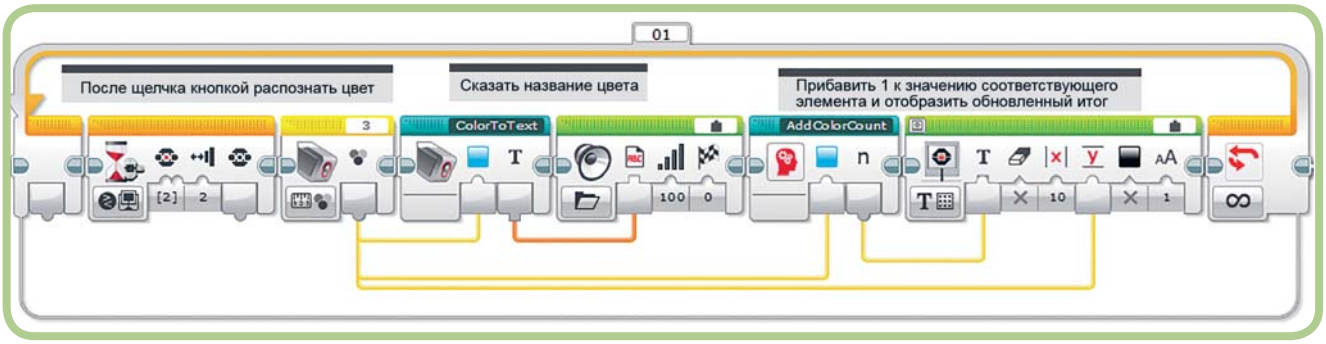


Рис. 15.25. Программа ColorCount, часть 2-я

После запуска этой части программы на экране модуля EV3 должно отобразиться следующее:

```
No Color    0
Black       0
Blue        0
Green       0
Yellow      0
Red         0
White       0
Brown       0
```

Благодаря тому, что значения отображаются в столбце 10, а не сразу после названия цвета, мы получаем удобный для восприятия список информации. Это упрощает процесс обновления содержимого экрана, в результате можно обновить количество цветных объектов, используя значение 10 для параметра **Столбец** (Column) и номер цвета для параметра **Строка** (Row). Для этого требуется гораздо меньше кода по сравнению с обновлением всего списка при каждом изменении значения.

Если на данном этапе ты захочешь протестировать программу, тебе понадобится добавить временный блок **Ожидание** (Wait) в конце. Если этого не сделать, программа завершится, и экран очистится прежде, чем ты успеешь ознакомиться с результатом.

Подсчет цветных объектов

Теперь мы готовы написать код для подсчета цветных объектов (рис. 15.25). Вот как он работает:

1. Блок **Цикл** (Loop) повторяется до тех пор, пока ты не завершишь программу.

2. В блоке **Ожидание** (Wait) используется режим **Кнопки управления модулем** (Brick Buttons) ⇒ **Сравнение** (Compare), поэтому программа находится в режиме ожидания щелчка кнопкой «Центр».

3. Блоком **Датчик цвета** (Color Sensor) в режиме **Измерение** (Measure) ⇒ **Цвет** (Color) определяется цвет объекта, находящегося перед датчиком. Это значение используется в следующих блоках тремя способами.

4. В контейнере **ColorToText** преобразуется номер цвета в соответствующее название цвета.

5. В блоке **Звук** (Sound) используется название цвета для выбора звукового файла, которое требуется воспроизвести.

6. С помощью контейнера **AddColorCount** к количеству объектов того или иного цвета прибавляется 1, а также в качестве выходного параметра выдается новое значение, которое отправляется в блок **Экран** (Display).

7. Блоком **Экран** (Display) отображается обновленное количество из контейнера **AddColorCount**. Номер цвета используется в качестве значения параметра **Строка** (Row), а для параметра **Столбец** (Column) задано значение **10**. Это приводит к перезаписи предыдущего значения количества объектов распознанного цвета.

После запуска программы на экране модуля должен отобразиться список цветов, за которыми следуют нули. После размещения объекта перед датчиком цвета и щелчка кнопкой «Центр» программа должна произнести название цвета, а значение измениться с 0 на 1. Продолжай тестировать разные объекты, пока не услышишь названия всех цветов.

Программа MemoryGame

Программа *MemoryGame* представляет собой простую игру в стиле «Саймон говорит», в которой применяются индикатор состояния модуля и кнопки. Индикатор состояния модуля загорается разными цветами в случайной последовательности, после чего пользователь нажимает кнопки, пытаясь повторить последовательность вспышек. Если ответ правильный, игра продолжается с более длинной последовательностью, в противном случае игра заканчивается.

Вся программа находится в блоке **Цикл** (Loop), который повторяется до тех пор, пока игрок не введет неправильный ответ. Мы создаем массив с именем Lights для хранения последовательности цветов, соответствующих значениям индикатора состояния модуля (0 = зеленый, 1 = оранжевый, 2 = красный). При каждом выполнении цикла мы используем блок **Случайное значение** (Random) для выбора случайного значения от 0 до 2, которое нужно добавить в массив Lights. По мере заполнения массива индикатор мигает соответствующим светом. Затем пользователь нажимает кнопки модуля в соответствии с последовательностью вспышек, используя кнопку «Влево» для зеленого, кнопку «Центр» — для оранжевого и кнопку «Вправо» — для красного цвета.

Рис. 15.26. Отключение подсветки индикатора состояния модуля перед запуском цикла

Программа сначала должна отключить подсветку индикатора состояния модуля. Если этого не произойдет, останется мигающий зеленый свет, что может привести к путанице. Затем запускается основной цикл программы (рис. 15.26).

Запуск цикла

Первая часть цикла показана на рис. 15.27. Значение **Параметр цикла** (Loop Index) используется программой для определения того количества значений, которые необходимо добавить в массив Lights. Однако поскольку значение этого параметра начинается с 0, а мы хотим начать игру с одним значением в массиве Lights, прибавим 1 к значению **Параметр цикла** (Loop Index) и используем его для указания количества элементов массива.

При каждом выполнении цикла программа использует контейнер **DisplayNumber** (см. гл. 12), чтобы отобразить количество элементов последовательности в виде “Level 1” («Уровень 1»), “Level 2” («Уровень 2») и т. д. Потом переменная Lights инициализируется пустым массивом. Затем программа выдает сообщение “Start” («Пуск») и делает короткую паузу, прежде чем запустить последовательность вспышек.

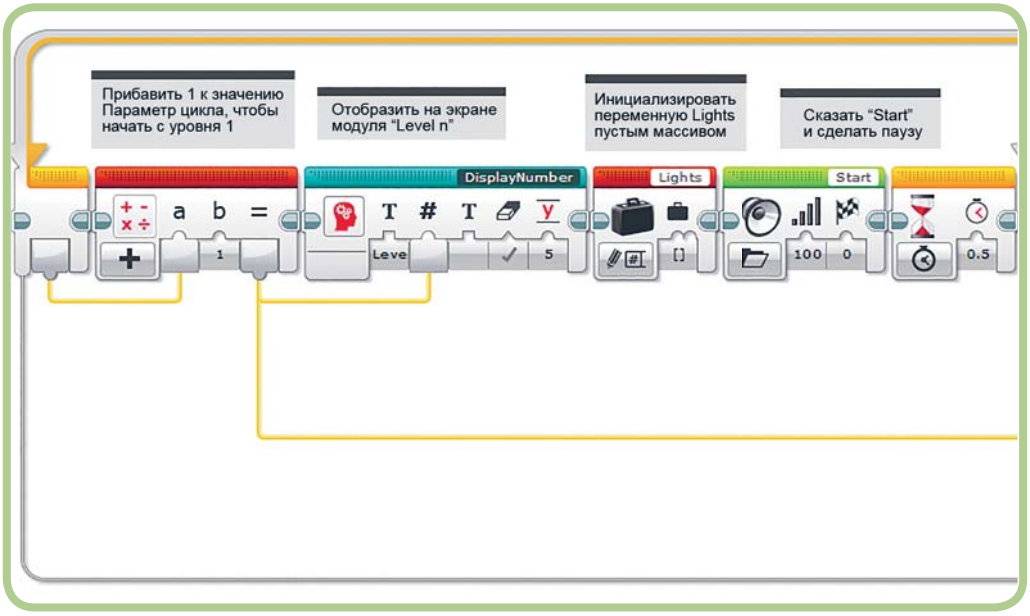


Рис. 15.27. Подготовка к выполнению цикла

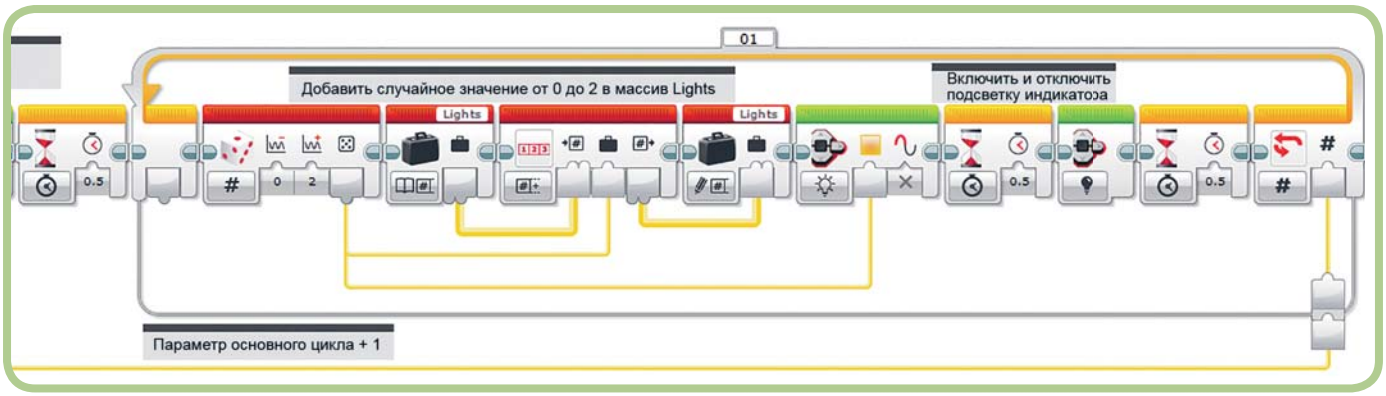


Рис. 15.28. Создание последовательности вспышек

*Рис. 15.29. Создание контейнера **WaitForButtons***

Создание последовательности вспышек

С помощью следующей части программы массив Lights заполняется случайными значениями, благодаря этому индикатор состояния модуля будет мигать в соответствии с ними (рис. 15.28). Значение из блока **Математика** (Math) на рис. 15.27 используется для задания количества повторов этого цикла и, следовательно, количества добавленных в массив элементов.

ПРИМЕЧАНИЕ **Чтобы нарисовать шину данных, идущую из блока Математика (Math) к блоку Цикл (Loop), нажимай на кнопку масштабирования (🔍) на панели инструментов, пока не появятся оба блока.**

При каждом выполнении цикла блоком **Случайное значение** (Random) генерируется число от 0 до 2. Это значение добавляется в массив Lights и используется для управления подсветкой индикатора состояния модуля. После короткой паузы подсветка индикатора отключается, затем следует еще одна пауза перед повторением или выходом из цикла. После завершения цикла массив Lights будет содержать все значения, соответствующие вспышкам индикатора состояния модуля. Эти значения пригодятся для проверки правильности ответа пользователя.

Контейнер WaitForButtons

Для последней части программы создадим контейнер **WaitForButtons** (рис. 15.29). Когда пользователь будет пытаться повторить последовательность световых вспышек с помощью кнопок модуля, благодаря этому контейнеру индикатор будет загораться соответствующим светом при нажатии каждой кнопки и выдавать соответствующее ей значение цвета подсветки. На рис. 15.30 видно, как этот контейнер будет выглядеть в твоей программе.

*Рис. 15.30. Контейнер **WaitForButtons***

В первом блоке **Ожидание** (Wait) используется режим **Кнопки управления модулем** (Brick Buttons) ⇒ **Сравнение** (Compare), из-за чего программа находится в режиме ожидания нажатия кнопки «Влево», «Центр» или «Вправо». На рис. 15.31 показано, как выбрать эти кнопки.

Обрати внимание на то, что мы используем идентификаторы кнопок от 1 до 3, в то время как номера цветов подсветки индикатора идут с 0 до 2. В блоке **Математика** (Math) из идентификатора кнопки вычитается 1, чтобы получить правильное значение для передачи блоку **Индикатор состояния модуля** (Brick Status Light). После включения подсветки второй блок **Ожидание** (Wait) находится в режиме ожидания до тех пор, пока нажатая кнопка не будет отпущена.

После того как кнопка будет отпущена, подсветка индикатора отключится. Контейнером «Мой блок» в качестве выходного параметра будет выдан номер, соответствующий

цвету подсветки индикатора (а не идентификатору кнопки). Это значение будет использовано в основной программе для того, чтобы убедиться в том, что пользователь нажал кнопку, соответствующую цвету подсветки, которая была включена в первой части программы.

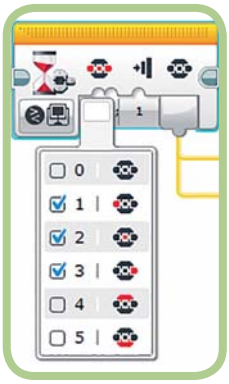


Рис. 15.31. Ожидание нажатия кнопки «Влево», «Центр» или «Вправо»

Проверка ответа пользователя

Последняя часть программы, показанная на рис. 15.32, используется для приема и проверки ответа пользователя. Первым делом она отображает сообщение “Go” («Вперед») с помощью блока **Звук** (Sound), и пользователь понимает, что пришло время повторить последовательность вспышек. Затем программа обходит массив, находясь в режиме ожидания, пока пользователь не нажмет кнопку, соответствующую значению цвета, хранящемуся в каждом его элементе, используя контейнер **WaitForButton**. С помощью блока **Сравнение** (Compare) можно определить, правильную ли кнопку нажал пользователь. Для этого необходимо проверить, соответствует ли цвет индикатора, связанный с нажатой кнопкой, значению, хранящемуся в массиве.

Если пользователь ошибется, будет выполнен код в блоке **Переключатель** (Switch), и программа выдаст сообщение “Game Over” («Игра окончена»), блоком **Прерывание цикла** (Loop Interrupt) осуществится выход из внутреннего блока **Цикл** (Loop), составляющим эту часть программы. Цикл называется *02*, что учтено в настройках блока **Прерывание цикла** (Loop Interrupt). Результат выполнения блока **Сравнение** (Compare) также учитывается основным блоком **Цикл** (Loop) для принятия решения о том, стоит ли продолжать его выполнение. Если пользователь не ошибся, это значение оказывается ложным, и основной цикл повторяется.

С помощью длины массива Lights задается количество повторений цикла. После завершения цикла в результате выполнения блока **Прерывание цикла** (Loop Interrupt) или вследствие ввода правильной последовательности во внешнем блоке **Цикл** (Loop) проверяется последнее выходное значение блока **Сравнение** (Compare). Если ответ пользователя был неверным, это значение оказывается истинным, осуществляется выход из цикла, и программа завершается. Если ответ был правильным, значение оказывается ложным, и цикл повторяется, выдавая новую, более длинную последовательность вспышек.

Запусти программу. Сначала отключится обычная мигающая зеленая подсветка, а затем индикатор состояния модуля

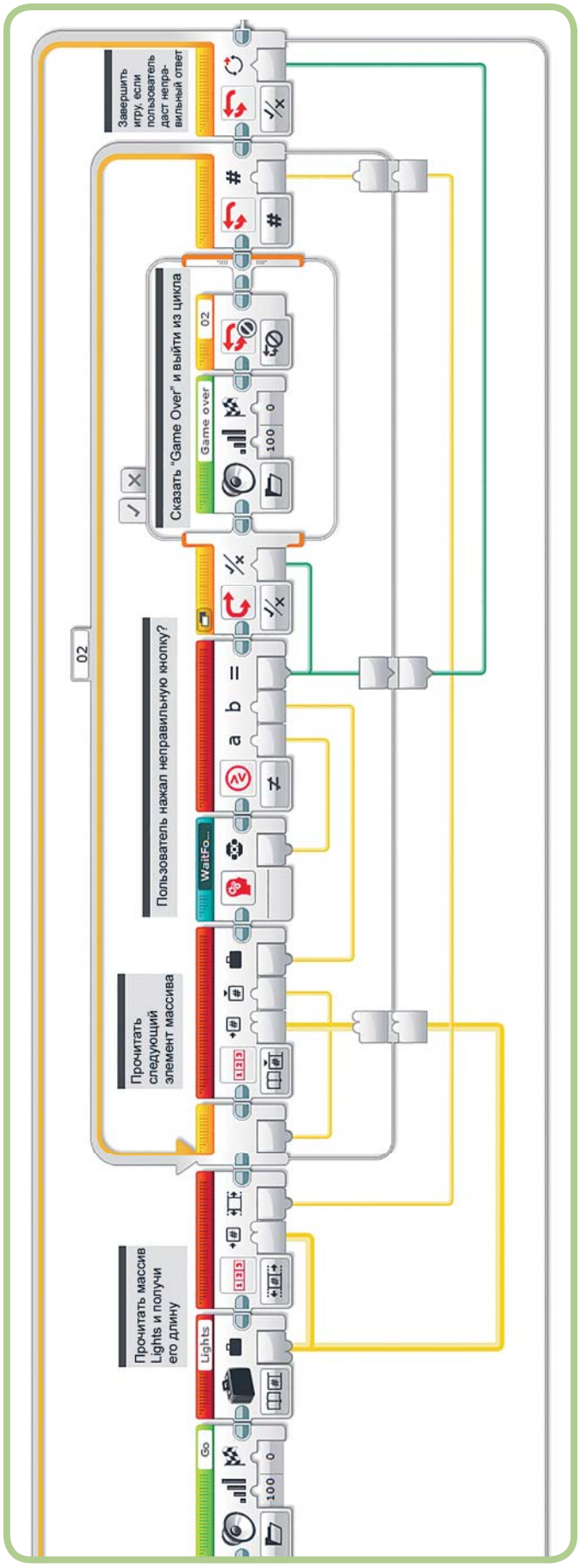


Рис. 15.32. Проверка ответа пользователя

мигнет один раз. Ответь на это, нажав соответствующую кнопку, и после этого индикатор мигнет два раза, после чего снова станет ждать твоего ответа. Программа будет работать до тех пор, пока ты не допустишь ошибку (или самостоятельно не завершишь программу).

Чтобы не перегружать программу, я не стал добавлять отображаемые на экране инструкции. Для того чтобы сделать программу удобнее для пользователя, ты можешь добавить блоки **Экран** (Display) для вывода инструкций в начале игры, а также подсказки, сообщающей пользователю о соответствии кнопок и цветов подсветки.

Дальнейшее исследование

Далее перечислены дополнительные упражнения для исследования представленных в этой главе концепций:

1. При работе с массивами часто возникают две ошибки: попытка прочитать значение несуществующего элемента и записать значение в массив по несуществующему индексу. Измени программу *ArrayTest*, чтобы понять, что происходит в этих случаях. Например, после заполнения массива попробуй прочитать значение элемента ArrayValue [10]. Затем попробуй записать значение по индексу ArrayValue [10] и посмотри, как это влияет на длину массива и хранящиеся в нем значения. Так, ты поймешь, как программное обеспечение реагирует на эти ошибки, и в дальнейшем сможешь обнаружить их.

2. Возможности программы *ButtonCommand* ограничены, поскольку она поддерживает только четыре команды. Ты можешь усовершенствовать ее, используя для выбора команды две кнопки. В исходной программе мы применяли идентификаторы кнопок для указания номеров команд. Для использования двух кнопок для указания команды необходимо преобразовать два идентификатора кнопок в один номер команды. Простой способ решения этой проблемы заключается в том, чтобы взять первый идентификатор кнопки и умножить его на 10, а затем прибавить второй идентификатор кнопки (рис. 15.33). В результате ты получишь номера команд между 11 («Влево», «Влево») и 55 («Вниз», «Вниз»). Не все числа между 11 и 55 будут соответствовать действительным командам, поскольку ты будешь использовать только числа, содержащие цифры 1, 3, 4 и 5.

   В этой версии программы тебе понадобится три столбца для отображения каждой команды: два — для отображения цифр, один — для пробела между командами.

3. Добавь звук в программу *MemoryGame*. Сделай так, чтобы при каждой вспышке индикатора состояния модуля воспроизводился тон, соответствующий ее цвету. Для этого создай массив, в котором будут храниться частоты для каждого цвета подсветки индикатора. Попробуй следующий набор значений: [261.626; 329.628; 391.995]. Эти ноты составляют аккорд до-мажор. Используй поисковый запрос «частоты музыкальных нот» для получения дополнительной информации о том, как частота тона соотносится с музыкальными нотами. Если назвать этот массив Tones, то элемент Tones [0] будет соответствовать зеленой подсветке, элемент Tones [1] — оранжевой, а элемент Tones [2] — красной.

Заключение

Из этой главы ты узнал о массивах EV3, с помощью которых можно хранить списки значений. Мы использовали блок **Переменная** (Variable) для создания и сохранения массивов, а также блок **Операции над массивом** (Array Operations) для получения доступа к элементам массива и определения его длины.

Программа *ButtonCommand* создала список команд для робота TriBot, который по сути представляет собой программу внутри программы. Благодаря массиву мы легко смогли расширить программу *RedOrBlueCount*, чтобы она обрабатывала все восемь цветов, распознаваемых датчиком цвета. С помощью программы *MemoryGame* ты можешь проверить память, используя индикатор состояния и кнопки модуля, а также все более длинные списки значений.

Массивы полезны, если ты хочешь использовать набор значений во время выполнения программы. В гл. 16 ты узнаешь о файлах, которые применяются для сохранения значений из программы, для загрузки значений из другой программы или с твоего компьютера.

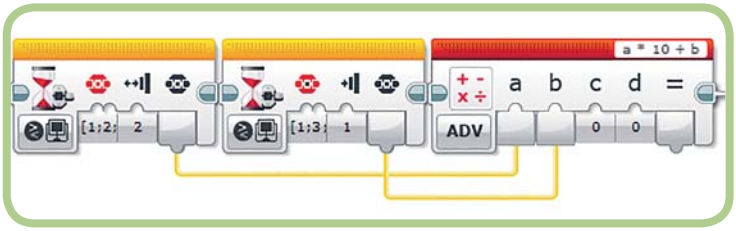


Рис. 15.33. Создание номера команды на основании нажатия двух кнопок

16

Файлы

В этой главе ты применишь блок **Доступ к файлу** (File Access) для создания и использования файлов в своих программах. Информация, хранящаяся в файле, *постоянна*, а это означает, что она остается доступной после завершения программы, даже в случае отключения модуля EV3. С помощью этих файлов ты можешь сохранять информацию из программы и позже использовать ее в этой же или в другой программе.

Сначала ты создашь нескольких тестовых программ, а затем изменишь программу *MemoryGame* из гл. 15 так, чтобы она сохраняла в файле лучший результат пользователя. Затем ты добавишь в программу *ColorCount* меню, позволяющее сохранить в файле количество объектов каждого цвета, а затем восстановить значения при следующем запуске программы. Ты также узнаешь, как управлять памятью модуля EV3, удалять файлы или перемещать их с модуля на компьютер и обратно.

Блок «Доступ к файлу»

Блок **Доступ к файлу** (File Access) находится на темно-синей вкладке палитры программирования, содержащей блоки дополнений, в нем предусмотрены четыре основных режима (рис. 16.1), с помощью которых ты можешь прочитать данные из файла, записать данные в файл, удалить файл и закрыть файл. Закрытие файла сообщает модулю EV3 о том, что пользователь закончил работу с ним. Для демонстрации этих операций создадим программу *FileTest*, с помощью которой в файл записываются три значения, а затем считываются и отображаются на экране модуля EV3.

Указание имени файла

В блоке **Доступ к файлу** (File Access) *всегда* нужно указывать имя файла, с которым ты собираешься работать. Для задания этого параметра можно щелкнуть по полю **Имя файла** (File Name) в правом верхнем углу блока или выбрать вариант **Проводной** (Wired), чтобы передать имя с помощью шины данных (рис. 16.2).

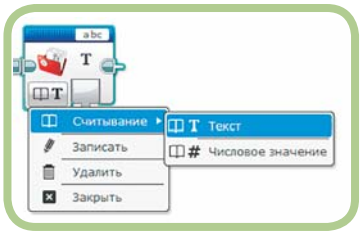


Рис. 16.1. Выбор режима блока ***Доступ к файлу***

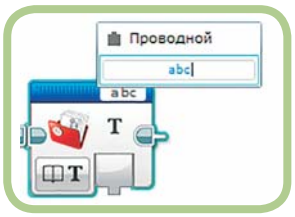


Рис. 16.2. Указание имени файла

Имена файлов чувствительны к регистру и могут содержать до 31 символа. Ты можешь использовать цифры, буквы, пробелы, а также символы нижнего подчеркивания (_) и тире (–). Старайся давать файлам описательные имена, которые позволяют получить представление об их содержимом.

Запись данных в файл

С помощью режима **Записать** (Write) можно сохранить информацию в существующем файле и создать новый файл, если он еще не существует. В блоке **Доступ к файлу** (File Access) новые данные всегда записываются в конец файла, поэтому, если файл уже существует, новое значение добавляется после него. Если ты хочешь заменить существующие данные, а не дополнить их, сначала придется удалить файл.

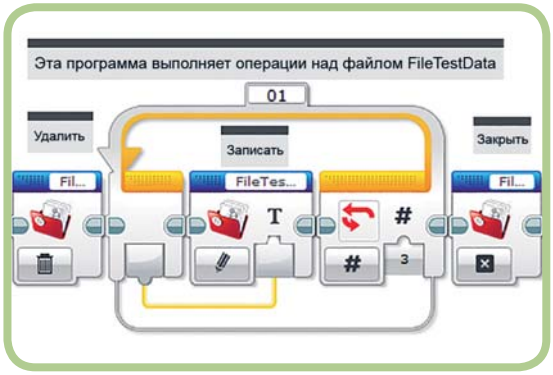


Рис. 16.3. Запись данных в файл

На рис. 16.3 показана первая часть программы *FileTest*, в которой используются три блока **Доступ к файлу** (File

Access) для записи в файл значений 0, 1 и 2. В качестве имени файла во всех трех блоках задано *FileTestData*.

Для того чтобы содержимое файла *FileTestData* заменялось при каждом запуске программы *FileTest*, в первом блоке **Доступ к файлу** (File Access) существует режим **Удалить** (Delete) для удаления уже существующего файла. Если опустить этот блок, то при первом запуске программы будет создан новый файл, содержащий три значения. Когда программа будет запущена во второй раз, откроется существующий файл, при этом в него добавятся еще три значения. При третьем запуске программы в этот файл будут добавлены еще три значения, таким образом в файле будет уже девять значений.

Во втором блоке **Доступ к файлу** (File Access) используется режим **Записать** (Write) для записи в файл значения **Параметр цикла** (Loop Index). Входной параметр блока **Доступ к файлу** (File Access) принимает текстовые значения, однако в режиме **Записать** (Write) тебе не нужно выбирать между вариантами **Числовое значение** (Numeric) и **Текст** (Text). Любые числовые значения, переданные в блок **Доступ к файлу** (File Access) в качестве входных данных, автоматически преобразуются в текст перед записью в файл.

При первом выполнении цикла создается файл, в который записывается первое значение (0). При следующих двух повторах цикла в этот файл добавляются значения 1 и 2. В третьем блоке **Доступ к файлу** (File Access) используется режим **Закрыть** (Close) для закрытия файла. После записи значений следует закрывать файл, чтобы при чтении данных в следующем разделе программы считывались значения с самого начала файла.

Чтение данных из файла

В режиме **Считывание** (Read) блока **Доступ к файлу** (File Access) можно считать числовые или текстовые значения из существующего файла. Если файл не существует, выполнение программы прерывается, а на экране модуля EV3 появляется сообщение об ошибке чтения файла.

В режиме **Считывание** (Read) необходимо выбрать тип данных выходного параметра, чтобы использовать шину данных правильного типа. Данные файла всегда хранятся в виде текста, поэтому ты можешь прочитать буквы и цифры как текст, используя режим **Считывание** (Read) ⇒ **Текст** (Text). Если ты точно знаешь, что в файле записано число, как в случае с *FileTestData*, используй режим **Считывание** (Read) ⇒ **Числовое значение** (Numeric) для получения числового значения. Это сработает только в том случае, если значение в файле *на самом деле* является числом. Если это не так, значение будет считано как 0, при этом никакого сообщения о том, что это значение не соответствует реальному содержимому файла, ты не увидишь.

На рис. 16.5 показана вторая половина программы *FileTest*, с помощью которой считываются и отображаются на экране модуля EV3 три значения из файла *FileTestData*. Посредством первого блока **Экран** (Display) очищается экран, и цикл повторяется три раза. При каждом выполнении цикла блоком **Доступ к файлу** (File Access) считывается и отображается число. После окончания цикла файл закрывается, и программа приостанавливается на пять секунд, чтобы дать тебе возможность ознакомиться с содержимым экрана. Хорошей практикой является закрытие файла после окончания работы с ним, поэтому программа *FileTest* содержит блок **Доступ к файлу** (File Access), который закрывает файл *FileTestData*.

ПРЕДОТВРАЩЕНИЕ ОШИБОК, СВЯЗАННЫХ С ИМЕНАМИ ФАЙЛОВ

При использовании одного и того же файла несколькими блоками **Доступ к файлу** (File Access), проверь, чтобы в каждом из этих блоков было введено правильное имя. Например, в программе, показанной на рис. 16.3, произойдет сбой, если во всех трех блоках **Доступ к файлу** (File Access) не будет указано одно и то же имя файла. Как правило, в блоке **Доступ к файлу** (File Access) видно только начало имени файла, однако для отображения полного имени достаточно навести указатель мыши на поле с именем файла (рис. 16.4). Используй этот прием, чтобы быстро проверить каждый из блоков **Доступ к файлу** (File Access) в своей программе и убедиться в том, что используется нужный файл.

Рис. 16.4. Отображение полного имени файла

Помимо букв и цифр в именах можно использовать тире и символ нижнего подчеркивания (– и _). Если ввести в поле **Имя файла** (File Name) другие специальные символы, например * и %, при открытии этого файла они будут заменены на пробел. По этой причине, если у тебя есть два блока **Доступ к файлу** (File Access) с именами файлов *Test*One* и *Test%One*, то оба блока будут использовать файл *Test One*. Скорее всего, это будет противоречить твоей задумке. Кроме того, не стоит забывать о том, что имена файлов чувствительны к регистру; *FileTestData*, *filetestdata* и *FILETESTDATA* — это разные файлы. Если ты запишешь значения в файл *FileTestData*, а попытаешься прочитать их из файла *filetestdata*, — произойдет сбой.

Один из способов предотвращения ошибок при вводе имени файла заключается в копировании существующего блока **Доступ к файлу** (File Access) вместо добавления нового. Например, для создания кода, показанного на рис. 16.3, можно сначала добавить блок для удаления файла и задать имя файла. Затем после добавления блока **Цикл** (Loop) скопируй первый блок **Доступ к файлу** (File Access), перетащив его в блок **Цикл** (Loop), удерживая нажатой клавишу **Ctrl**. Тебе потребуется изменить режим в блоке-копии с **Удалить** (Delete) на **Записать** (Write), однако в нем уже будет задано правильное имя файла.

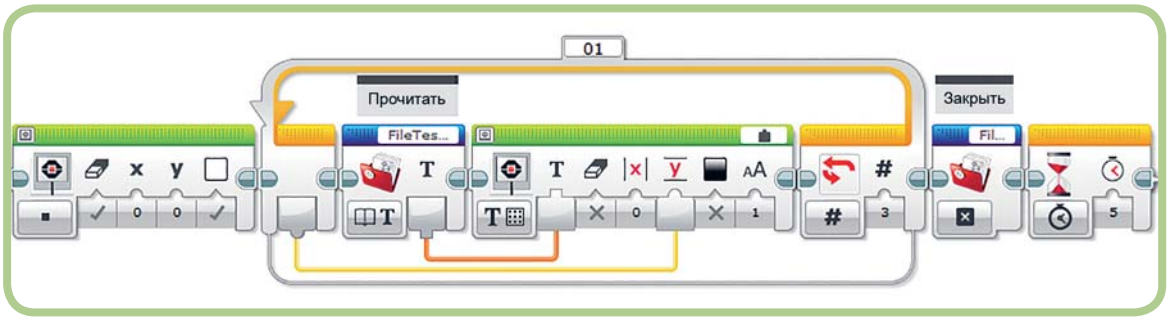


Рис. 16.5. Чтение данных из файла

Добавь блоки, изображенные на рис. 16.5, в конец программы, показанной на рис. 16.3, чтобы завершить создание программы *FileTest*. После запуска готовой программы должен быть создан файл *FileAccessTest*; в него должны быть записаны значения 0, 1 и 2; после чего значения считываются и выводятся на экран модуля EV3.

Сохранение рекорда в программе MemoryGame

В этом разделе добавим в программу *MemoryGame* из гл. 15 код для сохранения лучшего результата пользователя. Как ты помнишь, эта программа содержится в большом цикле, который завершается при неправильном ответе пользователя. При каждом выполнении цикла значение **Параметр цикла** (Loop Index) будет соответствовать количеству правильных ответов игрока, поэтому мы используем его в качестве рекордного значения.

Рекорд сохраняется в файле *MG_HighScore*. Когда пользователь дает неправильный ответ, основной цикл завершается, и программа сравнивает текущий счет со значением, хранящимся в файле *MG_HighScore*. Если пользователю удалось побить предыдущий рекорд, в программе сохранится новый результат, на экране модуля появится сообщение с поздравлением, а также раздадутся аплодисменты.

Рис. 16.6. Сохранение результата пользователя

Поместим большую часть нового кода в конец программы, однако нам нужно кое-что изменить в самом начале основного цикла. Чтобы отслеживать результат пользователя, мы сохраним значение **Параметр цикла** (Loop Index) в блоке **Переменная** (Variable) с именем *Score* (рис. 16.6). Когда цикл завершится, благодаря значению переменной *Score* мы узнаем количество правильных ответов пользователя.

Остальная часть нового кода следует за основным циклом. В первой части, показанной на рис. 16.7, происходит извлечение предыдущего рекордного значения, которое сравнивается с текущим счетом пользователя. Во всех блоках **Доступ к файлу** (File Access) в этой программе используется файл *MG_HighScore*. Рассмотрим каждый из них по порядку.

1. В первом блоке **Доступ к файлу** (File Access) значение 0 записывается в файл *MG_HighScore*. Добавление значения 0 в существующий файл не влияет на программу, однако этот блок находится в ее начале на случай, если файла *MG_HighScore* еще нет, и нам требуется его создать. В противном случае при попытке прочитать данные из несуществующего файла могла возникнуть ошибка.

2. С помощью второго блока закрывается файл, чтобы следующему блоку было доступно его содержимое.

3. Третьим блоком **Доступ к файлу** (File Access) из файла считывается предыдущее рекордное значение. Это будет либо 0, записанный первым блоком **Доступ к файлу** (File Access), если файл *MG_HighScore* на тот момент еще не существовал, либо предыдущий лучший результат.

4. С помощью следующего блока файл закрывается, поскольку мы закончили работу с ним. Обрати внимание на то, что 0, записанный первым блоком **Доступ к файлу** (File Access), никогда не читается, если файл уже существует.

5. Блоком **Переменная** (Variable) считывается результат пользователя.

В блоке **Сравнение** (Compare) проверяется счет пользователя. Если он превышает текущий рекорд, значение, содержащееся в шине данных истинно, если нет — ложно.

После выполнения этих блоков две шины данных будут содержать результат пользователя и логическое значение, обозначающее, является ли текущий результат новым рекордом.

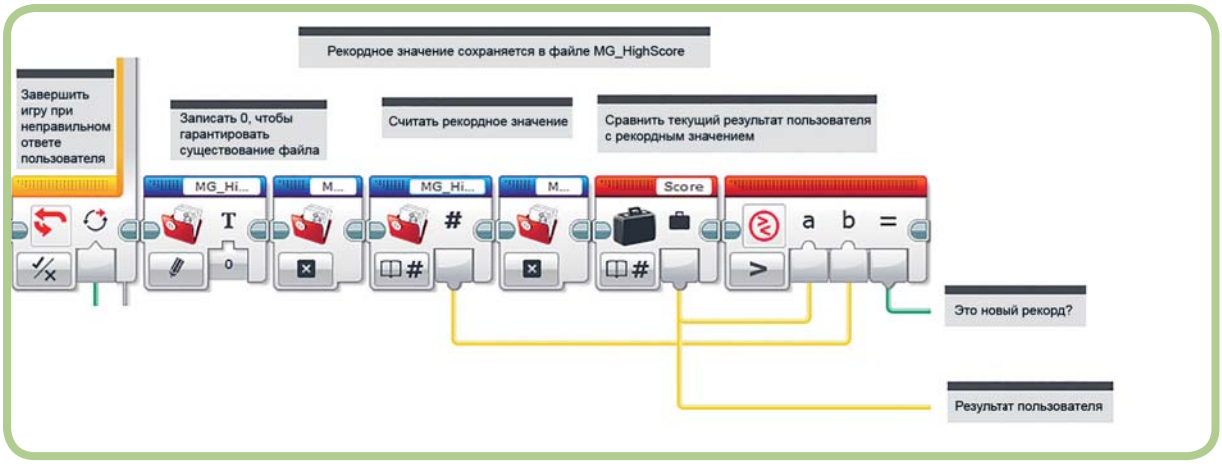


Рис. 16.7. Сравнение результата пользователя с предыдущим рекордом

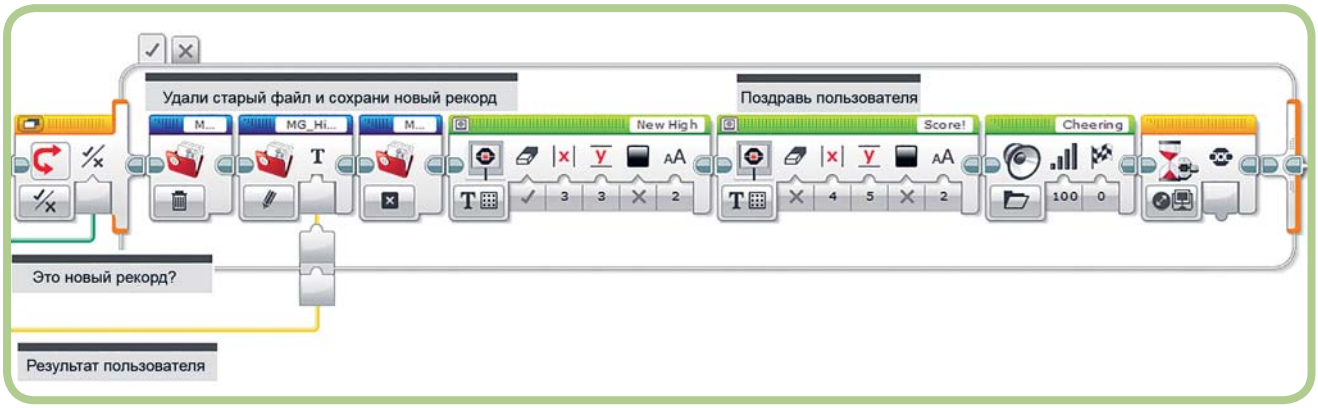


Рис. 16.8. Сохранение нового рекорда в случае «Истина» блока ***Переключатель***

Если это так, следующая часть программы, показанная на рис. 16.8, сохраняет новый рекорд и поздравляет пользователя.

Вот как работает эта часть программы:

6. Если логическое значение из блока **Сравнение** (Compare) истинно, в блоке **Переключатель** (Switch) выполняются блоки, показанные на рис. 16.8, для обновления лучшего результата.

7. С помощью первого блока **Доступ к файлу** (File Access) удаляется файл *MG_HighScore*, чтобы новое рекордное значение заменило старое.

8. В следующем блоке **Доступ к файлу** (File Access) записывается в файл новый лучший результат.

9. С помощью третьего блока **Доступ к файлу** (File Access) файл закрывается.

10. Благодаря двум блокам **Экран** (Display) большими буквами на двух строках примерно в центре экрана модуля отображается фраза "New High Score!" («Новый рекорд!»).

11. С помощью блока **Звук** (Sound) воспроизводится звук **Cheering**.

12. В блоке **Ожидание** (Wait) используется режим **Кнопки управления модулем** (Brick Buttons) ⇒ **Изменить** (Change) для приостановки выполнения программы до тех пор, пока пользователь не нажмет одну из кнопок.

13. Если при первом запуске обновленной программы *MemoryGame* ты дашь хотя бы один правильный ответ, то установишь новый рекорд, увидишь соответствующее сообщение и услышишь аплодисменты. После этого такое сообщение будет отображаться только в том случае, если ты побьешь предыдущий рекорд.

Помни о том, что перед тем, как прочитать предыдущий рекорд, мы записываем дополнительный 0 в файл *MG_HighScore* на случай, если этого файла не существует. Если пользователь установит новый рекорд, этот файл будет перезаписан, а дополнительный 0 исчезнет. Однако с течением времени побить рекорд будет все сложнее; мы просто будем записывать дополнительные нули, не заменяя файл, поэтому его объем станет излишне большим. Для гарантии того, что по завершении программы файл *MG_HighScore* будет содержать только одно значение, мы можем перезаписывать этот файл каждый раз, когда игра заканчивается (рис. 16.9).

Просто добавь новую шину данных для передачи текущего рекордного значения в блок **Переключатель** (Switch).

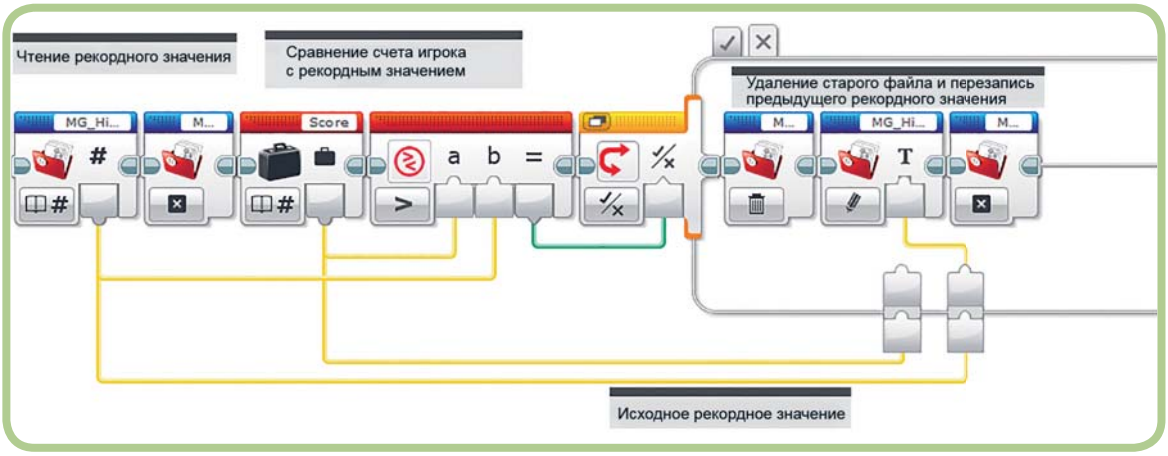


Рис. 16.9. Перезапись рекордного значения в случае «Ложь» блока ***Переключатель*** *(Switch)*

С помощью трех блоков **Доступ к файлу** (File Access) в случае «Ложь» блока **Переключатель** (Switch) существующий файл будет удален, запишется исходный рекорд, а файл закроется.

Программа FileReader

Возможность видеть то, что программа фактически записывает в файл, может оказаться весьма полезной. В этом разделе рассмотрим программу *FileReader* (рис. 16.10), с помощью которой содержимое файла отображается на экране модуля EV3. Это очень удобно, если в твоей программе используются короткие файлы. В данном примере в качестве имени файла задано *FileTestData*, однако ты можешь задать имя любого файла, содержимое которого хочешь прочитать.

При каждом выполнении цикла с помощью блока **Доступ к файлу** (File Access) из файла считывается значение в виде текста. Блок **Цикл** (Loop) настроен на бесконечное повторение, поскольку нам заранее неизвестно количество значений в файле, однако мы не хотим, чтобы цикл повторялся вечно. Нам нужен способ, при котором мы можем понять, достигли ли конца файла.

После того как все значения в файле будут прочитаны, блоком **Доступ к файлу** (File Access) в режиме **Считывание** (Read) ⇒ **Текст** (Text) будет возвращена пустая строка. Если файл не содержит пустых строк, мы можем выполнить проверку на наличие пустой строки и использовать ее результат в качестве доказательства того, что все данные прочитаны. В блоке **Сравнение** (Compare) используются только числа, поэтому вместо него используем блок **Переключатель** (Switch) в режиме **Текст** (Text). В верхнем случае задано значение «», соответствующее пустой строке. Когда с помощью блока **Доступ к файлу** (File Access) отражается пустая строка, выполняется код в верхнем случае, после чего благодаря

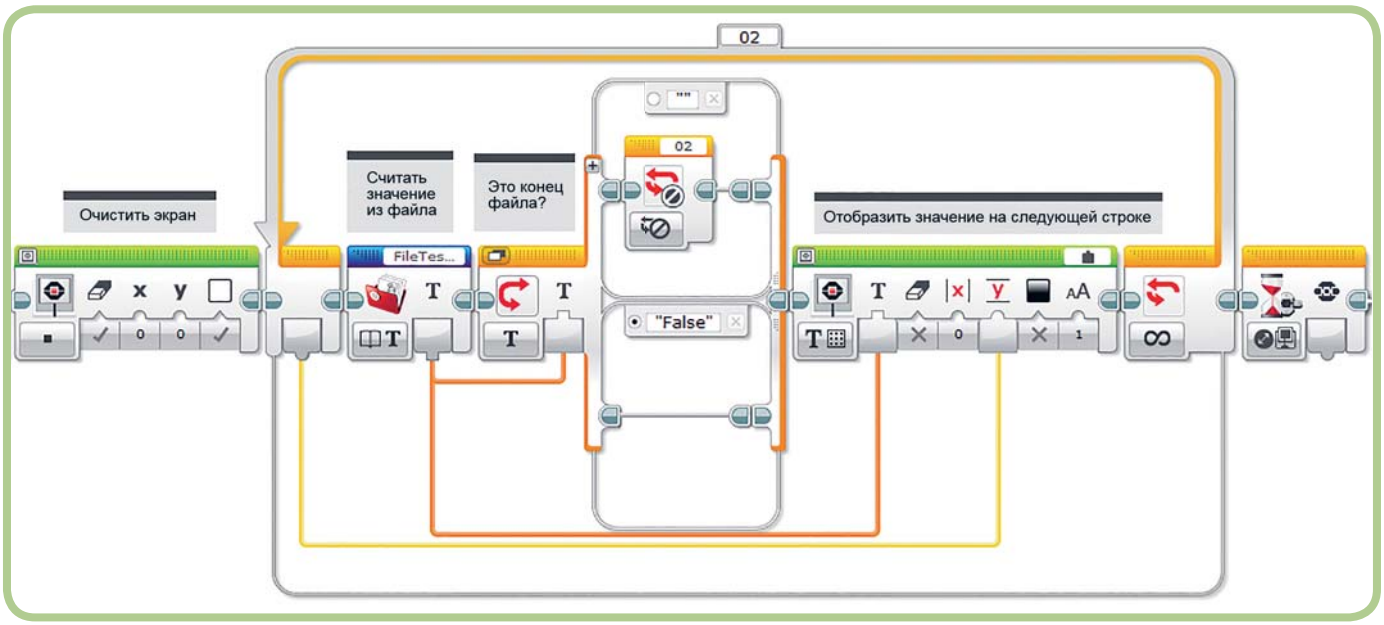


Рис. 16.10. Программа FileReader

блоку **Прерывание цикла** (Loop Interrupt) осуществляется выход из цикла. Другой случай задан по умолчанию, поэтому он будет использован, если в блоке **Доступ к файлу** (File Access) считается что-либо, кроме пустой строки. При выборе режима **Текст** (Text) для этого случая автоматически задается значение "False" («Ложь»).

Оставшиеся два блока должны показаться тебе знакомыми. Так, с помощью блока **Экран** (Display) отображается каждое считанное из файла значение, а в результате использования блока **Ожидание** (Wait) в режиме **Кнопки управления модулем** (Brick Buttons) ⇒ **Изменить** (Change) приостанавливается выполнение программы, что позволяет ознакомиться с этими значениями.

ПРАКТИКУМ 16.1

В настоящее время программа *FileReader* хорошо работает только с файлами, содержащими не более 12 значений, однако ты можешь приспособить ее для работы с более длинными файлами. Сделай так, чтобы после чтения и отображения 12 значений программа *FileReader* приостановилась и перешла в режим ожидания щелчка кнопкой. Затем запусти программу для очистки экрана и отображения следующих 12 значений. Этот процесс должен продолжаться до тех пор, пока не будут прочитаны все содержащиеся в файле значения.

Запусти эту программу после выполнения программы *FileTest*. На экране модуля EV3 должны отобразиться значения «0», «1» и «2». Измени имя файла на *MG_HighScore*, чтобы увидеть свой лучший результат в игре *MemoryGame*. При попытке задать имя несуществующего файла (например, *NotThere*) ты увидишь сообщение "File Read Error" («Ошибка чтения файла»).

Добавление меню в программу ColorCount

Использование файлов может сделать многие программы более функциональными! В этом разделе мы сохраним в файле данные, собранные программой *ColorCount* из гл. 15. Код для хранения значений будет похож на код в первой части программы *FileTest*, однако простое сохранение восьми значений в конце программы мало что дает. Нам также потребуется способ загрузки значений из файла. При этом необходимо обнулить эти значения.

ОПРЕДЕЛЕНИЕ КОНЦА ФАЙЛА

Программой *FileReader* считываются все данные из файла в виде текстовых значений, и используют пустую строку для определения конца файла. Однако этот метод не работает, если программа должна считывать данные в виде чисел. С помощью блока **Доступ к файлу** (File Access) в режиме **Считывание** (Read) ⇒ **Числовое значение** (Numeric) отображается значение 0 после завершения чтения данных из файла. В зависимости от содержимого файла 0 может являться допустимым значением, поэтому его часто нельзя использовать для определения конца файла.

При чтении чисел существуют три способа узнать о том, что из файла были прочитаны все данные:

1. Если количество значений известно заранее, можно явно указать, сколько значений требуется считать. Программой *FileTest* всегда считываются ровно три значения, поскольку известно, что файл содержит именно такое их количество.

2. При записи данных в файл ты можешь сначала записать количество значений. Затем код, отвечающий за чтение файла, должен прочитать это количество, а в дальнейшем оно будет использовано для определения количества значений для счета.

3. Если известно, что среди данных не встречается определенное значение, можно использовать его, чтобы отметить конец файла. Например, при работе с показаниями датчика цвета можно использовать значение –1, поскольку датчик цвета никогда не возвращает отрицательное число. По мере чтения файла программой может быть выполнена проверка на наличие этого значения подобно тому, как программой *FileReader* выполняется проверка на наличие пустой строки. Поскольку значением является число, ты можешь проверить его с помощью блока **Сравнение** (Compare).

Для выполнения этих функций добавим меню в программу *ColorCount*. После запуска отобразится список вариантов, затем программа перейдет в режим ожидания до тех пор, пока пользователь не выберет один из них с помощью кнопок модуля. После выполнения выбранной операции программа снова отобразит меню. Данное меню должно содержать четыре варианта:

Count (Подсчет) Начинается подсчет объектов. Происходит запуск кода из исходной программы *ColorCount* с несколькими изменениями.
Save (Сохранить) Сохраняются значения в массиве ColorCounts в файле *ColorCountData*.
Load (Загрузить) Загружаются значения из файла *ColorCountData* в массив ColorCounts.

Clear (Очистить) Задается 0 в качестве значения всех восьми элементов массива. Мы начнем с создания двух контейнеров. Один будет отображать четыре команды, а другой — управлять меню с учетом количества содержащихся в нем пунктов. Затем используем эти два контейнера и несколько других для создания улучшенной версии программы *ColorCount*. Описанные выше четыре пункта меню относятся к программе *ColorCount*, однако ты можешь использовать код, который обрабатывает нажатия кнопок, для выбора вариантов в любой другой программе.

Контейнер CreateMenu_CC

После запуска отображаются пункты меню, при этом выбранный пункт обозначен расположенным слева от него символом >. Содержимое экрана должно выглядеть следующим образом:

```
> Count
  Save
  Load
  Clear
```

Прежде чем писать код для управления меню, необходимо отобразить его пункты. Эту задачу решим, используя контейнер **CreateMenu_CC**. На рис. 16.11 показана программа *CreateMenu_CCBuilder*, на основе которой можно создать этот контейнер. Каждым блоком **Экран** (Display) отображается один из пунктов меню на отдельной строке, начиная с нулевой (самой верхней строки экрана). Для параметра **Столбец** (Column) в каждом блоке задано значение 2 для того, чтобы осталось место для маркера выделения (>) и пробела. Для параметра **Очистить экран** (Clear Screen) первого блока задано значение **Истина** (True), чтобы очистить экран модуля перед отображением меню. В остальных блоках для данного параметра задано значение **Ложь** (False).

Далее описана рекомендуемая мной последовательность действий для создания этого контейнера:

1. Создай новую программу под названием *CreateMenu_CCBuilder* и добавь в нее блоки, в соответствии с рис. 16.11.

2. Запусти программу *CreateMenu_CCBuilder*. На экране модуля EV3 должно отобразиться следующее:

```
Count
Save
Load
Clear
```

Убедившись в том, что меню отображается правильно, выполни следующие действия:

3. Выдели все четыре блока **Экран** (Display). Выбери команду меню **Инструменты** (Tools) ⇒ **Конструктор Мой блок** (My Block Builder), чтобы приступить к созданию контейнера.

4. В поле для имени введи **CreateMenu_CC**, а в поле для описания — **Создание меню для программы ColorCount**.

5. Выбери значок, который используется блоком **Экран** (Display) ( ).

6. Щелкни по кнопке **Завершить** (Finish), чтобы создать контейнер «Мой блок».

7. Нажми комбинацию клавиш **Ctrl+Z**, чтобы вернуть программу *CreateMenu_CCBuilder* в исходное состояние (это упростит процесс повторного создания контейнера, если в дальнейшем тебе потребуется внести изменения).

На данном этапе у тебя должен быть контейнер «Мой блок», отображающий пункты меню, которое мы будем использовать в начале программы *ColorCount*. Кроме того, этот контейнер потребуется для создания другого контейнера **SelectOption** для управления этим меню.

Контейнер SelectOption

Контейнер **SelectOption** позволяет пользователю выбрать один из пунктов меню. Кнопки «Вверх» и «Вниз» используются для выделения пункта, а кнопка «Центр» — для принятия этого выбора. Вот как это работает:

- После выполнения блока **CreateMenu_CC** содержимое экрана должно выглядеть следующим образом:

```
> Count
  Save
  Load
  Clear
```

- В переменной SO_Selection сохраняется значение, соответствующее выделенному в данный момент пункту меню. Оно начинается с 0 и изменяется при каждом щелчке кнопкой «Вверх» и «Вниз».
- При щелчке кнопкой «Вниз» маркер выделения должен переместиться вниз. Если перед щелчком кнопкой «Вниз» был выбран последний пункт (Clear), маркер должен вернуться к первому пункту (Count).

Рис. 16.11. Программа CreateMenu_CCBuilder

— При щелчке кнопкой «Вверх» маркер выделения должен переместить вверх. Если перед щелчком кнопкой «Вверх» был выбран первый пункт (Count), маркер должен перейти к последнему пункту (Clear).
— При каждом изменении выбранного пункта символ > удаляется и отображается слева от вновь выбранного пункта.
— При щелчке кнопкой «Центр» экран модуля очищается, и контейнером «Мой блок» выдается числовое значение, соответствующее выбранному пункту меню. Нумерация пунктов меню начинается с 0.

Контейнер **SelectOption** предусматривает два параметра: входной, предназначенный для задания количества пунктов меню и выходной — для возврата выбора пользователя. Этому блоку необязательно знать, какие именно пункты содержит меню (поскольку за их отображение отвечает контейнер **CreateMenu_CC**), однако ему нужно знать количество этих пунктов для правильного выполнения циклического возврата маркера выделения.

Для создания контейнера **SelectOption** напишем и тщательно протестируем программу *SelectOptionBuilder*. В этой программе используется блок **CreateMenu_CC** для отображения меню, а также блок **Константа** (Constant) для задания количества пунктов. В конце программы с помощью блока **Экран** (Display) возвращается номер выбранного пункта. Между ними создадим «сердце программы», которое превратим в контейнер. Оно будет содержать блоки, позволяющие пользователю выбрать пункт меню с помощью кнопок модуля «Вверх», «Вниз» и «Центр».

Выбор пункта меню

В программе используется одна переменная с именем SO_Selection, в которой хранится номер выбранного в данный момент пункта меню. В начале программы необходимо инициализировать переменную SO_Selection нулем, который соответствует первому пункту меню, а также отобразить символ > в строке 0, чтобы указать на выбранный пункт.

На рис. 16.12 показан раздел программы, отвечающий за инициализацию, а также код для обработки щелчка кнопкой «Центр». С помощью блоков в верхней части изображения на экран выводятся пункты меню, количество пунктов помещаются в шину данных, инициализируется переменная SO_Selection и отображается первый маркер выделения. За ними следуют блоки, представленные в нижней части изображения, благодаря которым программа включается в цикл и находится в режиме ожидания до тех пор, пока пользователь не щелкнет кнопкой «Вверх», «Вниз» или «Центр».

После того как пользователь нажал кнопку, в блоке **Переменная** (Variable) высчитывается номер ранее выбранного пункта, а в блоке **Экран** (Display) удалится отображенный ранее символ >, вместо него появится пробел. Номер кнопки, на которую нажал пользователь, передается в блок **Переключатель** (Switch), в котором предусмотрен случай для каждого идентификатора кнопки (4 для кнопки «Вверх», 5 для кнопки «Вниз» и 2 для кнопки «Центр»). На рис. 16.12 показан случай для кнопки «Центр», в котором для выхода из цикла применяется блок **Прерывание цикла** (Loop Interrupt) с именем *SO_02*. В этом случае не используются две шины данных, входящие в блок **Переключатель** (Switch). Они пригодятся для двух других случаев.

ПРИМЕЧАНИЕ **При создании контейнера «Мой блок», завершающего цикл с помощью блока Прерывание цикла (Loop Interrupt), присвой ему имя, которое ты никогда не станешь использовать в основном цикле программы, чтобы избежать ошибок. Например, если поместить контейнер с циклом 02 в основной цикл программы с именем 02, то блок Прерывание цикла (Loop Interrupt) прервет и основной цикл программы!**

При нажатии кнопки «Вниз» выполняется код, относящийся к кнопке «Вниз» (рис. 16.13), с помощью которого к текущему значению переменной SO_Selection прибавляется 1, для перехода к следующему варианту. В блоке **Математика** (Math) используется режим **Дополнения** (Advanced) для вычисления нового значения SO_Selection, (*a* + 1)% *b*, где *a* — номер выбранного в данный момент пункта; *b* — общее количество пунктов меню. Оператор деления по модулю гарантирует возврат к пункту 0 при достижении конца списка. После того как блоком **Математика** (Math) будет вычислен номер вновь выбранного пункта, с помощью блока **Экран** (Display) в соответствующей строке отобразится символ >, а вычисленное значение сохранится в переменной SO_Selection.

Снова применяем оператор деления по модулю в коде для кнопки «Вверх», хотя математические вычисления в нем будут отличаться, поскольку придется работать с отрицательными числами. При нажатии кнопки «Вверх» мы должны перемещаться по списку в направлении, противоположном тому, в котором мы перемещались при нажатии кнопки «Вниз». При этом важно уменьшать значение переменной и переходить к последнему пункту списка при достижении его начала. В данном примере, если до этого был выбран вариант 0, то теперь необходимо выбрать вариант 3.

Однако мы не можем просто переключиться на вычитание, поскольку результат выражения (*a* – 1)% *b* не позволит нам перейти к варианту 3 после варианта 0. Когда *a* равно 0, (*a* – 1) = –1, а остаток от деления –1 на 4 равен –1, а не 3.

Решение состоит в том, чтобы идти вперед, а не назад. Перемещение вперед на общее количество позиций –1 равнозначно перемещению на 1 позицию назад. Таким образом, в примере с четырьмя вариантами вместо вычитания 1 для перемещения на шаг назад, мы прибавляем 3, а затем применяем оператор деления по модулю, используя выражение (*a* + *b* – 1)% *b*. Это должно обеспечить корректное выполнение циклического возврата.

Определившись с выражением для блока **Математика** (Math), с помощью которого можно переместиться на 1 позицию списка вверх, напишем код (рис. 16.14). Единственное его отличие от кода для кнопки «Вниз» заключается в выражении, используемом в блоке **Математика** (Math).

Возвращение выбранного варианта

После перемещения маркера выделения (>) до нужного пункта пользователь щелкает кнопкой «Центр», что приводит

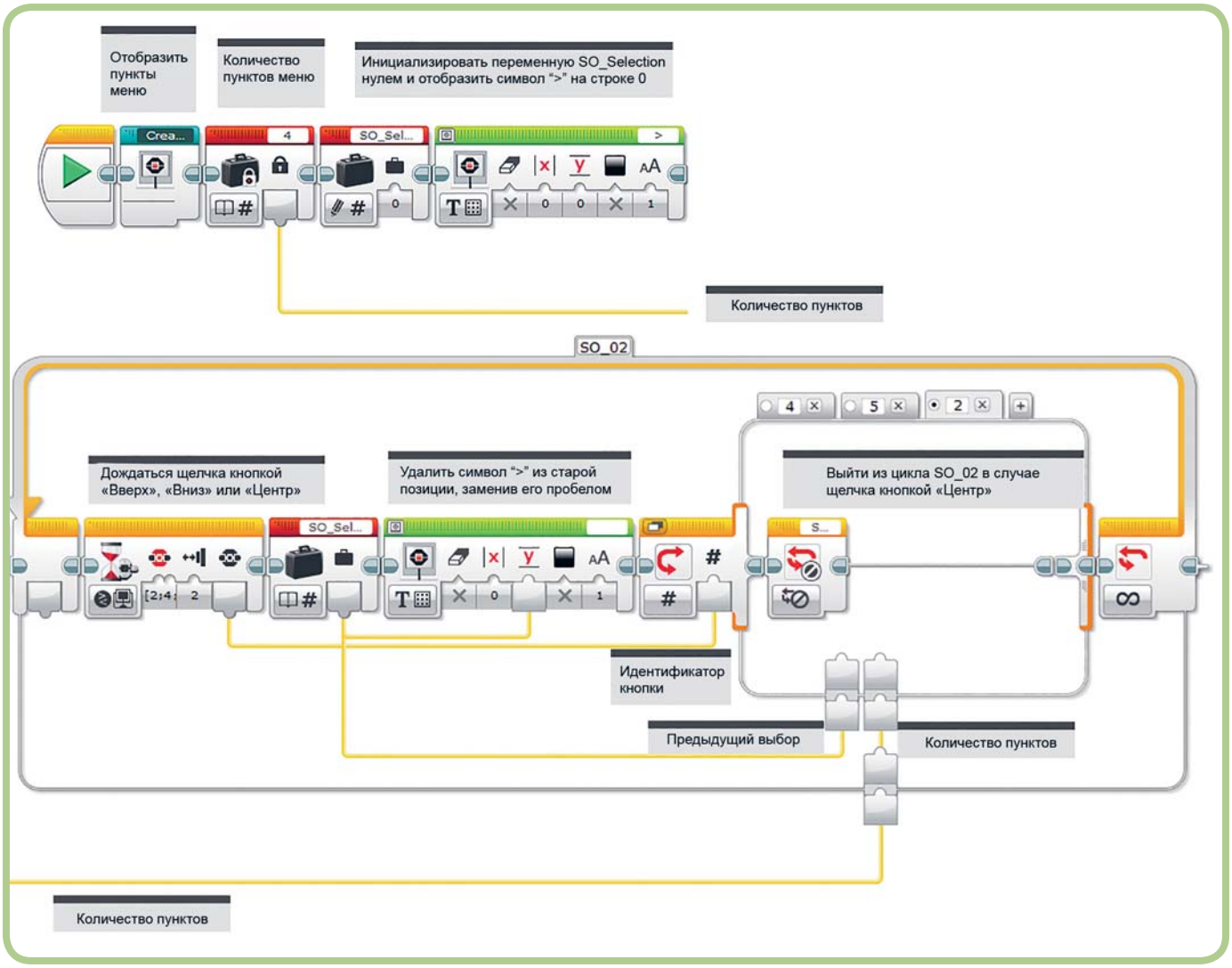


Рис. 16.12. Ожидание щелчка кнопкой и выход из цикла при щелчке кнопкой «Центр»

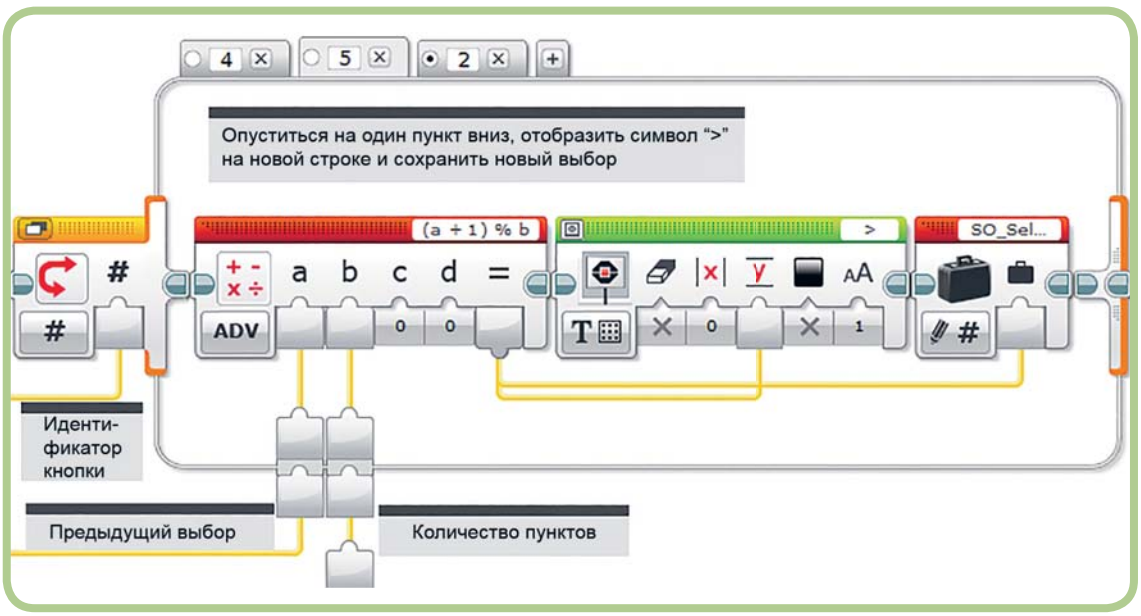


Рис. 16.13. Обработка щелчка кнопкой «Вниз»

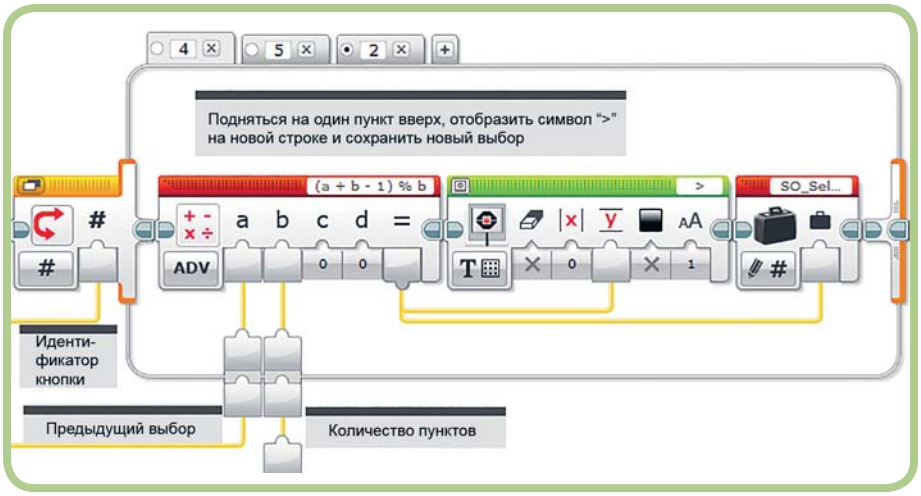


Рис. 16.14. Обработка щелчка кнопкой «Вверх»

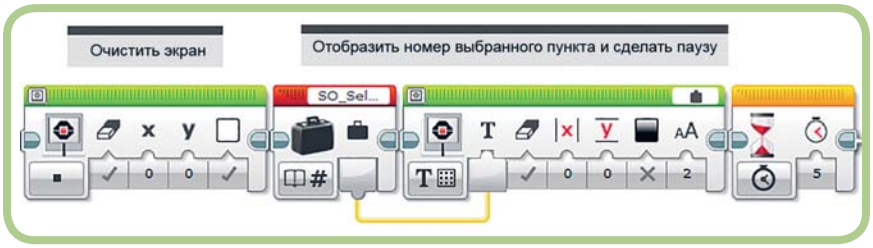


Рис. 16.15. Чтение и отображение выбранного пункта меню

к выходу из цикла. Затем с помощью контейнера «Мой блок» экран должен очиститься (это хорошая практика, когда блок убирает за собой), а в качестве выходного параметра снова отобразится выбранный вариант. Этот вариант сохраняется в переменной SO_Selection, поэтому для считывания значения потребуется блок **Переменная** (Variable). В программе-конструкторе мы выводим значение SO_Selection на экран модуля EV3, что позволяет нам легко протестировать программу перед использованием инструмента **Конструктор Мой блок** (My Block Builder). На рис. 16.15 показаны последние четыре блока программы.

После запуска программа должна отобразить пункты меню и правильно переместить маркер выделения в ответ на щелчки кнопками «Вверх» и «Вниз». При щелчке кнопкой «Центр» программ должна отобразить номер выбранного пункта.

Создание контейнера «Мой блок»

Убедившись в том, что программа *SelectOptionBuilder* работает правильно, ты можешь создать на ее основе контейнер **SelectOption**. Вот что нужно для этого сделать:

1. Щелкай по кнопке масштабирования ( ) в правой части панели инструментов области программирования, пока не отобразится вся программа.

2. Нарисуй рамку выделения вокруг всех блоков за исключением блоков **CreateMenu_CC** и **Константа** (Constant) в начале, а также блоков **Экран** (Display) и **Ожидание** (Wait) в конце.

3. Выбери команду меню **Инструменты** (Tools) ⇒ **Конструктор Мой блок** (My Block Builder). Если блоки были выбраны правильно, ты увидишь один текстовый ввод и один числовой вывод.

4. В качестве имени контейнера задай **SelectOption** и выбери значок для кнопок модуля ( ).

5. В качестве имени первого параметра задай **Number of Options** (Количество пунктов) и выбери значок с изображением решетки ( ).

6. Для второго параметра задай имя **Selection** (Выбор) и выбери значок, используемый для обозначения номера ( ).

7. Щелкни по кнопке **Завершить** (Finish).

8. Используй комбинацию клавиш **Ctrl+Z** для восстановления программы *SelectOptionBuilder*.

Теперь у нас есть один контейнер для создания пунктов меню в программе (*CreateMenu_CC*) и еще один для отображения меню и чтобы пользователь мог выбрать один из его пунктов. Далее мы создадим базовую структуру программы, а затем подробно разберем процесс выполнения каждой из перечисленных в меню команд.

Новая структура программы *ColorCount*

Программы, содержащие меню, обычно имеют следующую структуру, и новая программа *ColorCount* (рис. 16.16) не является исключением:

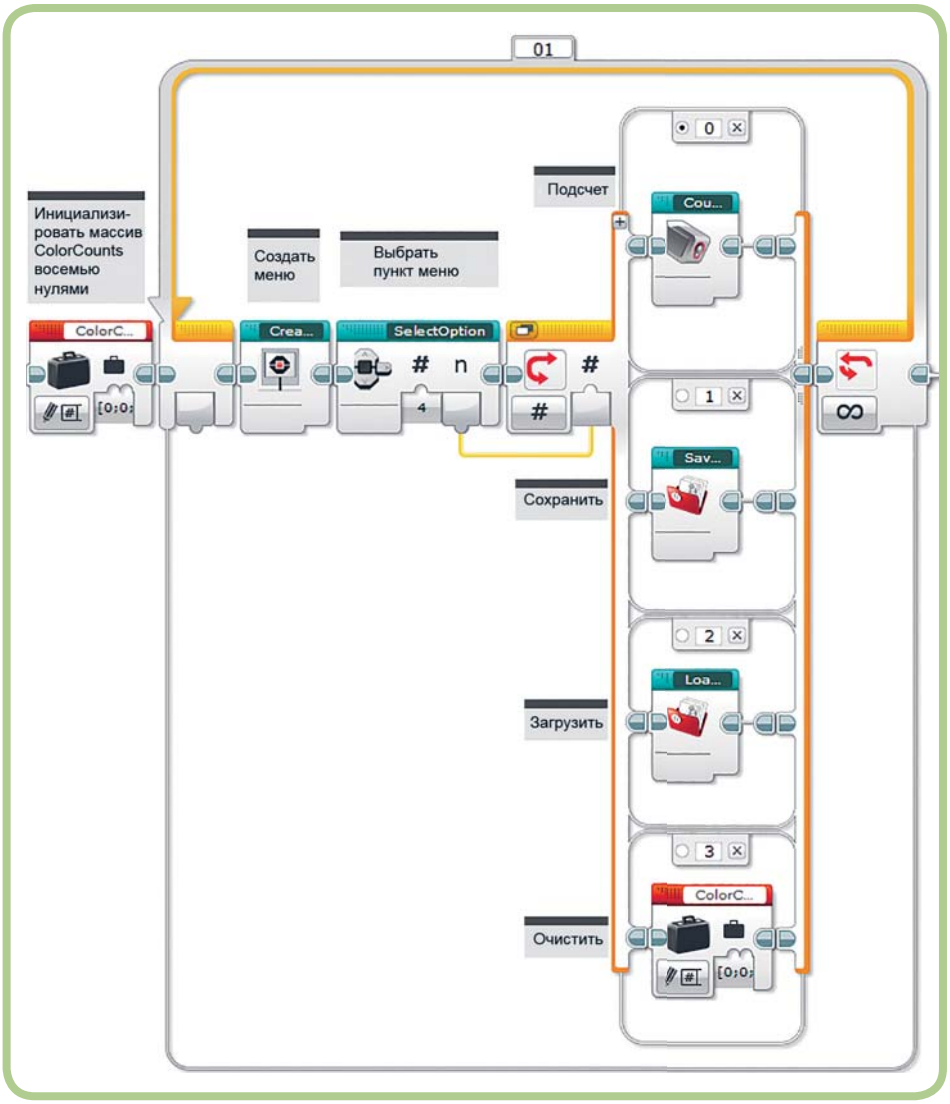


Рис. 16.16. Новая программа ColorCount

1. Инициализируй данные программы. В случае *ColorCount* это означает инициализацию переменной ColorCounts массивом с восемью нулями.

2. Зайди в цикл, который отображает меню, а затем используй контейнер **SelectOption**, чтобы пользователь мог выбрать необходимый вариант.

3. Выполни выбранную команду меню с помощью блока **Переключатель** (Switch). Удобнее реализовывать каждый пункт меню в контейнере, чтобы код не становился слишком громоздким.

В итоговой программе *ColorCount* используются три контейнера «Мой блок», которые мы создадим в следующих разделах, для подсчета объектов, а также для сохранения итоговых значений в файл и их загрузки из файла. Для обнуления итоговых значений требуется только один блок **Переменная** (Variable), поэтому данная функция реализуется напрямую.

Создай новую программу *ColorCount* пока без контейнеров для реализации функций и убедись в том, что пункты меню отображаются на экране модуля, а код для их выбора работает должным образом. Ты сможешь протестировать каждую функцию после создания соответствующего контейнера.

Подсчет объектов

Программа *ColorCount* из гл. 15 содержит нужную нам логику для подсчета объектов, поэтому мы используем ее в качестве основы для контейнера **Count_CC**. Даже если тебе пришлось создать новую программу *ColorCount* из этой главы, скопируй исходную программу *ColorCount* из проекта *Chapter15* в проект *Chapter16* и переименуй ее в *Count_CCBuilder*.

Программа *Count_CCBuilder* состоит из двух разделов: первый отображает название каждого цвета, за которым следует значение 0, а второй осуществляет подсчет. Однако в новой программе подсчет не обязательно будет начинаться с 0. Поэтому в первом разделе вместо отображения нулей нам нужно вывести на экран значения элементов массива ColorCounts, используя блок **Переменная** (Variable), за которым следует блок **Операции над массивом** (Array Operations), считывающий количество объектов каждого цвета.
На рис. 16.17 показана полная версия программы *Count_CCBuilder* с изменениями, которые необходимо внести.

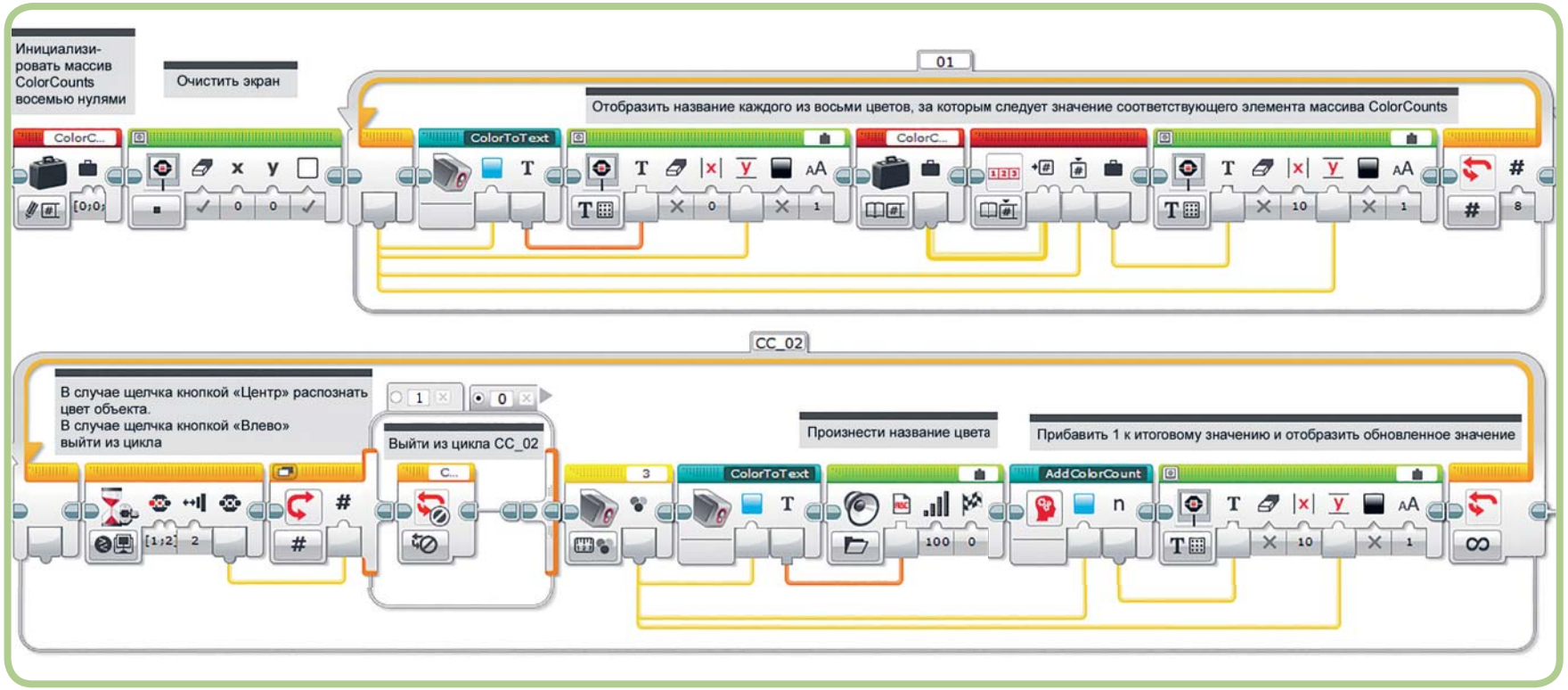


Рис. 16.17. Программа Count_CCBuilder

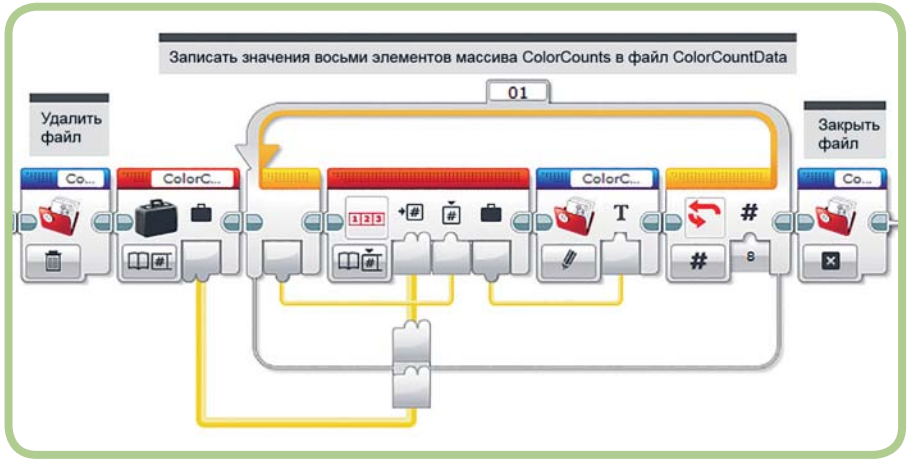


*Рис. 16.18. Контейнер **Save_CC***

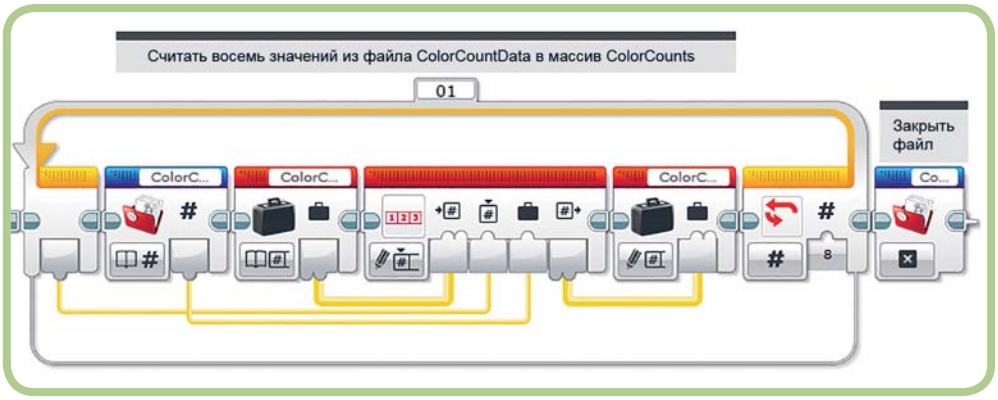


Рис. 16.19. Контейнер ***Load_CC***

Второй раздел программы уже правильно идентифицирует каждый цвет и сохраняет итоговые значения в массиве ColorCounts, однако исходный цикл повторяется до тех пор, пока ты его не остановишь. Следует предусмотреть способ завершения цикла и возврата к меню. Теперь блок **Ожидание** (Wait) находится в ожидании щелчка кнопкой «Центр» или «Влево». При щелчке кнопкой «Центр» программа распознает цвет объекта и обновляет соответствующее итоговое значение. При щелчке кнопкой «Влево» блок **Прерывание цикла** (Loop Interrupt) внутри блока **Переключатель** (Switch) осуществляется выход из цикла, и основная программа возвращается к меню.

Изменим также имя цикла с *02* на *CC_02*, чтобы блок **Прерывание цикла** (Loop Interrupt) случайно не завершил выполнение неправильного цикла. В созданной программе *ColorCount* нет цикла с именем *02*, однако в будущем, возможно, возникнет необходимость в повторном использовании этого контейнера, поэтому лучше изменить имя сейчас, чтобы избежать потенциального конфликта имен в дальнейшем.

Протестируй программу, чтобы убедиться в том, что она правильно подсчитывает количество цветных объектов и отображает итоговые значения, а затем создай контейнер **Count_CC**, используя блоки, показанные на рис. 16.17, за исключением блока **Переменная** (Variable) в самом начале. Ведь мы не хотим, чтобы значения элементов массива обнулялись при каждом выполнении этого контейнера! Затем добавь новый блок в программу *ColorCount*. Теперь после запуска программы и выбора пункта меню **Count** программой должна отобразиться названия цветов и итоговые значения, а затем начинаться подсчет.

ПРАКТИКУМ 16.2

Для завершения выполнения программы *ColorCount* придется использовать кнопку модуля «Назад» (или остановить программу из среды EV3), при этом ты рискуешь забыть о необходимости сохранить данные перед завершением выполнения программы. Добавь в меню пункт **Save & Exit**, позволяющий сохранить данные, а затем выйти из основного цикла программы.

Проверь работу программы на нескольких объектах, понаблюдай за изменением итоговых значений, а затем щелкни кнопкой «Влево». Программа должна вернуться в меню. После очередного выбора пункта **Count** на экране должны отобразиться итоговые значения, полученные в результате предыдущего теста.

На данном этапе ты также можешь протестировать работу функции **Clear**. Сначала выбери пункт **Count** и проверь работу этой функции на нескольких объектах. Выбери пункт **Clear**, а затем снова пункт **Count**, и ты увидишь, что итоговые значения обнулились.

Сохранение и загрузка итоговых значений

Следующим шагом является добавление двух контейнеров для сохранения и восстановления данных, которые используют файл *ColorCountData*. С помощью контейнера **Save_CC** (рис. 16.18) файл удалится, запишутся восемь значений из массива ColorCounts, а затем файл закроется.

В контейнере **Load_CC** (рис. 16.19) используется аналогичная структура для считывания восьми значений из файла *ColorCountData* и их помещения в массив ColorCounts. Обязательно используй команду **Save** хотя бы один раз перед тестированием команды **Load**! Если при запуске этого контейнера файл *ColorCountData* еще не существует, при попытке чтения первого значения блоком **Доступ к файлу** (File Access) возникнет сбой программы.

Тестирование

Добавь в свою программу контейнеры **Save_CC** и **Load_CC** (см. рис. 16.16) и пункт **Clear** в меню, если в нем его еще нет. Протестируй все четыре команды, чтобы убедиться, что они функционируют ожидаемым образом. После завершения программы запусти ее снова и используй пункт меню **Load**, чтобы гарантированно восстановить результаты предыдущего выполнения программы.

Управление памятью

Все файлы, хранящиеся в модуле EV3 (программы, звуки, изображения и файлы данных), занимают часть его памяти.

Из этого раздела ты узнаешь, как применять инструмент **Обозреватель памяти** (Memory Browser) для определения объема занятой памяти, удаления файлов с целью освобождения места, а также для перемещения файлов между проектами или между модулем и компьютером.

Для того чтобы открыть окно **Обозреватель памяти** (Memory Browser), выбери команду меню **Инструменты** (Tools) ⇒ **Обозреватель памяти** (Memory Browser) или щелкни по кнопке в правом нижнем углу вкладки **Информация о модуле** (Brick Information) (рис. 16.20).

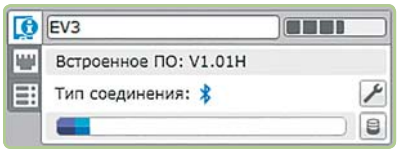


*Рис. 16.20. Кнопка **Обозреватель памяти** (Memory Browser) на вкладке **Информация о модуле** (Brick Information)*

В левой части окна **Обозреватель памяти** (Memory Browser) (рис. 16.21) показано, сколько осталось свободного места. На изображении видно, что большая часть памяти моего модуля EV3 по-прежнему пуста и доступна для использования.

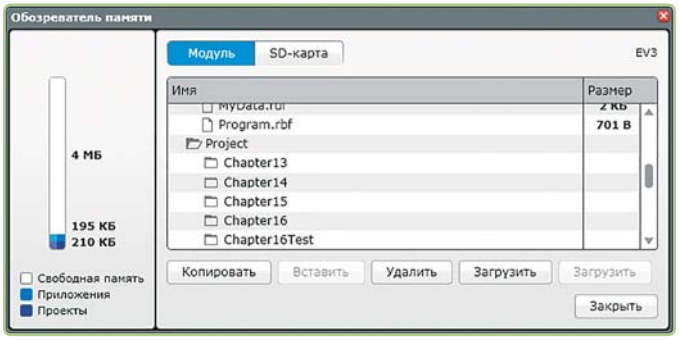


*Рис. 16.21. Инструмент **Обозреватель памяти** (Memory Browser)*

Правая часть окна **Обозреватель памяти** (Memory Browser) содержит список папок и файлов, хранящихся в модуле EV3. Работа с файлами, содержащимися в модуле EV3, аналогична работе с файлами в компьютере. Двойной щелчок по папке открывает ее, что позволяет увидеть ее содержимое. Например, на рис. 16.22 показаны некоторые файлы в папке *Chapter16*. Эта папка содержит все файлы, которые применяются в программах проекта *Chapter16*: файлы для каждой программы и контейнера «Мой блок»; звуковые файлы блока **Звук** (Sound); файлы изображений блока **Экран** (Display); файлы данных, созданные блоком **Доступ к файлу** (File Access). *Расширение* каждого файла (последние три буквы его имени после точки) определяет тип файла. Например, в программах и контейнерах «Мой блок» используется расширение *.rbf*, а при записи звуковых файлов — *.rsf*. Файлы данных, созданные блоком **Доступ к файлу** (File Access), имеют расширение *.rtf*.

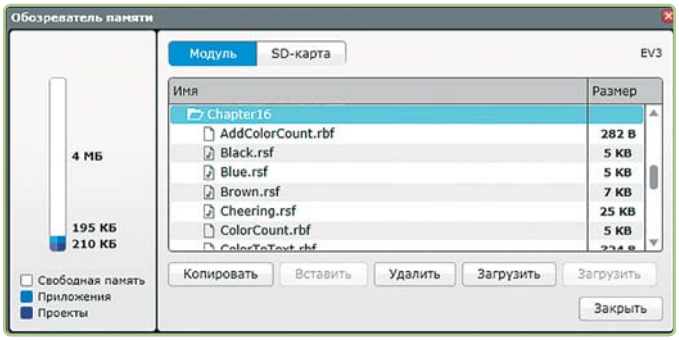


Рис. 16.22. Некоторые файлы из проекта Chapter16

Под списком папок и файлов находятся пять кнопок:

Кнопка Удалить (Delete) Удаляет выбранный файл или папку.

Кнопки Копировать (Copy) и Вставить (Paste) Копируют файлы из одного проекта в другой. Выдели файл для копирования и щелкни по кнопке **Копировать** (Copy). Затем выбери нужный проект и щелкни по кнопке **Вставить** (Paste).

Кнопка Загрузить (Upload) Используется для копирования выбранного элемента из модуля EV3 на компьютер. При щелчке по кнопке **Загрузить** (Upload) открывается диалоговое окно, позволяющее выбрать место для сохранения файла, а также изменить его имя. При копировании файла данных, созданного блоком **Доступ к файлу** (File Access), измени его расширение с *.rtf* на *.txt*, прежде чем пытаться открыть его в своем любимом текстовом редакторе.

Кнопка Загрузить (Download) Копирует файл с твоего компьютера на модуль EV3. При щелчке по этой кнопке открывается диалоговое окно, в котором можно выбрать файл для загрузки. Файл помещается в проект, выбранный в настоящий момент в окне **Обозреватель памяти** (Memory Browser).

Каждый проект имеет отдельный список файлов, ты не можешь записать файл из программы в одном проекте и прочитать его из программы в другом проекте. В этом можно убедиться с помощью программы *FileReader*:

1. Задай **FileTestData** в качестве имени файла и запусти программу *FileReader*, чтобы убедиться в ее работоспособности.

2. Создай новый проект под названием *Chapter16Test* и скопируй программу *FileReader* из проекта *Chapter16* в новый проект.

3. Запусти программу *FileReader* из нового проекта.

В этом случае возникнет сбой программы и на экране отобразится сообщение “File Read Error” («Ошибка чтения файла»), так как в проекте *Chapter16Test* нет файла с именем *FileTestData*. На самом деле это полезная функция, поскольку она гарантирует, что программы, содержащиеся в разных

ТЕКСТОВЫЕ ФАЙЛЫ EV3 И ОПЕРАЦИОННАЯ СИСТЕМА WINDOWS

В модуле EV3 используется операционная система Linux, и текстовые файлы, которые используются в этом модуле, включая те, что создаются твоими программами, записываются в текстовом формате Linux. К сожалению, текстовые файлы в операционной системе Windows имеют другой формат. В текстовых файлах Linux для обозначения конца строки есть один специальный символ под названием «перевод строки». В операционной системе Windows для обозначения конца строки используется пара специальных символов — перевод строки и возврат каретки. Отметим, что в операционной системе macOS применяется тот же текстовый формат, что и в операционной системе Linux, поэтому с ней не возникает проблем.

Это различие вызывает проблемы при просмотре или редактировании файлов EV3 в операционной системе Windows. В некоторых программах Windows, например WordPad, учтено это различие, поэтому текстовые файлы EV3 воспроизводятся корректно. При использовании других программ, например Блокнота (Notepad), отображается все содержимое файла на одной строке. Если тебе необходимо просто *просмотреть* файл, то это не проблема. Просто используй программу WordPad вместо программы Блокнот (Notepad).

С редактированием файла, который ты хочешь загрузить в модуль EV3 и использовать в своей программе, дела обстоят сложнее. Никакие стандартные инструменты Windows не позволяют записать файл в формате Linux. Если ты сохранишь файл в программе WordPad, каждая его строка будет оканчиваться двумя специальными символами, в то время как для модуля EV3 необходим только один. Если файл содержит только цифры, модуль EV3 просто «не заметит» дополнительный символ и считает числа правильно. Однако если файл содержит текстовые значения, то каждое значение, кроме первого, будет начинаться с дополнительного пробела, поскольку с помощью модуля EV3 символ возврата каретки преобразуется в пробел.

Создавая на своем компьютере файл, предназначенный для загрузки в модуль EV3 и дальнейшего использования его содержимого в программе EV3, перед его загрузкой необходимо удалить символы возврата каретки, чтобы обеспечить корректное чтение его содержимого модулем EV3. Для решения этой задачи существуют специальные программы; например, приложение *Tofrodos* (**www.thefreecountry.com/tofrodos/**), преобразующее файлы Windows в формат Linux.

Кроме того, до окончания процесса редактирования следует сохранять файл в текстовом формате, а не в формате Rich Text Format. Во многих программах выбор формата файла осуществляется в зависимости от расширения файла, и если ты сохранишь файл с расширением *.rtf*, эти программы могут обработать твой файл *.txt* как файл *.rtf*. Для того чтобы создать файл с командами, ты можешь сначала сохранить его как *commands.txt*, а затем перед загрузкой в модуль EV3 переименовать его в файл с расширением *.rtf*, с которым работает модуль EV3. Иными словами, преобразуй файл *commands.txt* в *commands.rtf*.

проектах, не смогут случайно перезаписать один и тот же файл. Если ты захочешь использовать файл из одного проекта в другом проекте, просто скопируй его с помощью кнопок **Копировать** (Copy) и **Вставить** (Paste).

В основном ты будешь использовать инструмент **Обозреватель памяти** (Memory Browser) для удаления старых проектов при заполнении памяти модуля EV3. Однако его также можно применять для копирования файлов данных с модуля EV3 на компьютер при использовании модуля EV3 для ведения журнала данных, о котором пойдет речь в гл. 17.

ВНИМАНИЕ! **Загрузка новой прошивки в модуль EV3 приведет к удалению всех созданных программ и файлов. Перед обновлением прошивки EV3 используй инструмент Обозреватель памяти (Memory Browser) для загрузки на свой компьютер всех файлов данных, которые ты хочешь сохранить.**

Дальнейшее исследование

Для того чтобы попрактиковаться в применении файлов, попробуй выполнить следующие упражнения:

1. Контейнеры **Save_CC** и **Load_CC** были созданы специально для программы *ColorCount*. Создай контейнеры более общего назначения для сохранения числового массива в файл и для его загрузки из файла. Так, контейнер, отвечающий за сохранение, должен принимать в качестве входных данных имя файла и массив, а контейнер, отвечающий за загрузку, в качестве входных данных принимать имя файла, а выдавать массив. При сохранении значений в этот файл сначала запиши в него количество элементов, чтобы при чтении файла точно знать, сколько значений требуется считать.

2. Если исходный массив пустой, то в контейнере для сохранения должно записаться только значение его длины (0). После того как контейнер для загрузки считает значение длины, он должен с помощью блока **Переключатель** (Switch) проверить длину и войти в цикл для чтения значений только в том случае, если значение длины превышает 0. Установка счетчика цикла на 0 не сработает, поскольку этот счетчик проверяется только после одного выполнения тела цикла.

3. Добавь в программу *ButtonCommand* меню, включающее команды для сохранения, загрузки, создания, отображения или запуска программы. Функция отображения полезна, поскольку после запуска программы с ее помощью можно увидеть предусмотренные в ней команды, как в случае создания программы с помощью кнопок.

Заключение

С помощью файлов можно сохранять данные из программы в память модуля EV3. Ты можешь использовать эти данные позже в этой же программе, при следующем запуске этой программы или даже в другой программе. Блоком **Доступ к файлам** (File Access) предусмотрены все функции, необходимые для создания, записи, чтения и удаления файла.

На примерах тестовых программ, описанных в начале главы, был продемонстрирован базовый принцип работы блока **Доступ к файлам** (File Access), после чего можно было применить полученные знания на практике для добавления в программу *MemoryGame* функции сохранения лучшего результата. Изменение программы *ColorCount* оказалось чуть более сложным, пришлось использовать контейнеры для реализации меню и хранения данных программы. Ты можешь использовать созданный для этой программы контейнер **SelectOption** в других программах, в которых требуется добавить меню.

С помощью инструмента **Обозреватель памяти** (Memory Browser) можно управлять хранящимися в модуле EV3 файлами (программами и файлами с данными). В этом окне ты можешь удалять файлы, чтобы освободить место для других программ, а также перемещать файлы между проектами или между модулем EV3 и компьютером.

17

Ведение журнала данных

Из этой главы ты узнаешь, как использовать уже изученные функции EV3 для записи данных о моторах и датчиках в файлы, т. е. как превратить модуль EV3 в *регистратор данных*. *Ведение журнала данных* — это процесс их сбора и записи.

Сначала мы поэкспериментируем, чтобы разобраться с параметром **Текущая мощность** (Current Power) блока **Вращение мотора** (Motor Rotation). Затем посмотрим, как функционирует параметр **Рулевое управление** (Steering) блока **Рулевое управление** (Move Steering). В конце главы мы проведем эксперимент, чтобы проверить надежность работы программы *LightPointer* (см. гл. 11), в которой используется датчик цвета для поворота робота TriBot в направлении источника света.

Мы будем применять только те функции, которые являются общими для обеих версий конструктора. Однако в образовательной версии предусмотрено больше способов сбора и представления данных, полученных в ходе экспериментов, что делает ее отличным инструментом для использования в классе. Если ты работаешь с образовательной версией конструктора, тебе стоит потратить время на изучение этих функций.

Сбор данных и модуль EV3

Сбор данных имеет огромное значение для любого эксперимента, однако собирать данные вручную утомительно, и при этом возможны ошибки. Большинство людей не в состоянии быстро фиксировать показания через точные временные интервалы или в течение длительного периода. К счастью, компьютеры отлично справляются с этой задачей. Комбинация компьютера и датчиков EV3 делает этот модуль идеальным средством для сбора данных.

При разработке программы бывает полезно протестировать поведение датчика или мотора в условиях, с которыми может столкнуться робот во время выполнения программы. Чем больше тебе известно о моторах, датчиках и блоках программирования, тем легче создавать работающие программы, поэтому давай приступим к экспериментам!

Исследование параметра «Текущая мощность»

В блоке **Вращение мотора** (Motor Rotation) предусмотрен режим **Измерение** (Measure) ⇒ **Текущая мощность** (Current Power), содержащий значение *мощности* мотора, как общего выражения силы в определенный момент времени. С этим режимом связан параметр **Мощность** (Power) блока **Рулевое управление** (Move Steering), и в следующих разделах мы напишем несколько простых программ для регистрации данных с целью изучения этой взаимосвязи.

Программа CurrentPowerTest

В ходе первого эксперимента мы будем записывать значения параметра **Текущая мощность** (Current Power) мотора B при корректировке параметра **Мощность** (Power) блока **Большой мотор** (Large Motor), используя программу *CurrentPowerTest* (рис. 17.1). При использовании этой программы мотор запускается на полную мощность, а затем входит в цикл, который постепенно уменьшает значение параметра **Мощность** (Power) блока **Большой мотор** (Large Motor) со 100 до 1. На каждом этапе в этой программе фиксируется значение текущей мощности, после чего значения параметров **Мощность** (Power) и **Текущая мощность** (Current Power) записываются в файл *CurrentPowerTestData*.

Вместо того чтобы записывать значения параметров **Мощность** (Power) и **Текущая мощность** (Current Power) по отдельности, используя блок **Доступ к файлу** (File Access), с помощью этой программы создается файл данных, в котором оба значения записываются в одной строке через запятую. Этот формат называется CSV (Comma-Separated Values — значения, разделенные запятыми). Файлы в таком формате обычно имеют расширение *.csv*. Не составит труда открыть подобные файлы с помощью электронных таблиц, поэтому такой способ упорядочения данных в дальнейшем облегчит нам процесс их анализа.

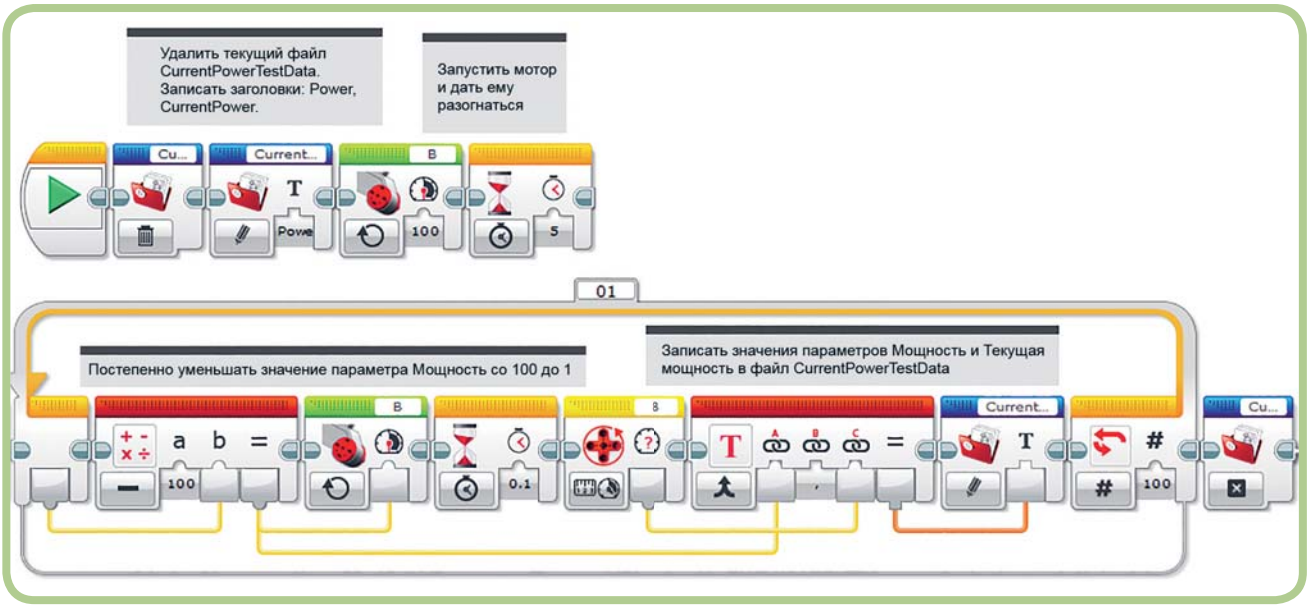


Рис. 17.1. Программа CurrentPowerTest

ПРИМЕЧАНИЕ **В США значения в файле CSV разделяются запятой; в других странах в качестве разделителя может использоваться точка с запятой. Измени программы из этой главы так, чтобы в них использовалась точка с запятой, если именно с таким разделителем работает твоя программа электронных таблиц.**

Рассмотрим назначение каждого блока этой программы для регистрации данных:

1. С помощью первого блока **Доступ к файлу** (File Access) удаляется файл. Ведь мы хотим создавать новый файл со свежими данными при каждом запуске программы.

2. Во втором блоке **Доступ к файлу** (File Access) заголовки "Power, CurrentPower"записываются в файл.

3. В блоке **Большой мотор** (Large Motor) используется режим **Включить** (On) для запуска мотора B на полную мощность.

4. Блоком **Ожидание** (Wait) приостанавливается выполнение программы на пять секунд для того, чтобы мотор развил полную мощность.

5. Блок **Цикл** (Loop) повторяется 100 раз, и при выполнении каждого цикла значение мощности уменьшается на 1.

6. С помощью блока **Математика** (Math) вычисляется значение параметра **Мощность** (Power), при этом вычитается **Параметр цикла** (Loop Index) из 100. При первом выполнении цикла результат этого вычитания равен 100; при следующем — 99; и т. д. При последнем выполнении цикла значение **Параметр цикла** (Loop Index) будет равно 99, а значение параметра **Мощность** (Power) — 1.

7. Результат выполнения блока **Математика** (Math) передается блоку **Большой мотор** (Large Motor) для корректировки параметра **Мощность** (Power).

8. Благодаря блоку **Ожидание** (Wait) обеспечивается небольшая пауза, чтобы мотор мог замедлить свое вращение в соответствии с новым значением параметра **Мощность** (Power).

9. В блоке **Вращение мотора** (Motor Rotation) используется режим **Измерение** (Measure) ⇒ **Текущая мощность** (Current Power), чтобы считать показание текущей мощности и поместить его в шину данных.

10. В блоке **Текст** (Text) объединяются значение параметра **Мощность** (Power), используемое блоком **Большой мотор** (Large Motor), и показание, посчитанное блоком **Вращение мотора** (Motor Rotation), и разделяются запятой (,).

11. Блоком **Доступ к файлу** (File Access) записывается значение из блока **Текст** (Text) в файл *CurrentPowerTestData*.

12. С помощью последнего блока закрывается файл *CurrentPowerTestData*.

При запуске программы мотор B работает на полную мощность. Через пять секунд он начинает замедляться и останавливается, когда программа завершается, — примерно через 10 секунд. В этот момент в памяти модуля EV3 должен находиться файл с именем *CurrentPowerTestData.rbt*, содержащий данные, полученные в ходе этого эксперимента.

Используй инструмент **Обозреватель памяти** (Memory Browser) (**Инструменты** (Tools) ⇒ **Обозреватель памяти** (Memory Browser)) для перемещения файла с модуля EV3 на компьютер. Перед сохранением файла измени его

расширение на.*csv*, чтобы твой текстовый редактор или программа электронных таблиц смогли распознать его формат. Также полезно добавить в конце имени файла число, позволяющее различить данные, полученные в результате разных запусков программы (рис. 17.2).

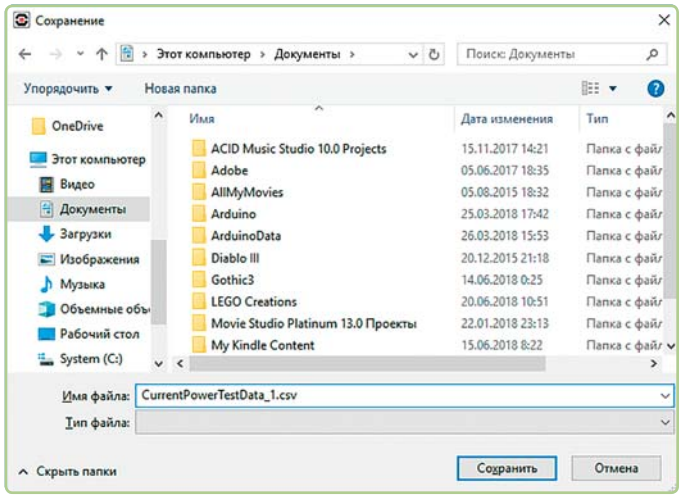


Рис. 17.2. Сохранение файла данных

Ты можешь открыть этот файл в текстовом редакторе или программе для работы с электронными таблицами. Удобно использовать для анализа данных электронную таблицу (например, OpenOffice.org Calc или Microsoft Excel), в которой можно просматривать необработанные данные и создавать графики. В табл. 17.1 показано, как должны выглядеть два заголовка и первые 10 значений в программе электронных таблиц (номер измерения соответствует номеру строки в электронной таблице). Значение параметра **Мощность** (Power) начинается со 100 и уменьшается на 1, как и ожидалось. Значение параметра **Текущая мощность** (Current Power) начинается с 76 и остается примерно на одном и том же уровне при проведении первых 10 измерений.

Табл. 17.1. Показания текущей мощности при проведении первых 10 измерений параметра Мощность

Измерение №	Мощность	Текущая мощность
1	100	76
2	99	77
3	98	76
4	97	74
5	96	75
6	95	76
7	94	77
8	93	75
9	92	75
10	91	76

Прокрутив файл до значения параметра **Мощность** (Power), равного 50, обрати внимание, что в этом месте показание текущей мощности больше соответствует значению мощности (табл. 17.2).

Табл. 17.2. Показания текущей мощности, соответствующие значениям мощности с 53 до 44

Измерение №	Мощность	Текущая мощность
49	53	54
50	52	53
51	51	52
52	50	52
53	49	49
54	48	47
55	47	47
56	46	47
57	45	46
58	44	45

График, построенный на основе всего набора данных (рис. 17.3), позволяет лучше понять, как связаны между собой значения параметров **Мощность** (Power) и **Текущая мощность** (Current Power).

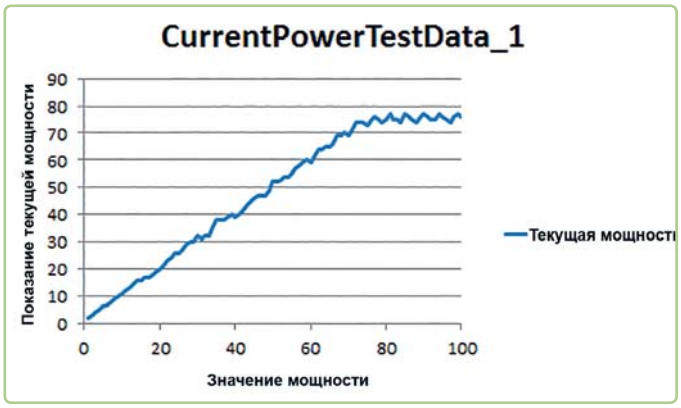


Рис. 17.3. График, построенный на основе значений мощности и показаний текущей мощности

На этом графике видно, что при значении параметра **Мощность** (Power) меньше 70, показание текущей мощности практически совпадает с ним. Эти два значения не всегда одинаковы, поэтому вместо прямой линии на графике видны небольшие отклонения, однако разница между этими значениями никогда не превышает единицы.

Показание текущей мощности достигает максимума, когда значение параметра **Мощность** (Power) равно 75, и остается на этом уровне при более высоких значениях мощности. Оказывается, максимальное значение зависит от конкретного мотора. При использовании мотора C получится похожий набор данных, но с максимальным значением 78 вместо 75. Другие моторы также дают несколько различные значения, хоть и расположенные примерно в одном диапазоне.

Рассмотрим взаимосвязь между параметрами **Текущая мощность** (Current Power) и **Мощность** (Power) более подробно, используя датчик вращения мотора для определения скорости его вращения при разных значениях мощности, и соотнесем полученные данные с показаниями текущей мощности. Для облегчения этой задачи создадим контейнер

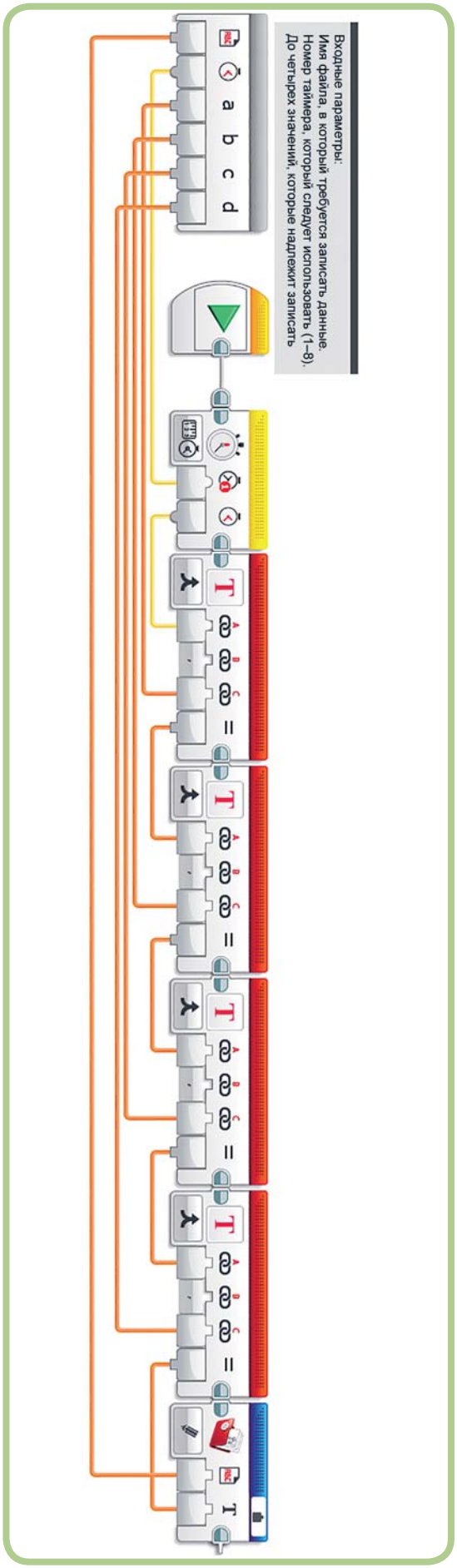


Рис. 17.4. Контейнер ***LogData***

LogData, позволяющий организовать несколько значений в список, разделенный запятыми, и записать их в файл.

Контейнер LogData

В программе *CurrentPowerTest* (см. рис. 17.1) используется блок **Текст** (Text) для объединения параметра **Мощность** (Power) и показания **Текущая мощность** (Current Power) в одно значение, разделенное запятой. Для того чтобы объединить три значения таким образом, понадобятся два блока **Текст** (Text), и для каждого дополнительного значения потребуется дополнительный блок **Текст** (Text). Такой код для форматирования данных может быстро стать слишком громоздким, что затруднит понимание логики программы. Для решения этой проблемы создадим контейнер **LogData** для того, чтобы разгрузить основную программу от кода для форматирования.

В этом блоке также отформатированные данные записываются в файл и к каждому значению с помощью блока **Таймер** (Timer) добавляется *временная метка*, отражающая, когда был зафиксирован результат измерения. Временны́е метки полезны при проведении экспериментов, в которых время является важным фактором. Кроме того, при их использовании можно выявить любые странные паузы и другие проблемы, связанные с временны́ми расчетами, возникающие при выполнении программы.

На рис. 17.4 показан контейнер **LogData**, объединяющий временну́ю метку и до четырех текстовых значений. Большинство значений — это числа. Однако здесь приведены и текстовые параметры, чтобы этот блок можно было использовать и для записи заголовков.

Вот как работает данный контейнер:

1. Номер используемого таймера передается с помощью шины данных. Блоком **Таймер** (Timer) считывается показание соответствующего таймера, затем это значение передается в первый блок **Текст** (Text).

2. В первом блоке **Текст** (Text) объединяются временна́я метка из блока **Таймер** (Timer) и один входной параметр, они разделяются запятой.

3. Значение из предыдущего блока **Текст** (Text) поступает в следующие три блока **Текст** (Text), добавляется запятая и еще один входной параметр.

4. В последнем блоке отформатированные данные записываются в файл, имя которого передается в контейнер «Мой блок» в качестве входного параметра.

В окне **Конструктор Мой блок** (My Block Builder) не показано, какой параметр подключен к какому блоку, но ты можешь просто задать имя и значок для каждого параметра слева направо и при необходимости переместить числовой параметр влево или вправо. Хотя на самом деле порядок не имеет значения. После создания контейнера ты сможешь переместить шины данных так, чтобы каждый входной параметр был подключен к соответствующему блоку.

Программа CurrentPowerTest2

Программа *CurrentPowerTest2* (рис. 17.5) основана на программе *CurrentPower* с добавлением нового измерения. При каждом выполнении цикла после изменения параметра **Мощность** (Power) эта программа считывает показание датчика вращения мотора, приостанавливается на одну секунду и снова считывает показание датчика вращения мотора. Разница между новым и предыдущим показаниями покажет нам, на сколько градусов повернулся мотор в течение этой секундной паузы, что позволит вычислить среднюю скорость в градусах в секунду. Показание текущей мощности мотора также будет зафиксировано, поэтому после сбора всех данных мы сможем установить взаимосвязь между показанием текущей мощности и фактической скоростью мотора.

В начале программы контейнером **LogData** заголовки записываются в файл *CurrentPowerTestData*. С помощью контейнера, находящегося в блоке **Цикл** (Loop), записывается значение параметра **Мощность** (Power), показание **Текущая мощность** (Current Power) и вычисленная скорость.

Выполнение этой программы занимает примерно на 100 секунд больше из-за секундной паузы между проверками показания датчика вращения мотора. По завершении программы скопируй файл *CurrentPowerTestData* на свой компьютер и проанализируй данные.

В табл. 17.3 приведены данные моего теста при значениях мощности, близких к 100 и 50. Скорость примерно в 10 раз превышает показание текущей мощности, поскольку текущая мощность — это фактически скорость вращения мотора, измеряемая в градусах в одну десятую долю секунды!

Табл. 17.3. Данные программы CurrentPowerTest2

Мощность	Текущая мощность	Скорость
100	79	803
99	80	801
98	79	803
97	79	801
96	79	803
95	79	799
94	78	802
93	79	800
92	78	802
…	…	…
50	51	510
49	48	498
48	48	490
47	47	481
46	47	470
45	44	460
44	43	450
43	43	438
42	42	430
41	41	420
40	40	408
39	39	400

Эти данные служат подтверждением того, что при значении параметра **Мощность** (Power) блока **Большой мотор** (Large Motor), равном 10, мотор будет вращаться со скоростью 100 градусов в секунду, а при значении 50 — со скоростью 500 градусов в секунду. Это соотношение сохраняется до тех пор, пока значение параметра **Мощность** (Power) не достигнет отметки 75; с этого момента увеличение мощности не приводит к ускорению вращения мотора. Данное состояние называется *насыщением*. Таким образом, на практике между значениями мощности 80 и 100 нет никакой разницы — оба значения заставляют мотор вращаться со скоростью примерно 750 градусов в секунду.

ПРИМЕЧАНИЕ **Значения текущей мощности и скорости, приведенные в табл. 17.3, не различаются *точно* в 10 раз, поскольку пауза между проверками показаний датчика вращения мотора на самом деле длится чуть больше секунды, что немного увеличивает расчетное значение средней скорости. Кроме того, модуль EV3 постоянно регулирует скорость вращения мотора. Можно более точно измерить промежуток времени между проверками показаний датчика вращения мотора, используя блоки Таймер (Timer), чтобы зафиксировать время перед каждым измерением, а затем вычислить разницу между двумя показаниями таймера. Однако для целей данной программы достаточно и более простого подхода.**

Так почему же мотор не вращается так быстро? На самом деле система EV3 была специально разработана таким образом, и это хорошее решение. Допустим, что с учетом используемого мотора EV3 и ожидаемого уровня заряда аккумулятора инженеры LEGO могут гарантировать скорость вращения каждого мотора, равную 700 градусам в секунду. Они *могли бы* спроектировать систему так, чтобы значение мощности, равное 100, соответствовало бы скорости вращения, равной 700 градусам в секунду, при этом значения мощности и скорости различались бы в 7 раз, а не в 10.

Это позволило бы сохранить соотношение между мощностью и скоростью мотора во всем диапазоне значений мощности, однако при этом нельзя было бы заставить некоторые моторы вращаться с максимальной скоростью. Максимальная скорость, которую можно было бы обеспечить, составляла бы 700 градусов в секунду, а мы знаем, что моторы могут вращаться немного быстрее. Фактический дизайн системы позволяет нам обеспечить максимально возможную скорость вращения мотора за счет потери соотношения между мощностью и скоростью в конце диапазона.

ПРАКТИКУМ 17.1

Запусти программу *CurrentPowerTest2*, используя средний мотор, чтобы посмотреть, как в этом случае связаны между собой параметр **Мощность** (Power), показание **Текущая мощность** (Current Power) и фактическая скорость вращения мотора. Замени в программе блоки **Большой мотор** (Large Motor) блоками **Средний мотор** (Medium Motor) и настрой параметр **Порт** (Port) в трех блоках **Вращение мотора** (Motor Rotation). Кроме того, открепи мотор от подъемного рычага или, по крайней мере, разъедини зубчатые колеса.

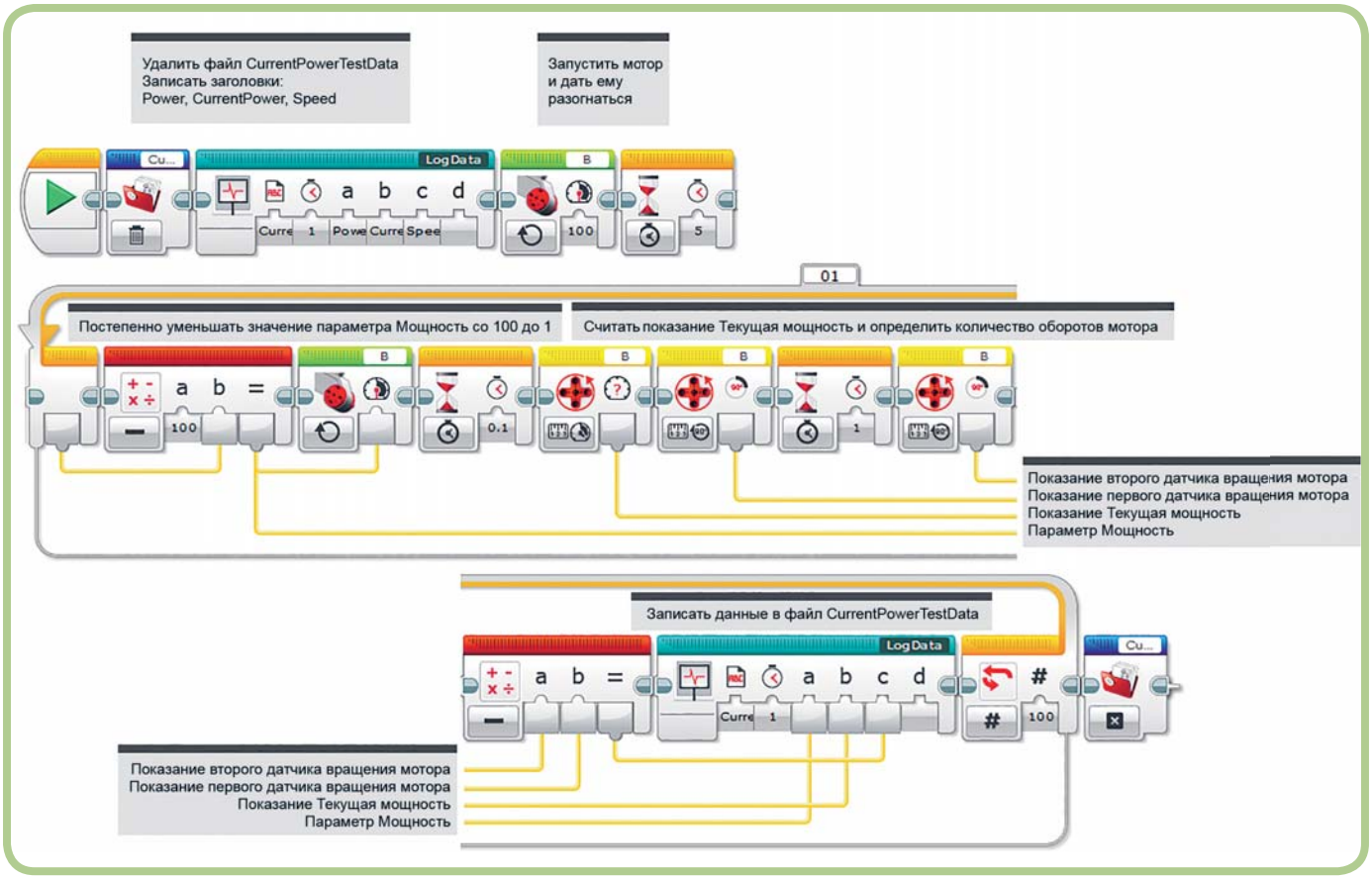


Рис. 17.5. Программа CurrentPowerTest2

Проверка текущей мощности с помощью блока «Рулевое управление»

В программе *CurrentPowerTest* использован блок **Большой мотор** (Large Motor) для определения взаимосвязи между параметром **Мощность** (Power) и показанием **Текущая мощность** (Current Power), которая отображает, как мощность соотносится с фактической скоростью вращения мотора. Однако в большинстве программ мы используем блок **Рулевое управление** (Move Steering). Являются ли эти соотношения актуальными и для данного блока?

Для того чтобы выяснить это, достаточно заменить два блока **Большой мотор** (Large Motor) блоками **Рулевое управление** (Move Steering) и снова запустить программу. На рис. 17.6 показан график, построенный на основе данных, полученных в ходе тестирования. Обрати внимание, при значениях мощности ниже 70 на графике отражено то же соотношение, что и при использовании блока **Большой мотор** (Large Motor). При значениях мощности выше 70 мы наблюдаем большее отклонение, чем в предыдущих тестах. Это связано с тем, что модуль EV3 пытается обеспечить одинаковую скорость вращения двух моторов, что сложно сделать при скорости, близкой к максимальной.

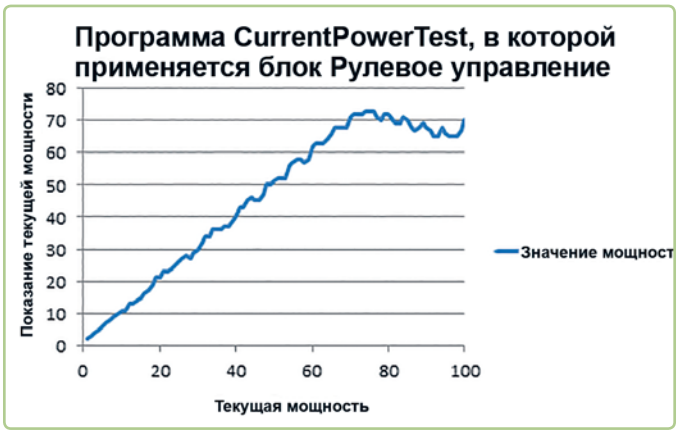


*Рис. 17.6. Проверка соотношения параметра мощности и показания текущей мощности с помощью блока **Рулевое управление** (Move Steering)*

Программа SteeringTest

Следующая программа, *SteeringTest* (рис. 17.7), позволяет выявить взаимосвязь между значением параметра **Рулевое управление** (Steering) блока **Рулевое управление** (Move Steering) и скоростью вращения двух моторов. Сначала программа *SteeringTest* удаляет файл *SteeringTestData*, а затем повторно создает его, записывая в его первой строке "Steering, Motor B, Motor C" в качестве заголовков столбцов. Затем эта программа запускает моторы при значении параметра **Рулевое управление** (Steering), равном 0. Внутри блока **Цикл** (Loop) значение параметра **Рулевое управление** (Steering) постепенно увеличивается с 0 до 100, для чего цикл повторяется 101 раз, поскольку необходимо включить в диапазон оба крайних значения. На каждом этапе контейнером **LogData** фиксируются значения параметра **Рулевое управление** (Steering) и показания **Текущая мощность** (Current Power) двух моторов. Для параметра **Мощность** (Power) блоков **Рулевое управление** (Move Steering) задано значение 50, которое находится в пределах диапазона, в котором значение мощности прямо зависит от показания текущей мощности.

После запуска этой программы оба мотора будут вращаться на протяжении примерно 15 секунд. После завершения программы ты сможешь загрузить файл *SteeringTestData* с модуля EV3 на свой компьютер.

На рис. 17.8 показан график, построенный на основе полученных данных. Показание текущей мощности для мотора B постоянно держится около отметки 50 (и варьируется в диапазоне от 49 до 51). Показание текущей мощности для мотора C начинается с 50 и уменьшается до –50 по мере того, как значение параметра **Рулевое управление** (Steering) увеличивается от 0 до 100.

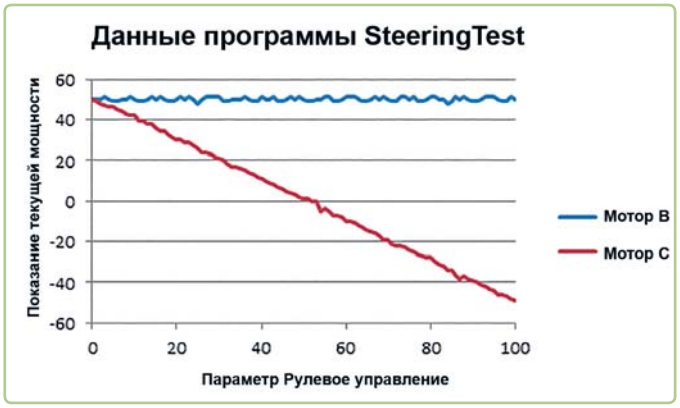


*Рис. 17.8. Зависимость показания текущей мощности от значения параметра **Рулевое управление** (Move Steering)*

Когда значение параметра **Рулевое управление** (Steering) равно 0, текущая мощность обоих моторов составляет 50, что вполне объяснимо, поскольку при таком значении параметра **Рулевое управление** (Steering) робот должен двигаться прямо. По мере увеличения значения параметра **Рулевое управление** (Steering) вращение мотора C замедляется, в результате чего робот совершает поворот.

Когда значение параметра **Рулевое управление** (Steering) равно 50, текущая мощность мотора C равна 0. Это означает, что он вообще не вращается. При этом значении параметра **Рулевое управление** (Steering) колесо мотора C остается неподвижным, а колесо мотора B продолжает вращаться, что заставляет робота поворачиваться на месте вокруг центра колеса мотора C. При значении параметра **Рулевое управление** (Steering), равном 100, текущая мощность мотора C составляет –50, поэтому он вращается так же быстро, как и мотор B, но в противоположном направлении. В результате робот вращается вокруг точки, находящейся между двумя колесами. При любом значении параметра **Рулевое**

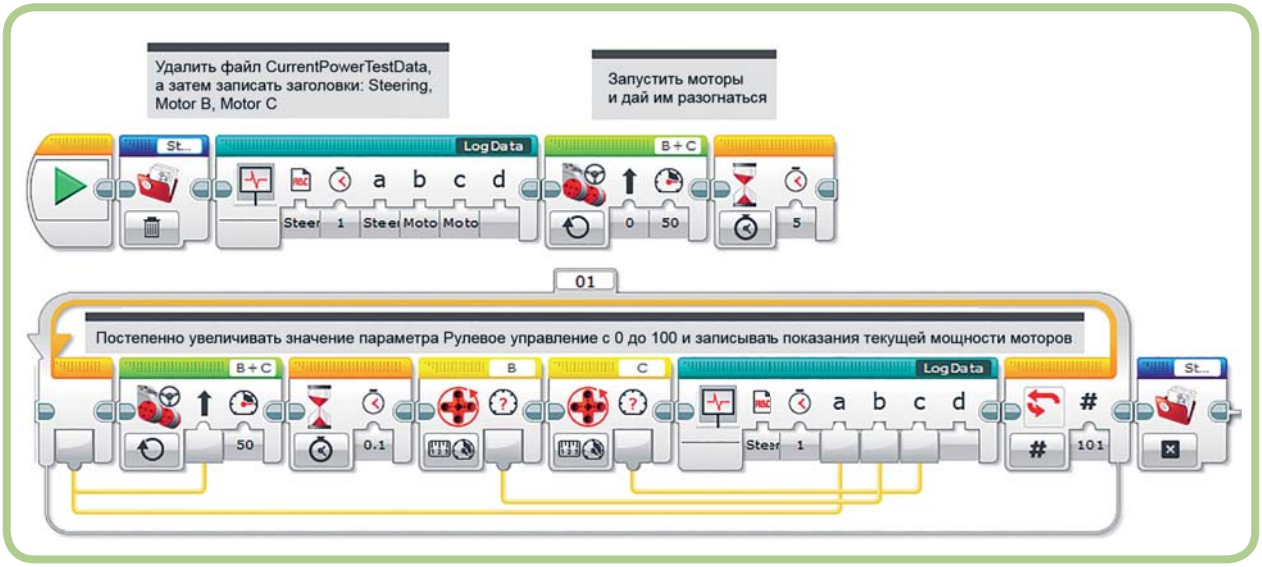


Рис. 17.7. Программа SteeringTest

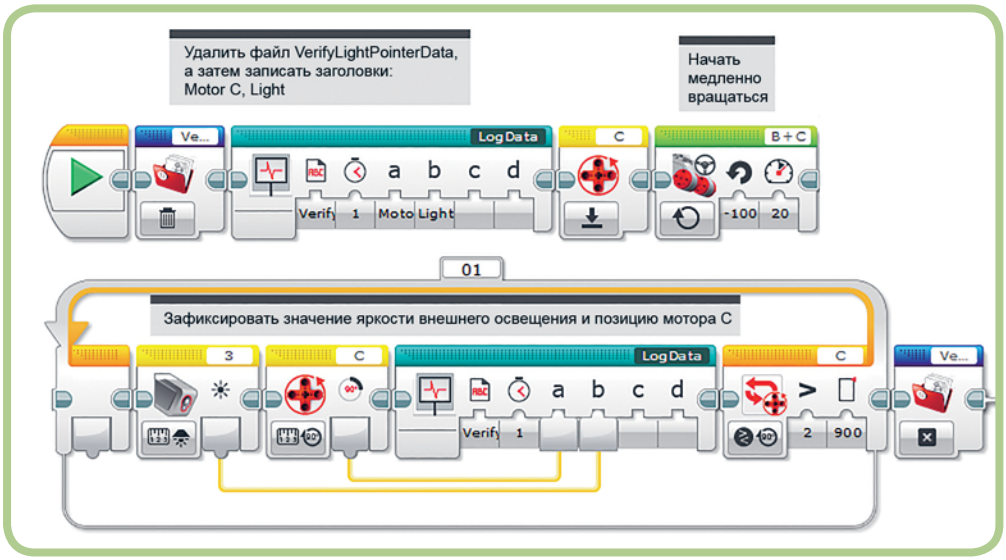


Рис. 17.9. Программа VerifyLightPointer

управление (Steering) в диапазоне от 50 до 100 робот будет вращаться; единственное различие заключается в центре вращения.

Если ты хочешь, чтобы твой робот продолжал двигаться вперед во время поворота, значение параметра **Рулевое управление** (Steering) никогда не должно быть больше 50. Несмотря на то что диапазон значений параметра **Рулевое управление** (Steering) при движении в одном направлении составляет от 0 до 100, для движения вперед следует использовать значения от 0 до 40. При повороте в другом направлении используй значения от 0 до –40. Значения выше 40 (или ниже –40) заставляют робота вращаться или двигаться по очень маленькому кругу.

Программа VerifyLightPointer

В программе *LightPointer* (см. гл. 11) используется датчик цвета для направления робота TriBot в сторону источника света. Робот вращается вокруг своей оси и запоминает положение, соответствующее максимальному уровню освещенности. После совершения полного оборота робот возвращается в сохраненное положение, в результате чего датчик цвета указывает на источник света.

Программа *LightPointer* предполагает, что датчик может обнаружить самый яркий уровень освещенности в процессе вращения робота. Если это предположение не верно, программа не будет работать. Например, возникнет сбой, если вследствие слишком быстрого вращения робот не сможет точно замерить уровень освещенности или если в помещении слишком много источников внешнего освещения, что не позволяет роботу определить направление, в котором находится источник света. Ты можешь проверить это предположение, собрав и проанализировав показания датчика в ходе эксперимента. Программа *VerifyLightPointer*, показанная на рис. 17.9, позволяет решить эту задачу.

Данная программа представляет собой комбинацию программы *LightPointer* и программ для регистрации данных. Сначала с помощью нее удаляется файл *VerifyLightPointerData*, а затем применяется блок **LogData** для записи заголовков столбцов “Motor C” и “Light” для файла.csv. Показание датчика вращения мотора C сбрасывается, после чего робот TriBot начинает медленно вращаться. При выполнении цикла в файл записываются значения яркости внешнего освещения, полученные от датчика цвета, а также положение мотора C. Цикл повторяется до тех пор, пока мотор C не обернется на 900° (при использовании образовательной версии конструктора используй значение 700°). На последнем блоке закрывается файл *VerfiyLightPointerData*.

Расположи робота TriBot и источник света как на рис. 17.10, чтобы источник света находился слева от робота под углом 90°, а затем запусти программу. Робот должен медленно вращаться по кругу, а затем остановиться. По завершении программы ты сможешь загрузить файл *VerifyLightPointerData* на свой компьютер и проверить показания датчика цвета и датчика вращения мотора.

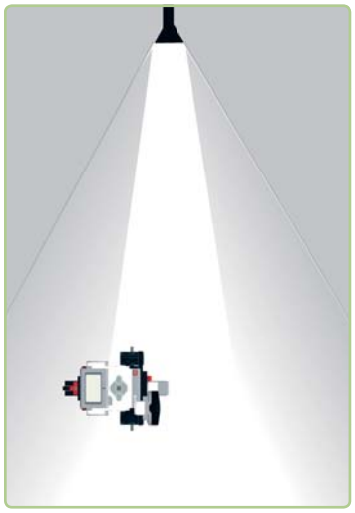


Рис. 17.10. Начальное положение робота при выполнении программы VerifyLightPointer

На рис. 17.11 показан график, построенный на основе измерений, произведенных во время моего тестирования. Уровень освещенности значительно увеличивается по мере поворота робота в сторону фонарика и образует один большой пик. При анализе численных данных (табл. 17.4) можно заметить, что между моментами считывания показаний датчика цвета мотор C поворачивает только на 1°. Мотор должен повернуть примерно на 840°, чтобы выполнить полный оборот, поэтому робот передвигается на очень незначительное расстояние между моментами считывания показаний датчика. В связи с этим вероятность того, что датчик не зафиксирует максимальный уровень освещенности, крайне мала.

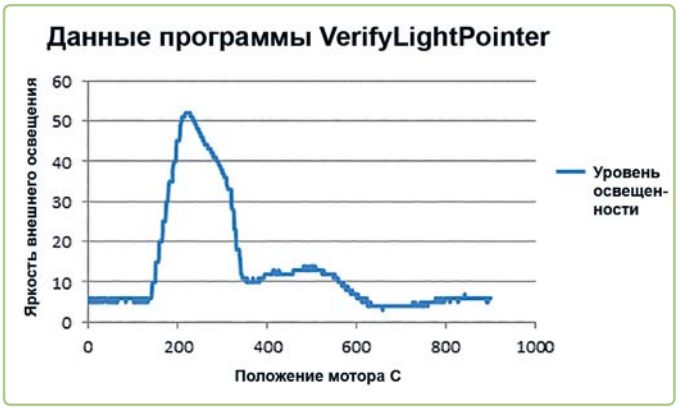


Рис. 17.11. Уровень внешнего освещения, соответствующий разным положениям мотора C

Табл. 17.4. Показание датчика вращения мотора

Положение мотора C	Уровень освещенности
2	6
3	6
3	4
4	6
5	6
5	5
6	5
7	5
7	5
9	6
9	6
10	6

Согласно этим данным, программа *LightPointer* должна корректно определять направление, в котором находится источник света. Если бы на графике были бы лишь небольшие отклонения в уровне освещенности, зафиксированные во время вращения робота, или на нем было бы видно несколько пиков, то такой уверенности в корректной работе программы *LightPointer* не было бы.

Управление количеством данных

Программа *VerifyLightPointer* максимально быстро собирает и записывает данные, создавая большой файл. Чего нельзя сказать о большинстве программ регистрации данных, при использовании которых тебе придется уделить больше внимания и проконтролировать частоту их фиксации.

Многие программы регистрации данных будут иметь ту же структуру, что и программа *VerifyLightPointer.* После того как с помощью нескольких блоков произойдет начальная настройка, в блоке **Цикл** (Loop) сформируется код для сбора и записи данных в файл. Ты можешь контролировать частоту фиксации данных, добавив блок **Ожидание** (Wait) в конец тела блока **Цикл** (Loop).

Какую паузу должен обеспечить блок **Ожидание** (Wait)? Это зависит от того, сколько времени, по-твоему, будет длиться эксперимент, а также от частоты изменения собираемых данных. Тебе следует записывать данные достаточно часто, чтобы не пропустить каких-либо важных изменений, но не настолько часто, чтобы в итоге получить огромный файл данных или заполнить всю память. Правильный баланс обычно можно найти методом проб и ошибок, поэтому не удивляйся, если тебе потребуется несколько раз изменить настройки, чтобы получить нужный результат.

Например, для того чтобы изменить программу *VerifyLightPoint* так, чтобы она делала 20 измерений в секунду, необходимо добавить паузу в конце блока **Цикл** (Loop) длительностью в одну двадцатую секунды или 0,05 секунды. На рис. 17.12 показан блок **Ожидание** (Wait), добавленный в основной цикл программы.

Теперь программа будет приостанавливаться на 0,05 секунды при каждом выполнении цикла, а значит, данные будут регистрироваться примерно 20 раз в секунду. Количество измерений в секунду необязательно будет *равно* 20, поскольку на управление моторами, сбор показаний датчиков и запись данных требуется некоторое время.

ПРАКТИКУМ 17.2

С помощью программы *VerifyLight* можно собрать достаточный объем данных для определения направления источника света, соответствующего большому пику на графике, поскольку робот TriBot вращается очень медленно. Измени значение параметра Мощность (Power) в блоке Рулевое управление (Move Steering) с 20 на 40 и проведи повторный тест для оценки разницы. Какова максимальная скорость вращения робота, при которой на графике все еще присутствует явно выраженный пик?

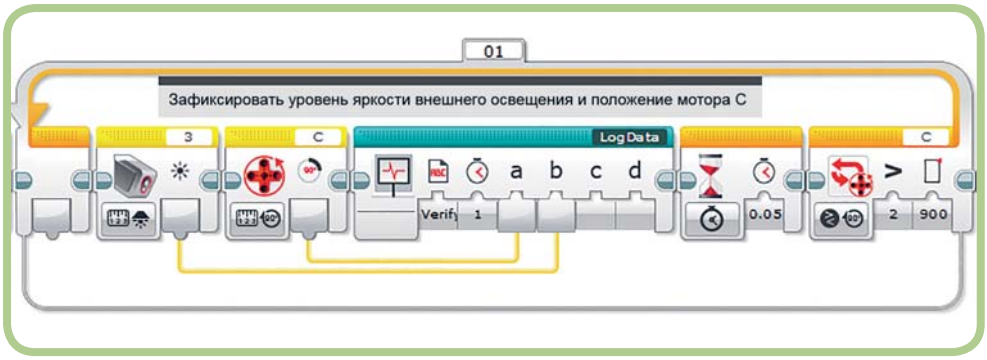


Рис. 17.12. Пауза продолжительностью 0,05 секунды

Дальнейшее исследование

Далее описаны дополнительные упражнения, связанные с регистрацией данных:

1. Проведи эксперимент, чтобы выяснить, как изменяются показание **Приближение** (Proximity) инфракрасного датчика (из домашней версии конструктора) или **Расстояние** (Distance) ультразвукового датчика (из образовательной версии) по мере удаления робота TriBot от объекта. Используй положение одного из моторов для измерения истинного расстояния до объекта. Расположи робота очень близко к объекту и заставь его медленно удаляться от него. Поэкспериментируй с объектами разного цвета и текстуры (например, твердая стена и висящее полотенце могут дать разные результаты).

2. Показание ультразвукового датчика соответствует истинному расстоянию, поэтому оно должно изменяться точно в соответствии с вращением мотора вплоть до момента, когда датчик перестанет обнаруживать объект.

3. Напиши программу, которая определяет, как изменяется значение **Направление маяка** (Beacon Heading) при изменении угла между роботом TriBot и удаленным инфракрасным маяком. Сначала помести маяк непосредственно перед роботом и фиксируй значение **Направление маяка** (Beacon Heading) по мере вращения робота (похожий подход был применен в программе *VerifyLightPointer*).

4. Используя режим **Измерение** (Measure) ⇒ **Скорость** (Rate) гироскопического датчика, измеряющего вращение в градусах в секунду, напиши программу, позволяющую определить взаимосвязь между значением параметра **Мощность** (Power) блока **Рулевое управление** (Move Steering) и скоростью вращения робота TriBot (при значении параметра **Рулевое управление** (Steering), равном 100). Сначала собери данные на низкой скорости. Затем увеличь скорость, чтобы проверить, сохраняется ли прежнее соотношение, и есть ли момент, когда скорость вращения робота становится настолько высокой, что гироскопический датчик уже не может предоставить надежные данные.

Заключение

С помощью модуля EV3 можно собирать, форматировать и записывать данные, полученные от различных датчиков, что делает его отличным регистратором данных. В этой главе приведено множество примеров, в которых описаны все действия, которые требуется выполнить при написании типичной программы регистрации данных. В том числе рассмотрены процессы создания файла данных, сбора показаний датчика, записи данных в файл с временной меткой и даже осуществления контроля над частотой фиксации данных.

Знания о регистрации данных позволяют тебе больше узнать о двигателях и датчиках EV3. Например, при использовании программы *CurrentPowerTest* была раскрыта тайна показания **Текущая мощность** (Current Power) блока **Вращение мотора** (Motor Rotation), мы смогли узнать о том, за что на самом деле отвечает параметр **Мощность** (Power) блока **Рулевое управление** (Move Steering). С помощью программы *MoveSteeringTest* удалось собрать данные, показавшие, как параметр **Рулевое управление** (Steering) влияет на движение робота, выявить диапазон наиболее полезных значений. В программе *VerifyLightPoint* регистрация данных использовалась для того, чтобы на основе результата эксперимента доказать, что программа *LightPointer* будет работать должным образом.

Ты можешь применять датчики EV3 дома или в классе для проведения экспериментов, которые не имеют отношения к робототехнике! Используй датчик цвета для сравнения уровня яркости лампочек различных марок или измерь площадь и объем с помощью датчика вращения мотора.

Для проведения еще большего количества экспериментов приобрети датчик температуры от LEGO Education (**www.legoeducation.com**) или один из многих EV3-совместимых датчиков от компаний HiTechnic (**www.hitechnic.com**), Mindsensors (**www.mindsensors.com**) или Vernier (**www.vernier.com**).

18

Многозадачность

В этой главе ты узнаешь о параллельном выполнении группы блоков, благодаря чему робот может одновременно выполнять несколько задач, т. е. о *многозадачности*. Например, одна часть твоей программы может управлять движением робота, а другая — сбором показаний датчика.

Сначала выясним, как добавить в программу *AroundTheBlock* простой одометр для измерения пройденного расстояния, а затем узнаем о способе добавления мигающей подсветки в программу *DoorChime*. Кроме того, мы обсудим правила, касающиеся процесса выполнения программы при использовании параллельных последовательностей блоков и синхронизации действий между ними.

Несколько блоков «Начало»

В программе EV3 группа соединенных друг с другом блоков называется *последовательностью*. Во всех созданных ранее программах использована одна последовательность, начинающаяся с блока **Начало** (Start), который появляется автоматически при создании новой программы. Многозадачность просто предполагает объединение нескольких последовательностей в одну программу.

Один из способов применения в своей программе нескольких последовательностей заключается в добавлении еще одного блока **Начало** (Start). Например, на рис. 18.1 показана версия программы *AroundTheBlock* (см. гл. 4), в которой используются две последовательности. Эти *параллельные последовательности* выполняются одновременно. Последовательность в верхней части изображения заставляет робота TriBot двигаться по квадратной траектории, а последовательность в нижней части изображения отображает положение мотора. После завершения программы будет показано количество оборотов мотора B. Ты можешь использовать подобный метод для измерения пройденного роботом расстояния при выполнении программы *LineFollower* или *WallFollower*.

Для создания этой программы выполни следующие действия:

1. Создай новый проект под названием *Chapter18*.
2. Открой проект *Chapter4* и скопируй программу *AroundTheBlock* в проект *Chapter18*.
3. Перетащи блок **Начало** (Start) с вкладки палитры программирования, содержащей блоки управления операторами.
4. Добавь блоки, показанные в нижней части рис. 18.1. В качестве единиц измерения установи значение **Градусы** (Degrees).

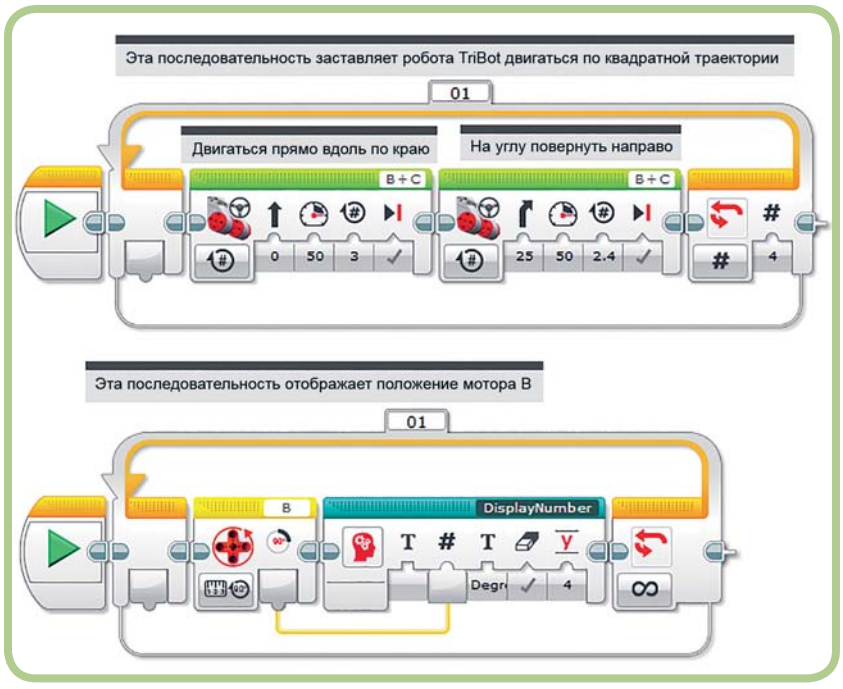


Рис. 18.1. Отображение положения мотора при перемещении по квадратной траектории

Выполнение данной программы начинается с того, что в модуле EV3 сначала запускаются оба блока **Цикл** (Loop), а затем происходит быстрое переключение между блоками каждой последовательности. *На самом деле* в компьютере модуля EV3 невозможно выполнять более одного действия за раз, однако он *может* быстро переключаться между двумя задачами, выполняя часть одной из них, а затем часть другой. Это переключение происходит так быстро, что его невозможно заметить.

После запуска программы робот TriBot должен передвигаться по квадратной траектории, при этом на экране модуля должно отображаться пройденное роботом расстояние. Даже после того, как робот опишет весь квадрат, программа продолжит отображать положение мотора B. Если ты возьмешь робота в руки и повернешь мотор, то увидишь, как изменится отображаемое на экране положение мотора.

Исходная программа *AroundTheBlock* завершается после того, как робот опишет квадрат, поскольку блок **Цикл** (Loop) настроен всего на четыре повтора, и после него в программе блоков нет. Когда в программе используется несколько последовательностей, она продолжает работать до тех пор, пока *все* они не будут выполнены. Блок **Цикл** (Loop), который отображает положение мотора, настроен на бесконечное повторение, поэтому программа будет работать до тех пор, пока ты ее не остановишь.

Блок «Остановить программу»

Для того чтобы остановить программу, ты, конечно, можешь просто взять робота в руки и нажать кнопку модуля «Назад», однако есть более удобный способ. Блок **Остановить программу** (Stop Program), отображенный на рис. 18.2, находится на вкладке с блоками дополнений, позволяет завершить выполнение всех запущенных последовательностей и остановить программу. Помести этот блок в конец верхней последовательности измененной программы *AroundTheBlock*, как показано на рис. 18.3, чтобы завершить программу после того, как робот TriBot опишет полный квадрат.

*Рис. 18.2. Блок **Остановить программу** (Stop Program)*

В данном примере блок **Стоп** (Stop Program) помещается в конце последовательности. Также ты можешь использовать его внутри блока **Переключатель** (Switch), если захочешь остановить выполнение программы только при определенных условиях.

Избегание цикла активного ожидания

Добавление второй задачи может отрицательно сказаться на программах, зависящих от быстрой реакции на показания датчиков, например, *LineFollower*. На рис. 18.4 показан код одометра, добавленного в программу *LineFollower*, которая рассмотрена в гл. 13.

Перед добавлением второй задачи можно было задать значение 50 для параметра **Мощность** (Power) блока **Рулевое управление** (Move Steering), и программа работала бы нормально. После добавления кода одометра максимальное значение параметра **Мощность** (Power), при котором сохраняется прежняя степень надежности, составляет 35.

Код для отображения положения мотора представляет собой пример *цикла активного ожидания*, который повторяется максимально быстро, вследствие чего используется большая часть вычислительной мощности модуля EV3. Это замедляет выполнение кода, отвечающего за движение вдоль линии, поэтому робот недостаточно быстро реагирует на изменения в ее направлении. Мы можем решить эту проблему, замедлив выполнение цикла отображения и позволив модулю EV3 выделить больше ресурсов на обеспечение движения робота вдоль линии. Добавление секундной паузы в блок **Цикл** (Loop) (рис. 18.5) для того, чтобы показание на экране обновлялось лишь один раз в секунду, а не максимально часто, позволяет задать значение 50 для параметра **Мощность** (Power). Даже при такой задержке отображаемое положение мотора должно быть достаточно точным, чтобы его можно было использовать.

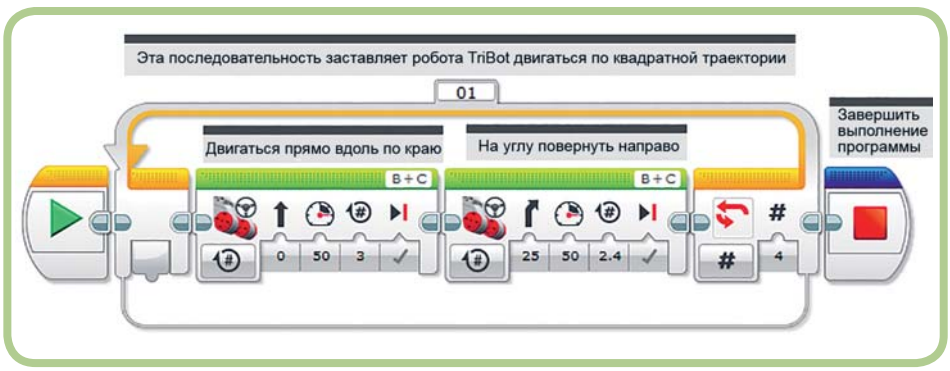


Рис. 18.3. Остановка программы после того, как робот опишет полный квадрат

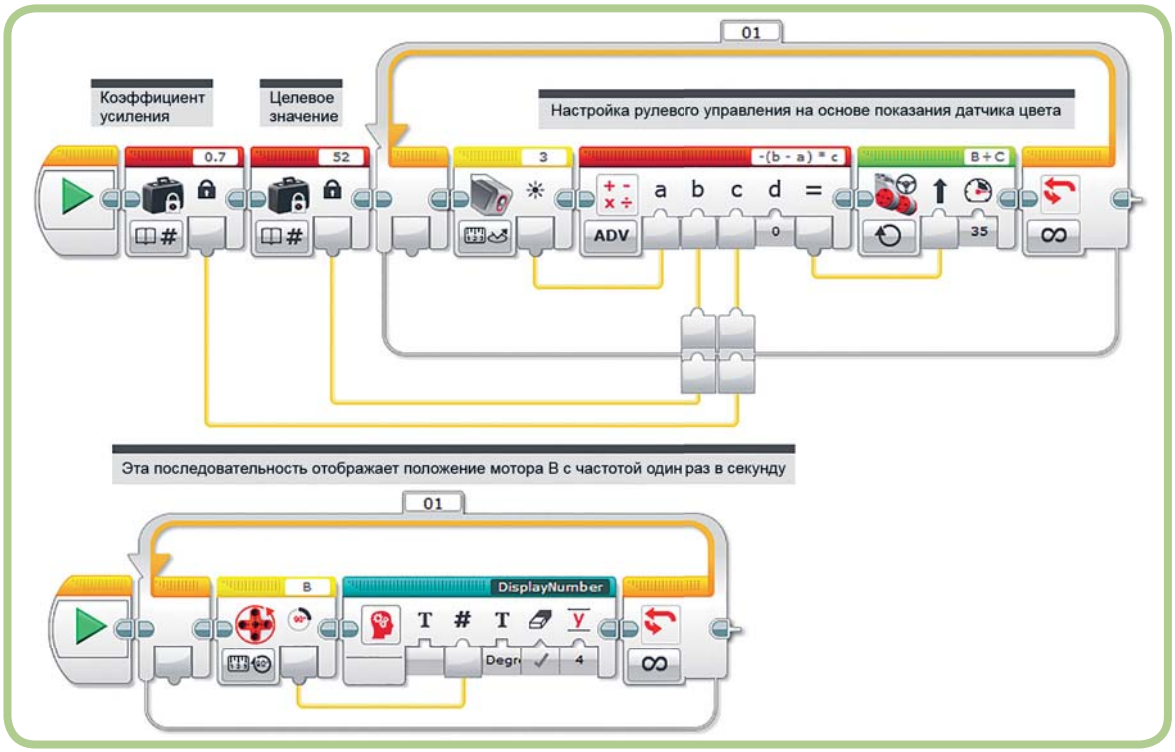


Рис. 18.4. Одометр, добавленный в программу LineFollower

Рис. 18.5. Замедление выполнения цикла активного ожидания

Добавление мигающей подсветки в программу DoorChime

В новой версии программы *AroundTheBlock* используются два блока **Начало** (Start) для выполнения двух независимых задач на протяжении всей работы программы. В этом разделе мы поместим несколько последовательностей в середину программы. Используй этот подход, если хочешь, чтобы два действия были выполнены на конкретных этапах работы программы.

ПРАКТИКУМ 18.1

Для того чтобы избежать усложнения программы *AroundTheBlock*, расстояние измеряется одометром в градусах вращения мотора, а не в сантиметрах или дюймах. Определи коэффициент преобразования, который требуется для перехода от градусов к сантиметрам или дюймам, и добавь блок **Математика** (Math) для преобразования показаний одометра в более удобные единицы измерения.

СОВЕТ Обратись к описанию программы *ThereAndBack* в гл. 4, чтобы освежить в памяти преобразование градусов вращения в сантиметры или дюймы.

Рис. 18.6. Программа DoorChime, рассмотренная в гл. 12

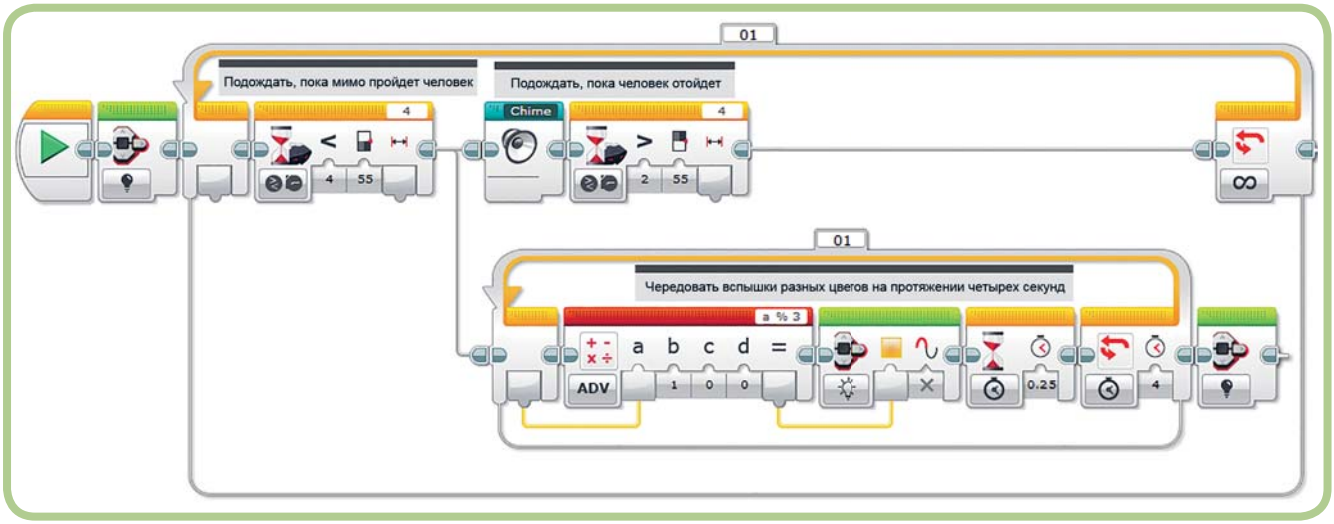


Рис. 18.7. Мигание индикатора состояния модуля во время воспроизведения звукового сигнала

Мы изменим программу *DoorChime* из гл. 12 (рис. 18.6) так, чтобы индикатор состояния модуля мигал во время воспроизведения звукового сигнала. Поскольку подсветка должна мигать только пока воспроизводится звуковой сигнал, следует добавить вторую последовательность в соответствующей точке программы вместо того, чтобы создавать совершенно новую последовательность, используя блок **Начало** (Start).

После внесения изменений, показанных на рис. 18.7, программа сначала отключает индикатор состояния модуля, а затем входит в цикл и ждет, пока мимо пройдет человек. В этот момент запускается контейнер **Chime** вместе с блоками другой последовательности. С помощью блоков второй последовательности включается подсветка индикатора состояния модуля, которая поочередно вспыхивает тремя цветами с паузой в четверть секунды. Остаток от деления значения **Параметр цикла** (Loop Index) на 3 определяет цвет, используемый блоком **Индикатор состояния модуля** (Brick Status Light), поэтому данное значение циклически изменяется в пределах от 0 до 2. Блоку **Chime** требуются четыре секунды на воспроизведение всех нот. По этой причине для того, чтобы индикатор состояния модуля мигал на протяжении такого же периода времени, блок **Цикл** (Loop) настроен на повторение в течение четырех секунд. После завершения цикла подсветка индикатора состояния модуля отключается.

Тонкая серая линия, соединяющая блоки твоей программы, называется *шиной последовательности*. Она соединяет *выходной разъем* в правой части одного блока с *входным разъемом* в левой части другого блока, как показано на рис. 18.8.

Рис. 18.8. Шина последовательности и разъемы

Для добавления новой последовательности достаточно перетащить новую шину последовательности от выходного разъема одного блока к входному разъему другого блока. Выполни следующие действия, чтобы создать новую программу *DoorChime*.

1. Скопируй программу *DoorChime* из проекта *Chapter12* в проект *Chapter18*.

2. Выдели блок **Цикл** (Loop) и перетащи средний маркер на его нижней границе, чтобы создать область для добавления еще одной последовательности (рис. 18.9).

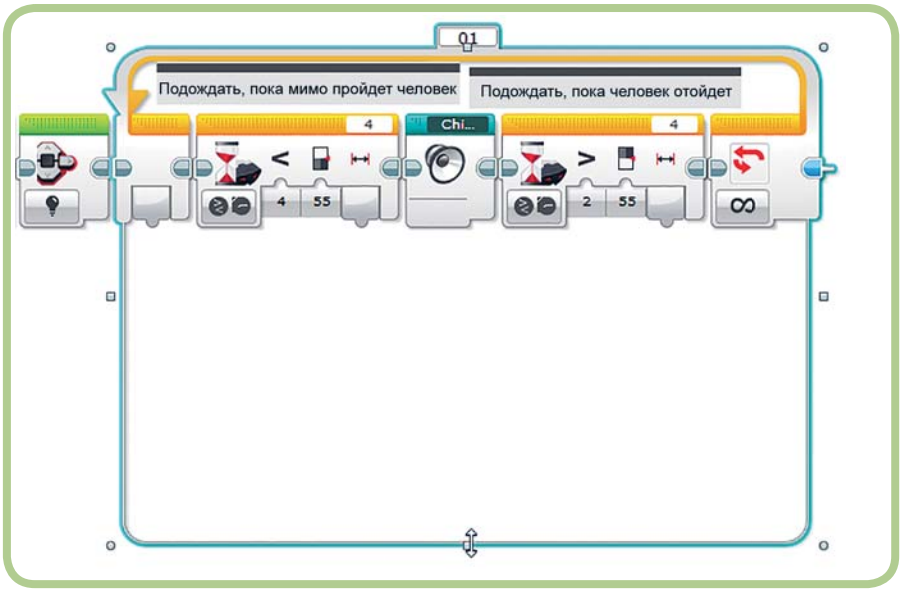


Рис. 18.9. Создание области для второй последовательности

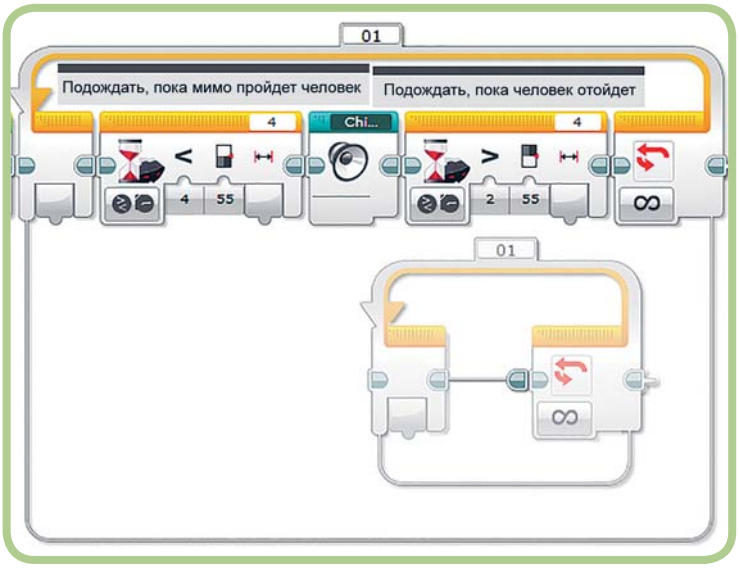


Рис. 18.10. Добавление нового блока ***Цикл***

3. Перетащи новый блок **Цикл** (Loop) в существующий и помести его под контейнером **Chime** чуть справа (рис. 18.10). Изображение нового блока будет немного бледным, поскольку он еще не подключен к программе.

4. Щелкни по выходному разъему блока **Инфракрасный датчик** (Infrared Sensor) и перетащи, чтобы создать новую шину последовательности (рис. 18.11).

5. Подключи шину последовательности к входному разъему с левой стороны нового блока **Цикл** (Loop). Теперь его изображение должно стать ярким (рис. 18.12).

6. После подключения блока **Цикл** (Loop) к программе добавь оставшиеся блоки и шины данных в соответствии с рис. 18.7.

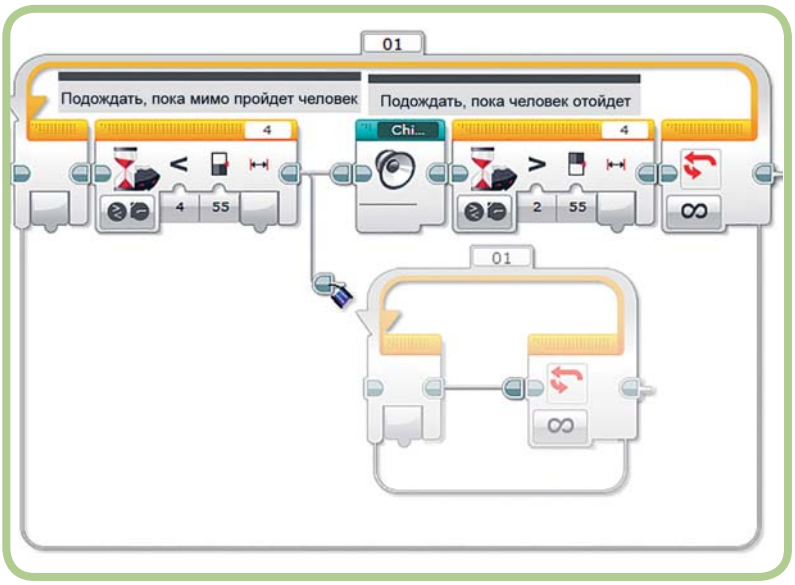


Рис. 18.11. *Перетаскивание шины последовательности*

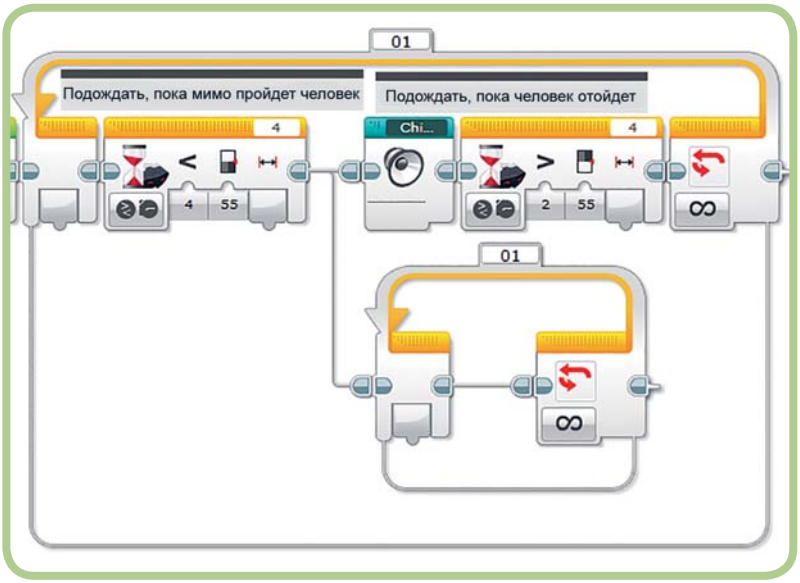


Рис. 18.12. Подключение нового блока ***Цикл*** *(Loop)*

Теперь запусти программу. Когда кто-нибудь пройдет мимо робота TriBot, во время воспроизведения звукового сигнала подсветка индикатора состояния модуля будет мигать тремя цветами.

Правила, касающиеся процесса выполнения программы

Использование нескольких последовательностей усложняет процесс выполнения программы в нескольких отношениях. Например, мы уже видели, что программа не завершается до тех пор, пока она не достигнет конца *всех* последовательностей или не будет остановлена блоком **Остановить программу** (Stop Program). В этом разделе ты узнаешь о других правилах, связанных с процессом выполнения программы, на которые влияет использование нескольких последовательностей. Здесь также будет описана пара простых программ для демонстрации соответствующих эффектов.

Запуск блоков и шины данных

Блок может быть запущен только после того, как во всех подключенных к нему шинах данных появятся значения, что видно на примере программы *BlockStartTest* (рис. 18.13). В блоке **Экран** (Display) в верхней последовательности отображается значение 1, а с помощью блока **Константа** (Constant) значение 2 записывается в шину данных, подключенную к нижнему блоку **Экран** (Display). Блок **Экран** (Display) в нижней последовательности не будет запущен до тех пор, пока с помощью блока **Константа** (Constant) значение 2 не поместится в шину данных. После запуска этой программы на экране модуля отобразится значение «1», а затем после секундной паузы к нему добавится значение «2».

Несмотря на то что обычно блоки и шины данных обрабатываются слева направо, на блоки разных последовательностей это правило не распространяется. Блок **Экран** (Display) в нижней последовательности находится слева от блока **Константа** (Constant). Однако он не будет выполнен до тех пор, пока не запустится блок **Константа** (Constant). Ты можешь переместить блок **Экран** (Display) вправо, что сделает работу программы более наглядной, однако имей в виду, что момент запуска блока определяет шина данных, а не его местоположение. Старайся упорядочивать блоки и последовательности в соответствии с поведением программы.

Блоки **Цикл** (Loop) и **Переключатель** (Switch) подчиняются тому же правилу, как видно на примере программы *LoopStartTest* (рис. 18.14). Блок **Цикл** (Loop) не может быть запущен до тех пор, пока значение с помощью блока **Константа** (Constant) не поместится в шину данных, несмотря на то, что это значение не используется до запуска второго блока **Экран** (Display) в блоке **Цикл** (Loop). После запуска программа отображает значение «1» (из блока **Экран** (Display) в верхней последовательности), а затем приостанавливается на одну секунду. После того как блоком **Константа** (Constant) значение помещено в шину данных, запускается блок **Цикл** (Loop). Отображается значение «2», затем программа берет паузу в течение еще одной секунды перед тем, как отобразить значение «3».

Программы *BlockStartTest* и *LoopStartTest* были созданы специально для демонстрации влияния шин данных на порядок выполнения блоков (а не для обеспечения интересного поведения робота, как ты мог подумать). Несмотря на то что применять шины данных для передачи значений между последовательностями очень удобно, следует обращать пристальное внимание на возникающие при этом зависимости. Использование нескольких последовательностей наиболее уместно, когда выполняемые ими задачи не зависят друг от друга.

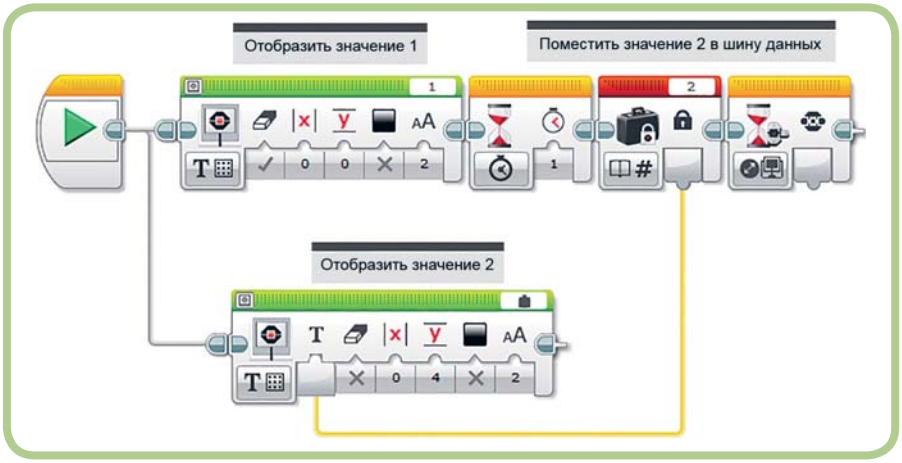


Рис. 18.13. Программа BlockStartTest

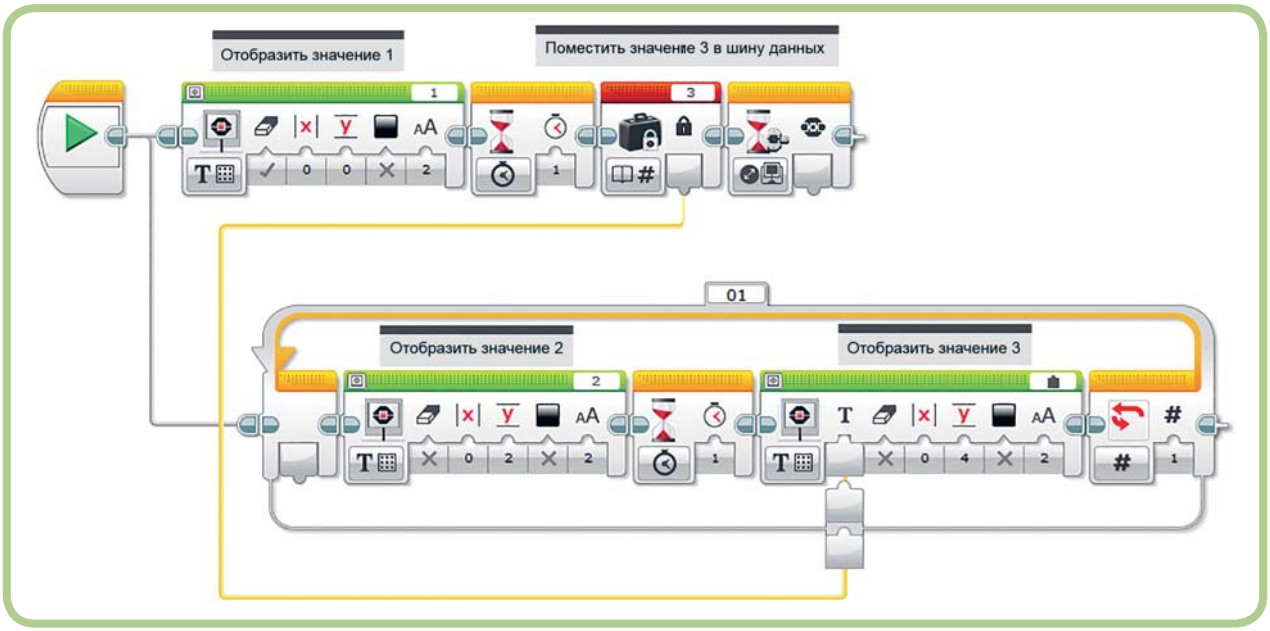


Рис. 18.14. Программа LoopStartTest

Использование значений из блоков «Цикл» или «Переключатель»

В шине данных, которая начинается внутри блока **Цикл** (Loop) и подключается к блоку, находящемуся вне блока **Цикл** (Loop), значение появляется только после выполнения блока **Цикл** (Loop). Это правило продемонстрировано на примере программы *LoopCountTest* (рис. 18.15). Блок **Цикл** (Loop) повторяется пять раз, приостанавливаясь в общей сложности на пять секунд. На блоке **Экран** (Display) в нижней последовательности отображается значение, попавшее в шину данных из вывода **Параметр цикла** (Loop Index) блока **Цикл** (Loop). Поскольку блок **Экран** (Display) находится вне блока **Цикл** (Loop), только последнее значение (4) передается ему с помощью шины данных и только после выполнения блока **Цикл** (Loop). После запуска эта программа приостанавливается на пять секунд, после чего отображается значение «4». Это правило также распространяется на блок **Переключатель** (Switch); в шинах данных, которые выходят из блока **Переключатель** (Switch), значение появляется только после завершения блока **Переключатель** (Switch).

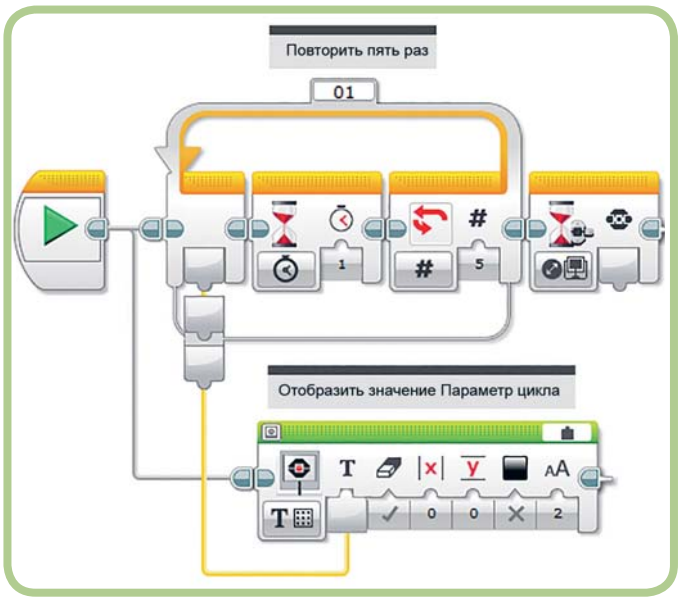


Рис. 18.15. *Программа LoopCountTest*

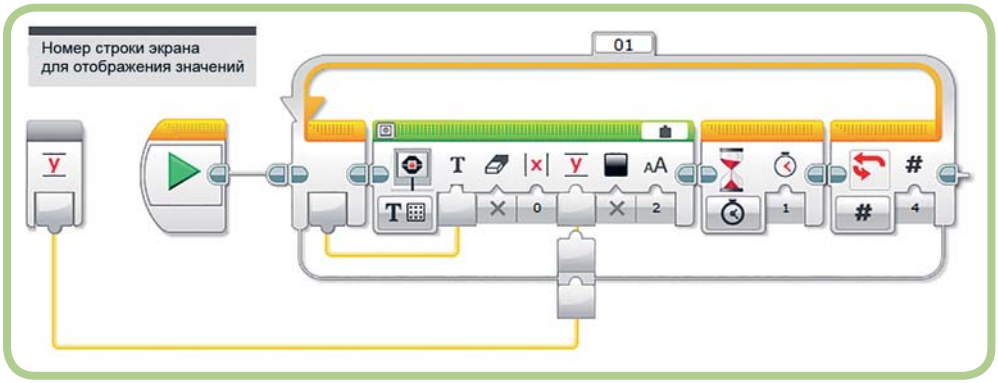


Рис. 18.16. Контейнер ***DisplayCount***

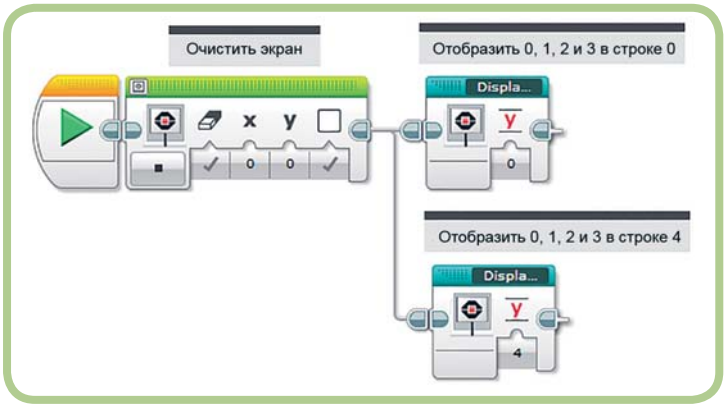


Рис. 18.17. Программа MyBlockTest

Использование контейнеров «Мой блок»

Одновременно может выполняться лишь одна копия контейнера «Мой блок», что продемонстрировано на примере контейнера **DisplayCount** (рис. 18.16). С помощью этого блока на экран модуля выводятся значения 0, 1, 2 и 3 с интервалом в одну секунду, при этом номер строки принят в качестве входного параметра.

В программе *MyBlockTest* (рис. 18.17) используются два экземпляра контейнера **DisplayCount**. После запуска этой программы отображаются числа «0», «1», «2» и «3» с паузой в одну секунду на строке 4 или 0. После эти четыре числа отображаются уже на другой строке. В ходе тестов значения всегда отображались сначала на строке 4, но это может быть простым совпадением. В любом случае один из двух контейнеров «Мой блок» будет выполнен перед запуском другого.

Это правило распространяется только на контейнеры «Мой блок». Если ты заменишь два блока **DisplayCount** блоками **Цикл** (Loop), содержащими блоки **Экран** (Display), то эти две группы чисел будут отображаться на экране модуля одновременно. Правило также распространяется только на две копии одного и того же контейнера «Мой блок». Два блока **DisplayCount** не будут выполняться одновременно, а блоки **DisplayCount** и **DisplayNumber** — будут.

ПРИМЕЧАНИЕ **Если тебе все-таки необходимо обеспечить параллельное выполнение двух копий одного и того же контейнера «Мой блок», создай копию контейнера и дай ему имя, отличное от имени первого контейнера. Переименованную копию можно запустить одновременно с оригиналом.**

Это поведение становится по-настоящему очевидным в случае с контейнерами, которые находятся в режиме ожидания наступления определенного события. Большинство других контейнеров «Мой блок» запускаются и выполняются так быстро, что ты не успеваешь заметить эффект от этого правила. Например, ты не заметишь, что два блока **DisplayNumber** выполняются не одновременно.

Синхронизация двух последовательностей

Ты можешь контролировать момент запуска второй последовательности в зависимости от того, в каком месте находится шина последовательности. Кроме того, ты можешь приостановить выполнение задачи в одной последовательности до завершения задачи в другой, задав переменную в одной последовательности, которая будет считана в другой.

Например, в программе *DoorChime* воспроизведение звукового сигнала и выполнение цикла, обеспечивающего мигание подсветки индикатора состояния модуля, происходит за четыре секунды. Однако этот период времени на самом деле зависит от настройки блоков **Звук** (Sound) в блоке **Chime**. Если ты добавишь дополнительные блоки **Chime** или изменишь продолжительность воспроизведения каждой ноты, то эти две задачи могут не завершиться одновременно.

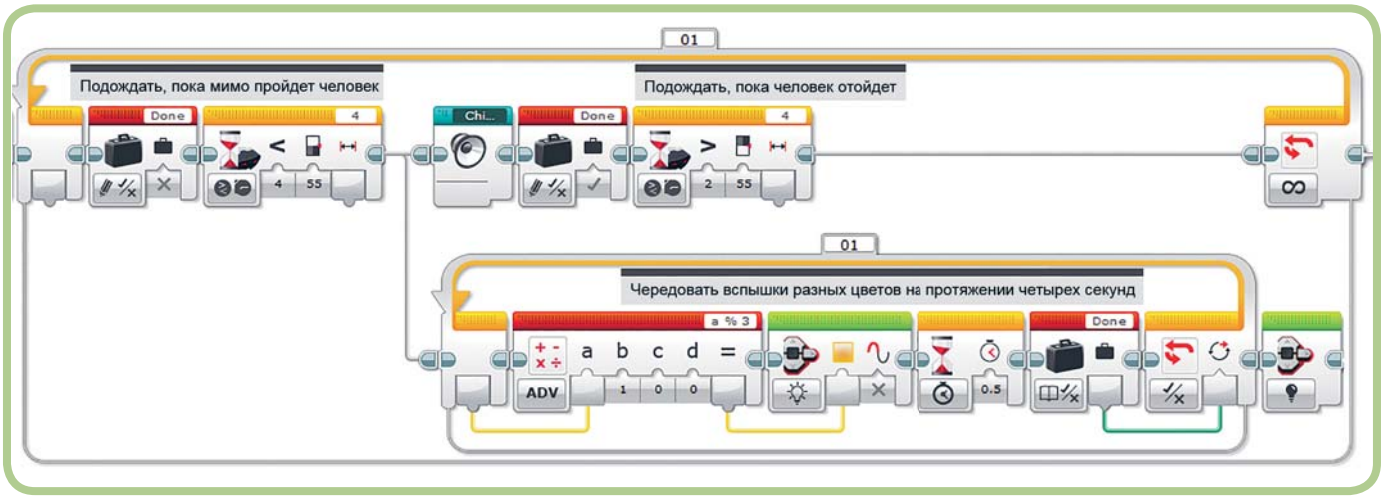


Рис. 18.18. Синхронизация последовательностей в программе DoorChime

На рис. 18.18 показано, как можно решить эту проблему, используя переменную Done для хранения данных логического типа. В начале цикла значением этой переменной является **Ложь** (False). Как только датчик обнаружит проходящего мимо человека, запустятся блок **Chime** и цикл в нижней последовательности. Цикл будет повторяться до тех пор, пока не завершится блок **Chime** и пока с помощью следующего блока **Переменная** (Variable) не изменится значение переменной Done на **Истина** (True). После этого при следующем чтении переменной произойдет выход из цикла. Важно отметить, что выход из цикла происходит не после изменения значения переменной, а только после того, как это значение считано блоком **Переменная** (Variable) и передано в блок **Цикл** (Loop).

Альтернативное решение для этой программы — использование блока **Прерывание цикла** (Loop Interrupt) в верхней последовательности для выхода из цикла в нижней последовательности, как показано на рис. 18.19. При этом необходимо переименовать цикл в нижней последовательности, иначе блок **Прерывание цикла** (Loop Interrupt) осуществит выход из основного цикла.

Предотвращение неполадок

Использование нескольких последовательностей затрагивает практически все аспекты программирования EV3, включая переменные, шины данных, контейнеры «Мой блок» и процесс выполнения программы. Добавление второй последовательности позволяет писать невероятные программы, однако это также повышает риск возникновения неполадок. Далее приведены несколько советов, которые помогут тебе избежать наиболее распространенных проблем:

— **Используй вторую последовательность, только когда это действительно необходимо.** Если возможно, найди такое решение своей задачи, при котором требуется использовать только одну последовательность. Старайся без необходимости не усложнять свою программу.

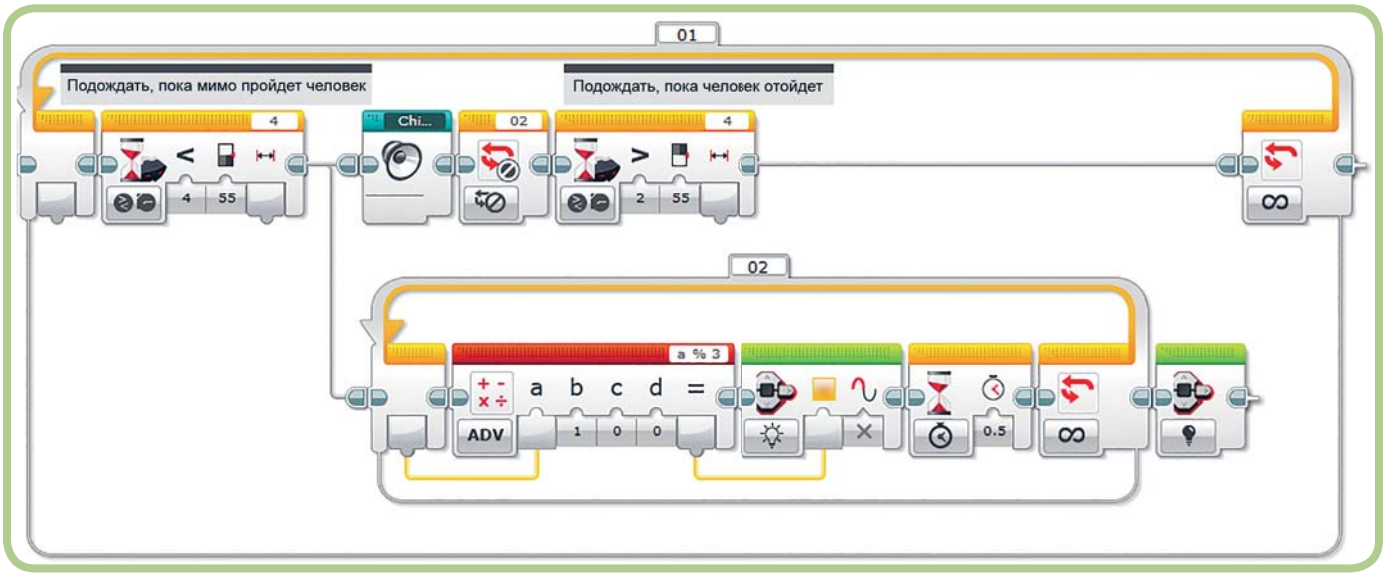


*Рис. 18.19. Синхронизация последовательностей с помощью блока **Прерывание цикла***

— **Редактируй программу медленно**. В программном обеспечении EV3 проблемы чаще возникают при редактировании программ с двумя или более последовательностями, особенно в процессе добавления шин данных.
— **Не пытайся управлять одним и тем же мотором или датчиком более чем из одной последовательности.** При управлении любым ресурсом (мотором, датчиком, таймером и т. д.) из нескольких последовательностей могут возникать проблемы, к тому же этот процесс очень сложно реализовать. Если ты хочешь обеспечить разное поведение робота в зависимости от показаний датчиков, рассмотри возможность использования вложенных блоков **Переключатель** (Switch) вместо нескольких последовательностей.
— **Используй переменные вместо шин данных для передачи информации между последовательностями**. Благодаря этому твоя программа станет более понятной.
— **Соблюдай особую осторожность с шинами данных, которые передают значения в блоки Цикл (Loop) и Переключатель (Switch) или из них.** Перечитай раздел «Правила, касающиеся процесса выполнения программы», приведенной ранее в главе, если не вполне понимаешь, на что следует обращать внимание.

Дальнейшее исследование

Попробуй выполнить следующие действия, чтобы лучше разобраться с использованием параллельных последовательностей:

1. Блок **Прерывание цикла** (Loop Interrupt) завершает любой цикл с соответствующим именем, даже если цикл находится в другой последовательности. Напиши тестовую программу, подтверждающую это. Что произойдет, если этот цикл находится в процессе выполнения блока **Рулевое управление** (Move Steering)?

2. Измени программу *SpiralLineFinder* из гл. 10 так, чтобы использовались две последовательности: одна для обеспечения движения робота TriBot по спиральной траектории, а другая — для обнаружения линии.

3. Добавь в программу *BumperBot* средство для дистанционного управления скоростью с помощью удаленного инфракрасного маяка. Сначала используй переменную для хранения значения параметра **Мощность** (Power) блока **Рулевое управление** (Move Steering), который перемещает робота TriBot вперед. Затем добавь вторую последовательность, при которой можно использовать кнопки на маяке для настройки значения переменной.

4. Добавь таймер обратного отсчета в программу *MemoryGame*, используя новую последовательность в части программы, которая принимает и проверяет ответ пользователя. Пока пользователь дает ответ, на экране модуля должно отображаться оставшееся время, и если время заканчивается, игра завершается. Время, выделенное пользователю на предоставление ответа, должно зависеть от количества элементов в списке; например, по одной секунде для каждого элемента.

Заключение

При использовании нескольких последовательностей программа может одновременно выполнять несколько задач, т. е. является формой многозадачности. Изменения, внесенные в программы *AroundTheBlock* и *DoorChime* в этой главе, позволили продемонстрировать два простых способа улучшения программы путем добавления второй последовательности.

Несмотря на то что многозадачность является очень полезным методом программирования, она и усложняет уже знакомые тебе правила, касающиеся процесса выполнения программы. По этой причине данную технику лучше всего применять к небольшим независимым друг от друга задачам.

19

Программа LineFollower с ПИД-регулятором

Программирование робота на движение вдоль линии — это очень интересная задача, для которой существует множество возможных решений, от простых до чрезвычайно сложных. Мы видели простой трехпозиционный регулятор, хорошо работающий на медленных скоростях при движении вдоль линии с плавными поворотами, а также рассмотрели более сложный пропорциональный регулятор, с помощью которого робот двигается быстрее и может выполнять более резкие повороты. В этой главе мы еще больше усовершенствуем программу, реализовав алгоритм пропорционально-интегрально-дифференцирующего (ПИД) управления. В этой программе корректируется параметр **Рулевое управление** (Steering) на основе расстояния между роботом и линией (как в случае с пропорциональным регулятором). Кроме того, в этой программе вычисляется *дифференциальная* составляющая, которая путем сравнения последних показаний определяет, движется ли робот по кривой или по прямой линии, а также *интегральную* составляющую, которая обнаруживает любое смещение робота, возникающее с течением времени.

Для выполнения программ из этой главы установи датчик цвета на передней панели робота TriBot, как показано на рис. 19.1.

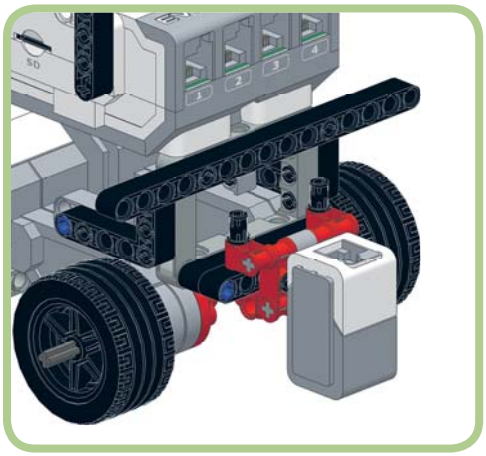


Рис. 19.1. Установка датчика цвета для следования вдоль линии

В начале этой главы рассмотрим вопросы, касающиеся следования по линии, а также алгоритм пропорционального регулирования, что позволит тебе лучше понять, как (и почему) работает программа *LineFollower* из гл. 13. Затем мы внесем в существующую программу небольшие улучшения, чтобы в ней использовались файлы и программа настройки для сбора и сохранения предельных значений датчика, а также переменные, облегчающие настройку параметров программы. После этого ты узнаешь, как использовать более совершенную стратегию ПИД-регулирования для следования вдоль линии. К концу главы у тебя будет чрезвычайно надежная программа для передвижения вдоль линии, а также понимание принципов, лежащих в основе сложного алгоритма ПИД-регулирования, который применяется для решения множества задач в робототехнике.

ПИД-регулятор

Пропорционально-интегрально-дифференциальный (ПИД) регулятор представляет собой очень распространенный и полезный метод управления различными типами машин, в том числе роботами. Концепции, лежащие в основе ПИД-регулятора, были разработаны около 100 лет назад и используются для управления всеми видами механизмов во всех сферах от судовой навигации и принтеров до музыкальных инструментов.

Подобно пропорциональному регулятору, представленному в гл. 13, в ПИД-регуляторе используется показание датчика, называемое входной переменной, для настройки контрольной переменной. В программе для следования вдоль линии, которую мы разрабатываем, входной переменной является показание датчика цвета, а контрольной переменной — параметр **Мощность** (Power) блока **Рулевое управление** (Move Steering). С помощью регулятора сравниваются значение входной переменной и целевое значение для вычисления значения ошибки. Затем значение ошибки используется для определения того, как следует изменить контрольную переменную.

Пропорциональный регулятор делает значение параметра **Рулевое управление** (Steering) пропорциональным значению ошибки (разнице между показанием датчика цвета и целевым значением). Когда робот TriBot находится близко к краю линии, значение параметра **Рулевое управление** (Steering) невелико, а когда он находится дальше от края линии, этот параметр имеет большее значение. Пропорциональный регулятор оказывается более эффективным по сравнению с трехпозиционным регулятором из гл. 9, поскольку он использует больше информации о том, насколько хорошо робот следует вдоль линии. В то время как трехпозиционный регулятор использует показание датчика цвета для выбора одного из трех возможных значений параметра **Рулевое управление** (Steering) (поворот влево, поворот вправо или движение прямо). Пропорциональный регулятор использует величину значения ошибки для выбора из более широкого диапазона возможных значений параметра **Рулевое управление** (Steering), что позволяет роботу TriBot гораздо лучше следовать вдоль линии. Несмотря на то что пропорциональный регулятор значительно превосходит трехпозиционный, у него есть некоторые существенные ограничения.

Пропорциональный регулятор учитывает расстояние между датчиком и краем линии в данный момент, но он не замечает, как линия изменяется со временем, а также то, что робот постепенно смещается в том или ином направлении. Поскольку он реагирует только на значение ошибки в конкретный момент времени, не учитывая прошлые значения ошибки, он может не очень эффективно реагировать на внезапные изменения линии и не принимать во внимание источники постоянных ошибок. С помощью ПИД-регулятора можно решить эти проблемы за счет добавления двух дополнительных составляющих в выражение, используемое для вычисления значения параметра **Рулевое управление** (Steering): интегральной и дифференциальной. Далее ты увидишь, что эти две составляющие используют предыдущие значения ошибки, благодаря чему робот двигается быстрее, выполняет резкие повороты и более плавно следует вдоль прямой линии.

Одна из особенностей ПИД-регулятора, делающая его популярным средством управления автоматическими системами, заключается в том, что его можно легко настроить для решения конкретной задачи. В этой главе мы будем использовать ПИД-регулятор в программе, заставляющей робота двигаться вдоль линии, однако фрагмент программы с регулятором может применяться для решения множества задач. Например, его можно использовать с инфракрасным или ультразвуковым датчиком, чтобы робот следовал за тобой, держась на определенном расстоянии, или с гироскопическим датчиком, чтобы робот балансировал на двух колесах. Когда ты разберешься с принципом работы ПИД-регулятора и научишься настраивать его для решения конкретной задачи, ты сможешь использовать его код снова и снова.

Пропорциональное регулирование

Прежде чем добавить ПИД-регулятор в нашу программу для следования вдоль линии, давай посмотрим, как изменяются показания датчика, когда робот приближается к линии и пересекает ее, и разберемся с тем, как наша существующая программа с пропорциональным регулированием реагирует на эти изменения.

В программе *LightTest*, показанной на рис. 19.2, используется способ регистрации данных из гл. 17 для сбора значений яркости отраженного света во время пересечения линии. Это позволяет нам легко записывать показания датчиков на разных расстояниях от линии, и позднее мы применим аналогичную программу для считывания и записи минимальных и максимальных показаний датчиков, которые будут использоваться в программе для следования вдоль линии.

Полученные данные сохраняются в файле LightTestData. Блок **Цикл** (Loop) настроен на повторение до тех пор, пока

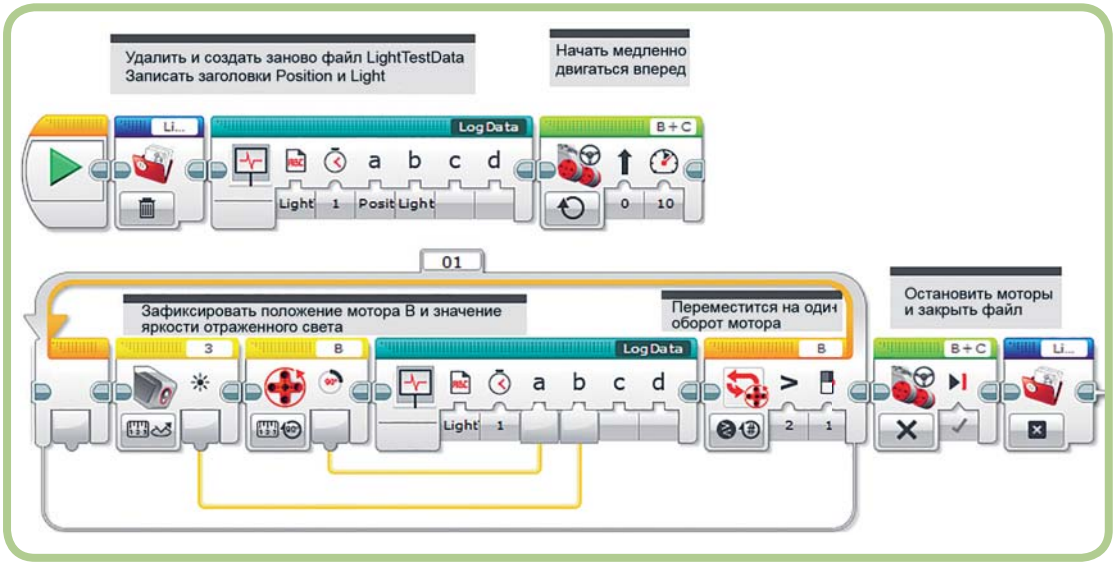


Рис. 19.2. Программа LightTest

показание датчика вращения мотора B не превысит одного оборота, что позволит роботу переместиться достаточно далеко для сбора всех необходимых данных.

Направь робота TriBot в сторону линии так, чтобы датчик цвета располагался на расстоянии около 5 см от нее, как показано на рис. 19.3. После запуска программы робот должен медленно двигаться вперед и останавливаться, как только датчик цвета полностью пересечет линию. Линия должна оказаться примерно на полпути между начальным и конечным положениями датчика цвета.

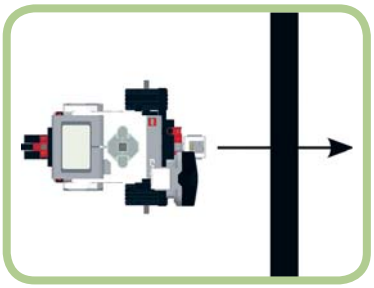


Рис. 19.3. Начальное положение робота при выполнении программы LightTest

Необработанные данные

После выполнения программы и загрузки файла LightTestData на компьютер на основе полученных данных с помощью программы электронных таблиц был создан график (рис. 19.4). Для наглядности на графике обозначено положение линии. При запуске программы показание датчика составляет около 62 и остается у отметки 60, пока робот приближается к линии. Вблизи линии показания датчика резко уменьшаются до тех пор, пока датчик полностью не окажется над линией, в этот момент его показание равно примерно 5. Показание остается возле этой отметки до тех пор, пока датчик не начнет приближаться к другому краю линии. Затем показания снова резко увеличиваются, пока датчик не окажется за краем линии, после чего показания снова остаются около отметки 60. В самом конце графика значение повышается до 65.

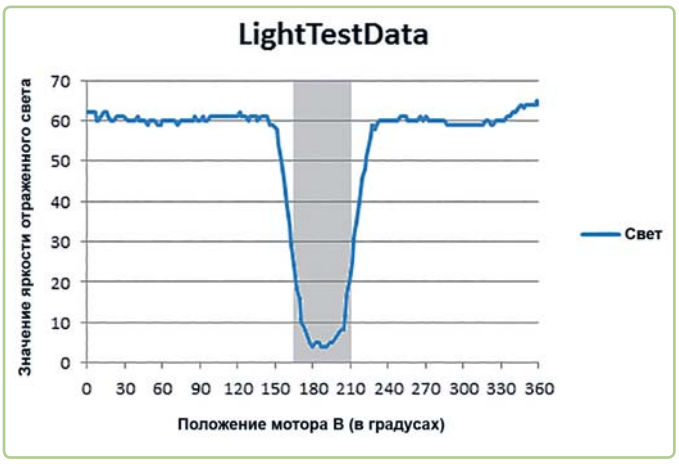


Рис. 19.4. Изменение значения яркости отраженного света при пересечении линии

ПРИМЕЧАНИЕ **Показанные здесь значения отличаются от приведенных в предыдущих главах, поскольку здесь использована другая тестовая линия с более темным фоном. Помимо незначительной разницы в цвете фона, тестовая линия аналогична описанной в гл. 6 и представляет собой простой овал, обозначенный черной изолентой на белом листе картона.**

Из этого графика можно сделать вывод о том, что показание датчика будет держаться около отметки 60, когда робот вдали от линии, и около отметки 5, когда датчик находится непосредственно над линией. Когда датчик расположен у края линии, его показание будет находиться между 60 и 5, и мы можем использовать это значение для определения расстояния между датчиком и краем линии. Прямая линия, соответствующая изменению показания с 60 до 5, свидетельствует о том, что это показание пропорционально расстоянию до края линии, поэтому мы можем использовать его в качестве надежной основы для реализации алгоритма пропорционального регулирования в программе для следования вдоль линии.

Обрати внимание на то, что график симметричен, т. е. его левая и правая половины являются зеркальными отражениями друг друга. Именно из-за этой симметрии мы не пытаемся заставить робота следовать вдоль середины линии. Мы можем определить, находится ли датчик в середине линии (или рядом с ней), этому положению соответствует значение, очень близкое к 5. Однако, если показание далеко от отметки 5, мы не можем решить, в какую сторону следует повернуть роботу, чтобы приблизиться к линии. Например, если показание датчика равно 20, мы не можем сказать, находится ли датчик за левым или за правым краем линии.

Хорошие и плохие зоны

Теперь давай более подробно рассмотрим показания датчика, когда он находится вблизи линии, и подумаем о том, как будет вести себя программа для следования вдоль линии. В гл. 13 в качестве целевого мы выбрали среднее значение между показаниями датчика, соответствующими его положению вне линии и над ее центром. Здесь мы можем сделать то же самое, используя максимальное и минимальное значения из файла LightTestData, равные 65 и 5 соответственно. Значит, среднее значение — 35. На рис. 19.5 показано измененное представление данных с зеленой горизонтальной линией, соединяющей точки, в которых показание датчика равно 35. Для упрощения следующего обсуждения ось x была сдвинута и масштабирована, чтобы отобразить расстояние в сантиметрах от точки, в которой показание датчика равно 35.

График был разделен на четыре зоны на основании положения датчика относительно точки, соответствующей его среднему показанию. Эффективность программы следования вдоль линии зависит от того, в какой зоне находится датчик.

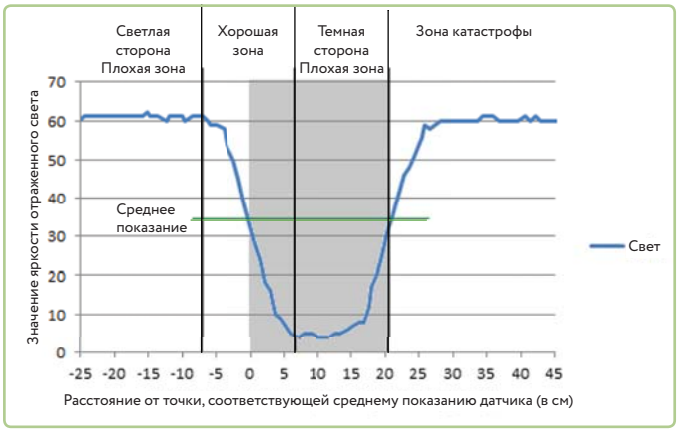


Рис. 19.5. Показания датчика у края линии

Хорошая зона

После задания в качестве целевого значения среднего показания датчика программа будет пытаться удерживать датчик в положении, соответствующем 0 на графике, помещая робота в середину хорошей зоны, где показания датчика будут примерно пропорциональны расстоянию до линии. Пока робот остается на расстоянии около 7 см от края линии, показание датчика будет давать нам хорошее представление о том, насколько далеко от целевой точки находится робот, а пропорциональный регулятор сможет использовать это показание для направления робота к краю линии.

Светлая сторона, Плохая зона

В левой части графика находится Светлая сторона, Плохая зона, которая начинается на расстоянии более 7 см от положения точки, соответствующей среднему показанию. Когда робот находится в этой зоне, показание датчика означает, что робот отъехал слишком далеко влево, но мы не знаем, насколько далеко. На расстоянии 10 или 20 см от края линии датчик показывает одно и то же значение.

Робот может оказаться в этой области тремя способами, как показано на рис. 19.6. На этом изображении красный кружок соответствует местоположению датчика, а голубая линия описывает путь робота по мере его движения вперед.

— Когда линия поворачивает влево (рис. 19.6, а), датчик начинает пересекать ее. Если программа компенсирует это, используя слишком большое значение параметра **Рулевое управление** (Steering), робот может переместиться слишком далеко влево.
— Когда линия поворачивает вправо (рис. 19.6, б), показания датчика увеличиваются. Если программа среагирует слишком поздно, используя недостаточно большое значение параметра **Рулевое управление** (Steering), робот может отклониться слишком далеко влево.
— Наконец, если при движении робота вдоль прямой линии используется слишком большой коэффициент усиления, робот может слишком сильно колебаться из стороны в сторону, т. е. *осциллировать* (рис. 19.6, в); в этом случае он также может оказаться слишком далеко от линии.

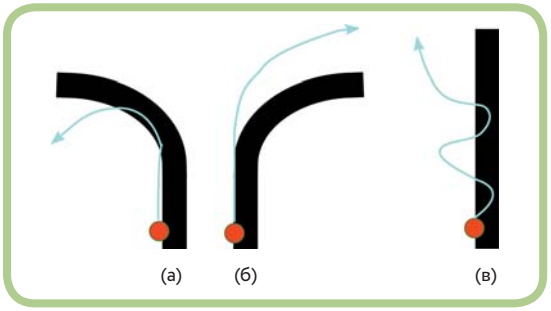


Рис. 19.6. Заезд робота на Светлую сторону в Плохую зону

Пропорциональный регулятор, который отлично работает в Хорошей зоне, скорее всего, будет использовать слишком большие значения параметра **Рулевое управление** (Steering) при нахождении робота в Плохой зоне, часто заставляя его двигаться по кругу. Если робот снова приблизится к линии (в Хорошей зоне), то сможет продолжить движение вдоль нее. Однако, если он отклонится слишком далеко, например, если линия сделает резкий U-образный поворот вправо, то робот будет бесконечно вращаться по кругу.

Темная сторона, Плохая зона

Справа от Хорошей зоны находится Темная сторона, Плохая зона, соответствующая положению робота, когда он слишком далеко заезжает за край линии. Эта область состоит из двух частей. В пределах от 7 до 12 см от средней точки показание датчика остается практически постоянным. При этом возникают те же проблемы, что и при нахождении робота на Светлой стороне в Плохой зоне: мы не можем точно сказать, насколько далеко от края линии находится робот, а используемое программой значение параметра **Рулевое управление** (Steering), скорее всего, заставит робота вращаться на месте.

Ситуация еще хуже, когда расстояние превышает 12 см, поскольку с этого момента показание датчика начинает увеличиваться. Это заставляет программу вести себя так, как будто робот приближается к краю линии, хотя на самом деле он отдаляется от нее.

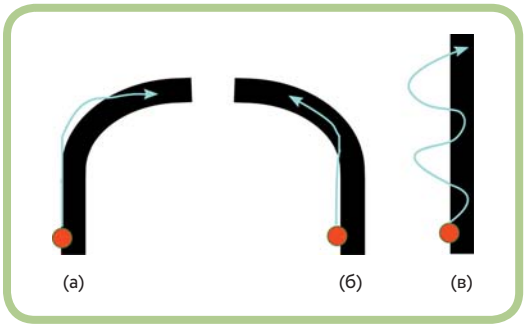


Рис. 19.7. Заезд робота на Темную сторону в Плохую зону

На рис. 19.7 показано, как робот может оказаться в этой области. Программа может использовать слишком большое значение параметра **Рулевое управление** (Steering), когда

линия поворачивает вправо (рис. 19.7, а), или недостаточно большое значение, когда линия поворачивает влево (рис. 19.7, б). Кроме того, робот может оказаться в этой области из-за сильных колебаний при движении вдоль прямой линии (рис. 19.7, в).

Когда робот находится в этой области, программа пытается вернуть его обратно к краю линии, хотя она может сделать это недостаточно быстро. Если программа сумеет повернуть робота влево, в Хорошую зону, то он сможет продолжить движение вдоль линии. Если робот отклонится слишком далеко вправо, он окажется в Зоне катастрофы.

Зона катастрофы

Зона катастрофы начинается у другого края линии в том месте, где показание датчика превышает среднее значение. По достижении этой точки программа заставляет робота двигаться в неправильном направлении — он смещается вправо, ожидая уменьшения показания датчика. После этого робот уже не сможет взять правильный курс: он будет кружиться или начнет двигаться вдоль другого края линии в противоположном направлении (что довольно интересно наблюдать!)

Эти четыре зоны и их влияние являются специфичными для программы следования вдоль линии. Любая программа, использующая ПИД-регулятор, будет предусматривать набор показаний, соответствующий Хорошей зоне, где показания изменяются пропорционально контролируемому значению. Однако то, что происходит за пределами этой зоны, зависит от конкретной программы (например, благодаря одной программе робот следует за тобой, можно легко заставить остановиться, когда расстояние равно 0, или двигаться на полной скорости в случае отставания робота). Таким образом, для этой программы будут существовать только Хорошая и Плохая зона (когда робот далеко, но не ясно, насколько). Однако для балансирующего робота будет существовать одна Хорошая зона и две Зоны катастрофы — по одной с каждой стороны, поскольку он упадет, если наклонится слишком далеко вперед или назад.

Выбор целевого значения

Теперь, когда мы разобрались с принципом работы регулятора, как это повлияет на наш выбор целевого значения? Использование среднего значения между максимальным и минимальным показаниями датчика разумно, поскольку при этом программа будет стараться удерживать робота в середине Хорошей зоны, где она работает эффективно, однако можно пойти еще дальше.

Мы можем сделать программу более надежной (т. е. реже дающей сбой), немного увеличив целевое значение. Например, если мы используем 40 вместо 35, робот будет смещаться влево от центра Хорошей зоны (примерно на 2 см от средней точки на рис. 19.5). После такого изменения робот с большей вероятностью уедет слишком далеко влево, но с меньшей вероятностью переместится слишком далеко вправо. Это улучшит программу, поскольку у робота больше шансов вернуться на прежний курс, если он находится на Светлой стороне в Плохой зоне, чем тогда, когда он оказывается на Темной стороне в Плохой зоне или в Зоне катастрофы. Слишком большое отклонение влево — это плохо, однако слишком большое отклонение вправо еще хуже, поэтому имеет смысл запрограммировать некоторое смещение влево.

Наилучшее целевое значение зависит от твоей тестовой траектории. Если робот должен совершать повороты влево и вправо, использование целевого значения, обеспечивающего некоторое смещение робота влево от центра Хорошей зоны, дает хороший результат. Однако если твоя траектория представляет собой овал и робот должен поворачивать лишь в одном направлении, ты можешь скорректировать целевое значение для обеспечения большей надежности при выполнении поворота в этом направлении.

Если робот движется вдоль внутренней стороны овала, как в примере, то он всегда поворачивает влево, а сбой в программе возникает из-за того, что робот пересекает линию. В этом случае, используя целевое значение, из-за которого робот смещается немного влево, можно получить лучший результат.

Однако, если робот следует вдоль внешней стороны овала, то наиболее распространенный сбой происходит при недостаточно быстром повороте и слишком сильном отклонении от линии. В этом случае установка целевого значения в центре Хорошей зоны или чуть правее поможет лучше всего.

Определение минимального и максимального показаний датчика

Поскольку для работы программы с разными траекториями, датчиками, конструкциями робота и уровнями освещенности могут потребоваться изменения целевого значения, создадим калибровочную программу *LineFollowerCal*, которая будет определять минимальное и максимальное показания датчика цвета и сохранять эти два значения в файл. Затем мы изменим программу *LineFollower* так, чтобы она считывала значения из файла и использовала их для вычисления целевого значения.

Как и программа *LightTest*, программа *LineFollowerCal* заставляет робота TriBot пересечь линию и наблюдает за количеством отраженного света, полученным от датчика цвета. Вместо регистрации всех показаний эта программа отслеживает только самые высокие и самые низкие значения. После остановки робота программа отображает эти два значения и дает тебе возможность принять или отклонить их. Это позволяет тебе выяснить предельные значения, используемые в программе *LineFollower*, а также избежать любых проблем, возникших в процессе калибровки (например, если датчик был подключен не к тому порту или запуск робота произошел слишком близко или слишком далеко от линии).

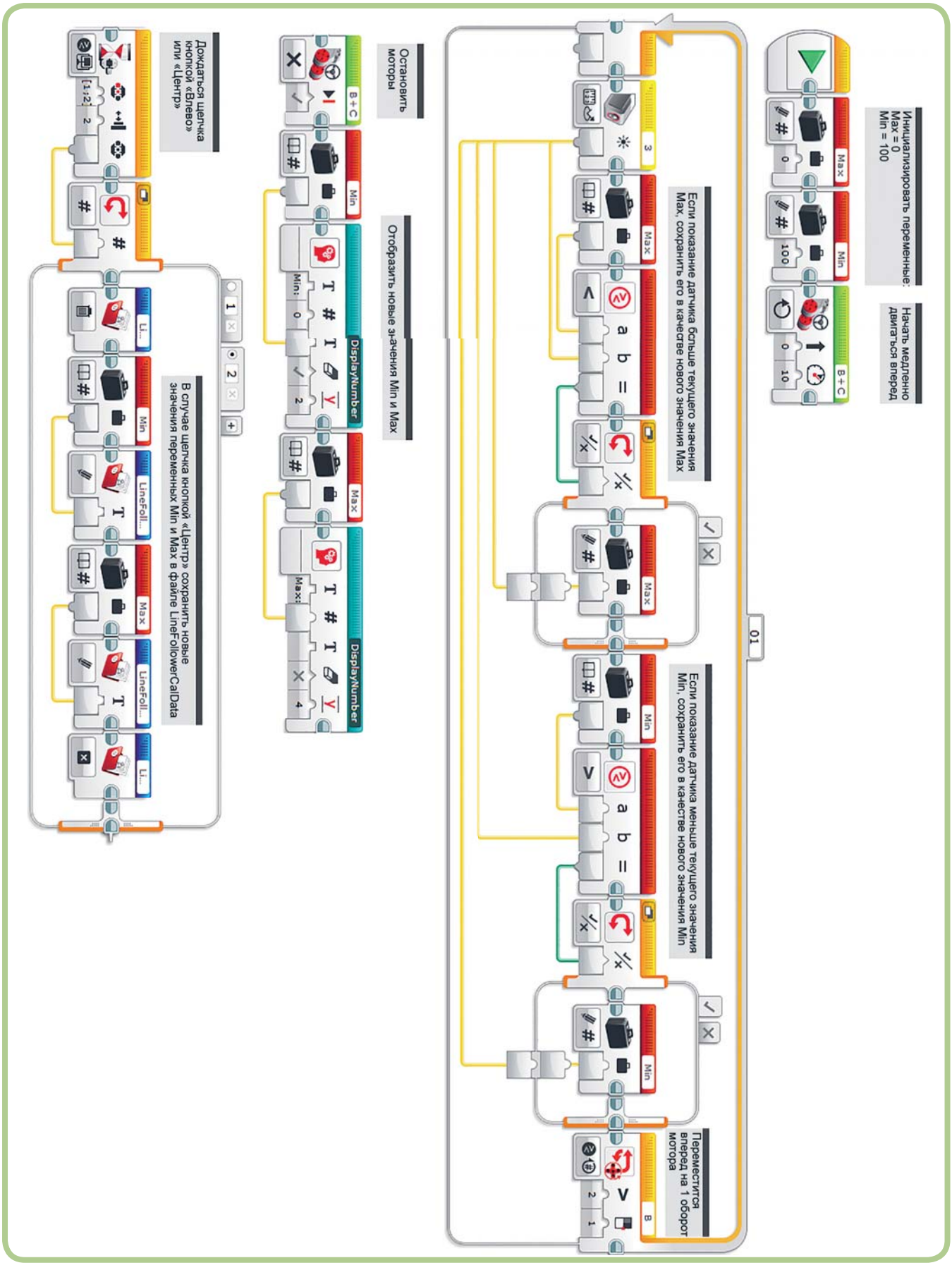


Рис. 19.8. Программа LineFollowerCal

На рис. 19.8 показана длинная, но не очень сложная программа. Она начинается с присвоения переменным Min и Max значений 100 и 0 соответственно. Затем колеса робота начинают медленно вращаться вперед, программа входит в цикл и начинает считывать показания датчика цвета, используя режим **Измерение** (Measure) ⇒ **Яркость отраженного света** (Reflected Light Intensity). Если новое показание больше текущего значения Max или меньше текущего значения Min, соответствующая переменная обновляется. Обрати внимание на то, что при первом выполнении цикла показание (почти всегда) будет меньше 100 и больше 0, поэтому будут обновлены обе переменные.

После того как робот TriBot переместится вперед на один оборот мотора, произойдет выход из цикла, и моторы остановятся. Два блока **DisplayNumber** отобразят на экране модуля новые значения переменных Min и Max, а затем программа перейдет в режим ожидания нажатия кнопки. Если отображенные значения покажутся тебе разумными, нажми кнопку «Центр», чтобы удалить старый файл LightFollowerCal и записать два значения в новый файл. Если показания покажутся неправильными, например, если робот был расположен недостаточно близко к линии, нажми кнопку «Влево», чтобы завершить программу, не сохраняя новые значения. (В случае блока **Переключатель** (Switch), соответствующем нажатию кнопки «Влево», нет блоков.)

Запусти программу. Если все работает правильно, на экране модуля отобразятся минимальное и максимальное показания датчика цвета. Нажми кнопку «Центр», чтобы сохранить значения в файле. Ты можешь использовать инструмент **Обозреватель памяти** (Memory Browser), чтобы проверить существование файла, и запустить программу *FileReader* из гл. 16, для проверки, что два отображенных значения действительно были записаны в файл.

Нормализация показаний датчика и целевых значений

Программа *LineFollowerCal* определяет минимальное и максимальное показания датчика цвета, чтобы можно было вычислить целевое значение. Благодаря этому можно адаптировать программу *LineFollower* к различным тестовым линиям, однако ее можно сделать еще более гибкой.

Вот проблема: допустим, запускаем первую программу *LineFollowerCal* на одной тестовой линии, получаются значения Min = 5 и Max = 65. Ты запускаешь вторую программу на своей линии и получаешь значения Min = 15 и Max = 55. В обоих случаях средним значением будет 35, однако программа должна реагировать на каждую линию по-разному в связи с различием в диапазоне значений. Показание датчика, равное 55, полученное при движении вдоль первой линии первой программы, говорит о том, что робот находится слева от линии, но все еще рядом с ней, при этом то же показание, полученное при движении вдоль твоей линии, означает, что робот совершенно отдалился от линии. Значение параметра **Рулевое управление** (Steering), используемое программой для возвращения робота на нужный курс, в обоих случаях должно быть разным.

Для решения этой проблемы применим процесс, известный как *нормализация* данных, т. е. преобразование данных с целью использования одного и того же диапазона. Вместо того, чтобы работать с необработанными показаниями датчика, которые будут находиться в диапазоне от 5 до 65 в первом случае и от 15 до 55 — во втором. Мы можем преобразовать каждое показание в процент от ожидаемого диапазона значений (другими словами, так преобразовать каждое показание датчика, чтобы оно принадлежало диапазону от 0 до 100). Запишем формулу для вычисления нормализованного показания датчика на основании его необработанного показания и значений Min и Max:

$$\text{Нормализованное показание} = \frac{100 \times (\text{Показание датчика} - \text{Min})}{(\text{Max} - \text{Min})}$$

На рис. 19.9 показано, как с помощью блоков **Датчик цвета** (Color Sensor) и **Математика** (Math) можно вычислить нормализованное показание датчика. Результат выполнения блока **Математика** (Math) будет принадлежать диапазону от 0 до 100 и сообщит количество отраженного света относительно диапазона ожидаемых значений. Показание датчика, равное 35, будет нормализовано до 50 при использовании диапазонов значений для любой из тестовых линий. Однако показание датчика, равное 55, в случае первой тестовой линии будет нормализовано до 83, а в случае второй тестовой линии — до 100. Это позволит программе *LineFollower* реагировать в соответствии с диапазоном ожидаемых значений, и при движении вдоль второй тестовой линии совершать более резкие повороты, чем при движении вдоль первой тестовой линии.

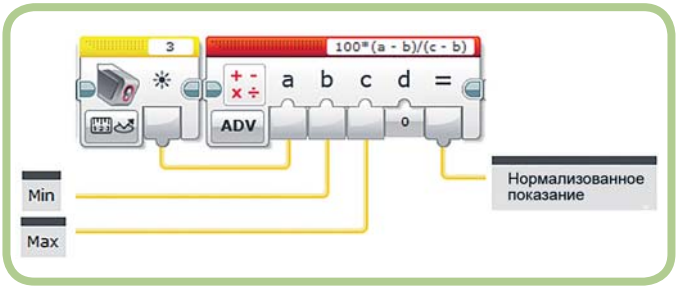


Рис. 19.9. Нормализация показания датчика цвета

При создании программы *LineFollower* мы используем блоки, изображенные на рис. 19.9, для нормализации показания датчика, а это означает, что целевое значение мы также должны указать в диапазоне от 0 до 100. Целевое значение 50 соответствует среднему значению Min и Max. Как говорилось в разделе «Выбор целевого значения» ранее в этой главе, можно использовать немного большее значение. Необработанное показание датчика, которое составляет 40

при использовании моих значений Min = 5 и Max = 65, соответствует нормализованному значению около 60, поэтому в программе *LineFollower* в качестве целевого значения я буду использовать 60.

Доработка программы LineFollower с пропорциональным регулированием

Теперь, после завершения подготовительного этапа, мы можем добавить код в программу *LineFollower* с пропорциональным регулированием (за основу взята версия из гл. 13), для того, чтобы она использовала значения из файла LineFollowerCalData и нормализовала показания датчика и целевые значения.

Были внесены еще некоторые небольшие изменения для того, чтобы сделать программу более понятной. Для облегчения процесса настройки регулятора в начале программы заданы четыре переменные.

- Переменная Power используется для регулирования скорости робота.
- В переменной Target хранится нормализованное целевое значение.
- *Kp* — это **пропорциональный коэффициент усиления**.
- Для переменной Direction задано значение 1 или –1 в зависимости от того, требуется ли изменить знак расчетного значения параметра **Рулевое управление** (Steering). Для следования вдоль левой стороны линии это значение должно быть равно –1. Сохранение этого значения в переменной облегчает процесс повторного использования той части кода, которая отвечает за регулирование.

В программе, описанной в гл. 13, переменная, в которой хранился **пропорциональный коэффициент**, называлась *Gain*. Здесь имя программы изменено на *Kp*, поскольку в конечном итоге мы получим три коэффициента усиления, по одному для каждой составляющей ПИД-регулятора (пропорциональной, интегральной и дифференциальной). В формулах буквой *K* обычно обозначается постоянная величина, как в данном случае, а буква *p* используется потому, что коэффициент усиления является пропорциональным.

РЕЖИМЫ КАЛИБРОВКИ БЛОКА «ДАТЧИК ЦВЕТА»

Датчик цвета предусматривает встроенную поддержку нормализации показаний датчика. Для этого используются режимы **Калибровка** (Calibrate) блока **Датчик цвета** (Color Sensor) (рис. 19.10). Установи минимальное и максимальное ожидаемое значение, используя режимы **Калибровка** (Calibrate) ⇒ **Яркость отраженного света** (Reflected Light Intensity) ⇒ **Минимум** (Minimum) и **Калибровка** (Calibrate) ⇒ **Яркость отраженного света** (Reflected Light Intensity) ⇒ **Максимум** (Maximum) соответственно, и показания датчика будут нормализованы в соответствии с заданным диапазоном. Режим **Калибровка** (Calibrate) ⇒ **Яркость отраженного света** (Reflected Light Intensity) ⇒ **Сброс** (Reset) вернет датчик цвета в исходное состояние.

При использовании режимов **Калибровка** (Calibrate) для установки минимального и максимального показаний датчика модуль EV3 запоминает эти значения и продолжает использовать их в других программах. Они не исчезают из памяти модуля даже после его отключения. Для удаления этих значений нужно запустить блок **Датчик цвета** (Color Sensor) в режиме **Калибровка** (Calibrate) ⇒ **Яркость отраженного света** (Reflected Light Intensity) ⇒ **Сброс** (Reset). Поэтому, если после применения этой функции в любой из программ ты заметишь, что датчик цвета перестал работать правильно, попробуй использовать режим **Сброс** (Reset) и посмотри, решит ли это твою проблему.

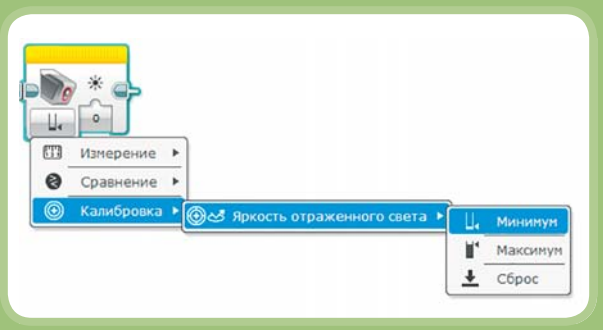


*Рис. 19.10. Режимы **Калибровка** (Calibrate) блока **Датчик цвета** (Color Sensor)*

Проведем нормализацию в программе *LineFollower* вместо использования режимов **Калибровка** (Calibrate), поскольку лучше, чтобы датчик всегда работал одинаково, не подвергаясь влиянию результатов калибровки, выполненной в других программах, даже если это требует добавить в программу дополнительный блок **Математика** (Math). Кроме того, тебе придется явно выполнить этот шаг в любой программе, использующей другой датчик (поскольку другие датчики не предусматривают режимов калибровки).

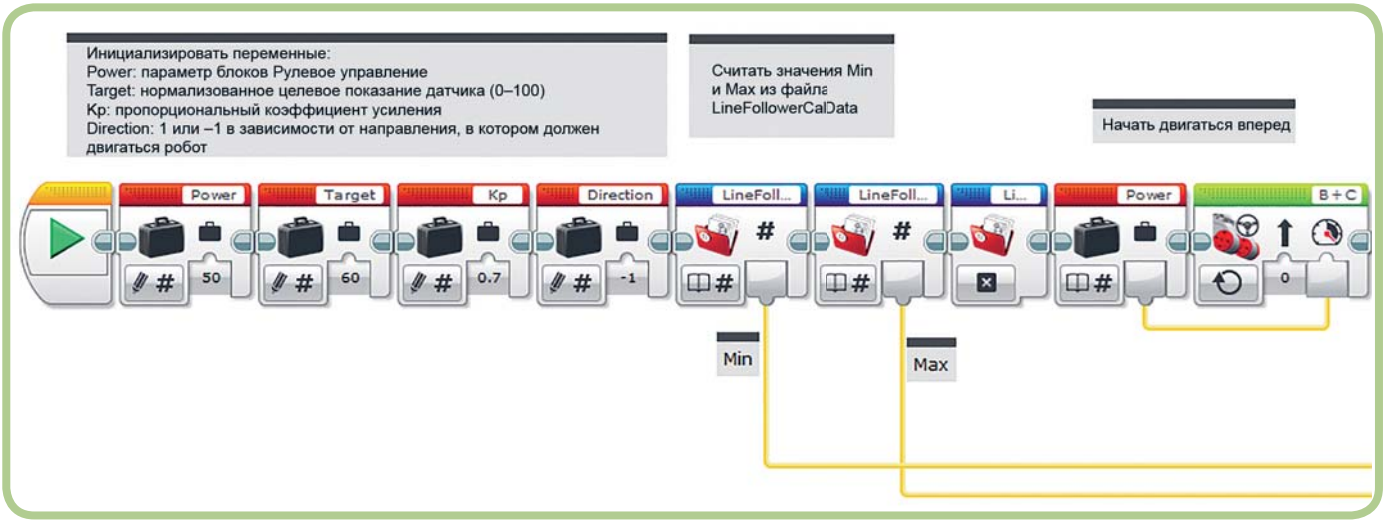


Рис. 19.11. Программа LineFollower с пропорциональным регулятором, часть 1

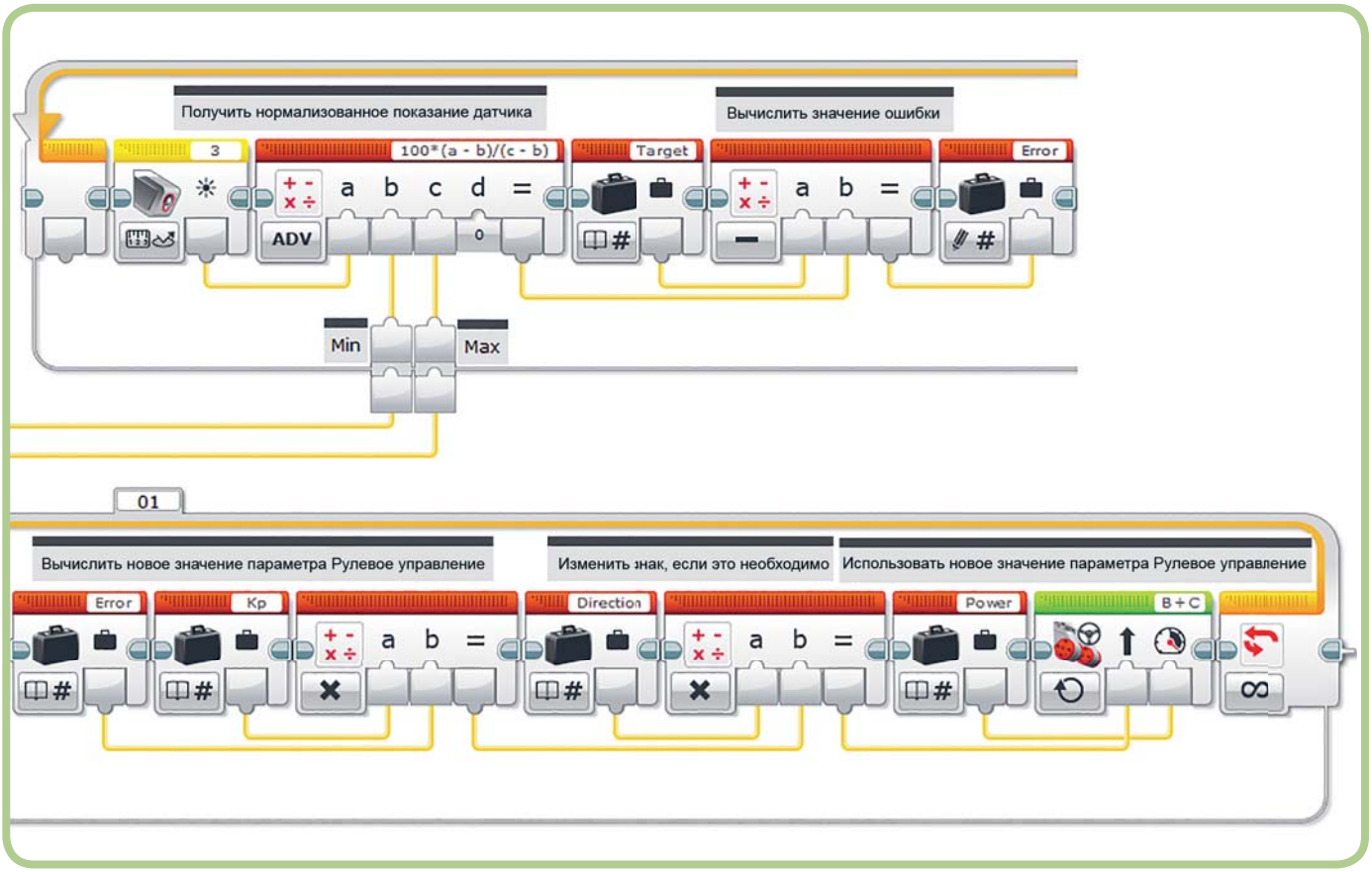


Рис. 19.12. Программа LineFollower с пропорциональным регулятором, часть 2

Первая часть программы, которая инициализирует переменные, считывает файл калибровки и заставляет робота начать движение вперед, показана на рис. 19.11.

На рис. 19.12 изображен основной цикл программы, в котором происходит считывание показания датчика и корректировка значения параметра **Рулевое управление** (Steering). Обрати внимание на то, что значение ошибки хранится в переменной, а не передается непосредственно в блок **Математика** (Math) — это позволит избежать сложностей по мере увеличения программы. По этой же причине значение переменной Direction применяется вместе с отдельным блоком **Математика** (Math).

ПРИМЕЧАНИЕ **Значение ошибки сохраняется в переменной, а затем извлекается исключительно для ясности; данное значение можно передать непосредственно в блок Математика (Math), в котором оно используется. В следующих разделах по мере увеличения размера программы это станет более важным.**

Протестируй программу и скорректируй ее, чтобы найти подходящий коэффициент усиления и значение параметра **Мощность** (Power), как это было сделано в гл. 13. В результате тестов были получены приемлемые результаты при использовании коэффициента усиления 0,7 и значения параметра **Мощность** (Power), равного 40. Новая программа, объединенная с программой *LineFollowerCal*, легче адаптируется к различным линиям и уровням освещенности. Тем не менее она должна вести себя так же, как и предыдущая версия, поскольку мы не изменили алгоритм управления.

Реализация ПИД-регулятора

Чтобы сделать программу еще более надежной, мы заменим пропорциональный регулятор полнофункциональным ПИД-регулятором, добавив еще две составляющие в формулу, используемую для вычисления нового значения параметра **Рулевое управление** (Steering). Сначала мы добавим дифференциальную составляющую, которая позволит программе *LineFollower* реагировать на внезапные изменения в направлении линии. После этого включим интегральную составляющую, которая будет учитывать любые постоянно возникающие ошибки. В результате у тебя будет надежный регулятор, который отлично подходит для следования вдоль линии и легко адаптируется под другие программы, использующие датчик для управления моторами.

Добавление дифференциальной составляющей ПИД-регулятора

Пропорциональный регулятор работает очень хорошо, если линия прямая. Однако он может не справиться с поворотами, если робот движется быстро или поворот является слишком крутым. Пропорциональный коэффициент усиления, используемый регулятором, определяет степень изменения значения параметра **Рулевое управление** (Steering), исходя из значения ошибки. Проблема в том, что коэффициент усиления, хорошо работающий при движении вдоль прямой линии, оказывается слишком маленьким для совершения резких поворотов. В то же время другой довольно велик для совершения поворотов и вызывает сильные колебания при движении вдоль прямой линий.

Концептуально мы нуждаемся в том, чтобы при движении вдоль прямой линии программа вносила незначительные изменения, а на поворотах — большие. Для этого нам нужно добавить в наше выражение еще одну *дифференциальную* составляющую для вычисления нового значения параметра **Рулевое управление** (Steering). Дифференциальная составляющая определяет степень изменения значения ошибки. При движении вдоль прямой линии значение дифференциальной составляющей будет небольшим, поскольку показания датчика меняются не сильно. Когда робот приближается к повороту, это показание внезапно начинает меняться по мере отклонения робота от линии или ее пересечения.

Простым, но эффективным способом аппроксимации дифференциального значения ошибки (Derivative) заключается в вычитании предыдущего значения ошибки (LastError) из ее текущего значения (Error). Чтобы вычислить новое значение параметра **Рулевое управление** (Steering), мы прибавим к пропорциональной составляющей (которая равна произведению коэффициента усиления *Kp* на значение ошибки) дифференциальную составляющую, умноженную на другой, дифференциальный, коэффициент усиления, обычно обозначаемый буквами *Kd*. Теперь для вычисления значения параметра **Рулевое управление** (Steering) будем использовать следующую формулу:

$$Derivative = Error - LastError$$
$$Steering = Kp \times Error + Kd \times Derivative$$

При движении вдоль прямой линии между последовательными значениями ошибки не будет большой разницы, а дифференциальная составляющая будет невелика или равна 0, поэтому ее значение не повлияет на движение робота. Когда робот доедет до поворота, разница между последовательными значениями ошибки начнет расти и дифференциальная составляющая приведет к существенному изменению значения параметра **Рулевое управление** (Steering). Таким образом, по достижении поворота робот получит больший толчок в правильном направлении, сила которого зависит от дифференциального коэффициента усиления.

Для того чтобы включить в программу дифференциальную составляющую, понадобятся две новые переменные: *Kd* содержит дифференциальный коэффициент усиления, а в LastError будет храниться предыдущее значение ошибки. Коэффициент *Kd* необходимо задать в начале программы, используя в качестве его значения результат проведенных экспериментов, а переменная LastError должна быть инициализирована нулем.

На рис. 19.13 показан код, который мы будем использовать для вычисления дифференциальной составляющей

Рис. 19.13. Вычисление дифференциальной составляющей

Рис. 19.14. Использование дифференциальной составляющей при вычислении значения параметра Рулевое управление

и сохранения предыдущего значения ошибки. На рис. 19.14 показан результат включения дифференциальной составляющей в выражение для вычисления нового значения параметра **Рулевое управление** (Steering). Мы будем использовать эти фрагменты кода в финальной программе, однако перед этим необходимо описать третью *интегральную* составляющую ПИД-регулятора.

Добавление интегральной составляющей ПИД-регулятора

Созданная нами формула имеет один недостаток: она предполагает, что если значение ошибки равно 0, то значение параметра **Рулевое управление** (Steering) тоже равно 0. Значение ошибки, равное 0, означает, что робот TriBot находится в нужном месте, поэтому имеет смысл настроить блок **Рулевое управление** (Move Steering) на прямолинейное движение. Однако существует несколько факторов, которые не позволяют роботу двигаться прямо при значении параметра **Рулевое управление** (Steering), равном 0 (например, робот несбалансирован, диаметры колес немного различаются или они проскальзывают), при этом постоянно будет возникать ошибка, которую необходимо компенсировать.

К примеру, робот движется по наклонной плоскости, из-за чего смещается чуть влево, поэтому для движения по прямой ему нужно, чтобы значение параметра **Рулевое управление** (Steering) было равно 2. По мере движения вдоль прямой линии робот будет медленно смещаться влево, а ошибка увеличиваться. Значение ошибки будет изменяться незначительно, поэтому дифференциальная составляющая не внесет никаких корректировок. Пропорциональная составляющая изменит значение параметра **Рулевое управление** (Steering) и вернет робота обратно к краю линии. Затем робот снова будет смещаться влево, а потом возвращаться к краю линии. Так будет продолжаться вечно, а траектория движения робота будет выглядеть, как на рис. 19.15.

Рис. 19.15. Небольшие колебания, вызванные постоянной ошибкой

Для того чтобы это исправить, необходимо добавить в выражение третью *интегральную* составляющую. Интеграл суммирует все ошибки, с которыми программа столкнулась до сих пор. Если все сбалансировано, то сумма всех значений ошибки будет равна 0, поскольку некоторые из них будут положительными, а другие отрицательными, так что они должны отменить друг друга. Если робот смещается в одну сторону, как в только что описанном примере, то интеграл будет мерой этого смещения, и мы сможем использовать его для исправления ошибки (рис. 19.16).

Один из способов расчета интеграла заключается в простом суммировании всех значений ошибки. Недостатком данного подхода является то, что при его использовании ошибки, возникшие давно, имеют такой же вес, что и у недавних ошибок. Например, интеграл может оказаться большим, когда робот совершает крутой поворот, поскольку при этом суммируются многочисленные большие значения ошибок. Если программа не столкнется с таким же количеством ошибок в противоположном направлении, интеграл может оставаться большим, даже когда программа начнет двигаться вдоль прямого отрезка линии, где значение ошибки постоянно остается близким к 0.

Эту проблему можно решить, уменьшая значение интеграла при каждом выполнении цикла перед добавлением

нового значения ошибки. При этом старые значения ошибки со временем удаляются. В программе *LineFollower* для вычисления нового интеграла (New integral) на основании предыдущего значения интеграла (Integral) и ошибки (Error) мы будем использовать выражение:

New integral = 0,5 × Integral + Error

Рис. 19.16. Вычисление интеграла

Мы добавим в нашу программу третий интегральный коэффициент усиления *Ki* и изменим формулу для расчета значения параметра **Рулевое управление** (Steering) следующим образом:

Steering = *Kp* × Error + *Kd* × Derivative + *Ki* × Integral

На рис. 19.17 и 19.18 показана программа EV3. Организация блоков **Математика** (Math) и шин данных близко соответствует приведенному в тексте описанию формул. Если ты используешь USB-соединение с модулем EV3, то можешь отслеживать значения в шинах данных, пока робот следует вдоль линии, чтобы увидеть, как изменяются значения, или выявить ошибки, возникшие при копировании программы. Убедившись в работоспособности программы, ты можешь сократить ее, объединив некоторые из блоков **Математика** (Math).

Настройка ПИД-регулятора

Одна из особенносте, делающая ПИД-регулятор предпочтительным решением для многих задач, заключается в том, что его алгоритм, а следовательно, и код не нужно менять для других условий и даже других приложений. Чтобы использовать этот код для решения конкретной задачи часто достаточно *настроить* регулятор, т. е. выбрать значения для трех коэффициентов усиления: *Kp*, *Kd* и *Ki*. Цель заключается в установке коэффициентов, позволяющих программе оставаться в Хорошей зоне. Дифференциальная и интегральная составляющие не обязательно помогут роботу вернуться в эту зону, однако их правильные значения могут помочь роботу ее не покидать.

Вот как можно настроить программу с ПИД-регулятором для следования вдоль линии:

1. Задай значение 50 для переменной Power.

2. Начни со значений *Kd* = 0, *Ki* = 0 и *Kp* = 1. При целевом значении 60 параметр **Рулевое управление** (Steering) будет варьироваться в пределах от –60 до 40 по мере изменения нормализованного показания датчика в диапазоне от 0 до 100.

3. Протестируй программу сначала на прямой линии. Значение *Kp* = 1, вероятно, окажется слишком большим и вызовет заметные колебания. Постепенно уменьшай значение *Kp* на 0,05 до тех пор, пока робот не перестанет колебаться из стороны в сторону при следовании вдоль линии; допустимо лишь небольшое отклонение в сторону от ее края.

4. Постепенно увеличивай значение *Ki* на 0,01, пока робот не начнет двигаться вдоль края прямой линии без колебаний. Если робот не будет постоянно смещаться в сторону, ты сможешь оставить значение *Ki* = 0. Помни о том, что высокое значение *Ki* (выше 0,05) приведет к усилению колебаний.

5. Теперь протестируй программу на линии с кривыми участками. Увеличивай значение переменной Power до тех пор, пока робот не утратит способность совершать повороты.

6. Постепенно увеличивай значение *Kd* на 1 до тех пор, пока робот не сможет преодолеть всю траекторию.

При тестировании программы на своей линии я использовал значение переменной Power = 80 с коэффициентами *Kp* = 0,7, *Kd* = 12 и *Ki* = 0,05. Тебе, вероятно, придется использовать несколько иные значения.

Для других программ коэффициенты усиления могут быть совершенно другими, поскольку их значения сильно зависят от взаимосвязи между показанием датчика и регулируемым параметром (обычно это **Рулевое управление** (Steering) или **Мощность** (Power)). Взаимосвязь между этими тремя коэффициентами усиления зависит от ожидаемой степени изменения значения ошибки. В случае с программой для следования вдоль линии мы ожидаем получить небольшое значение ошибки при движении по прямой линии, что соответствует небольшому значению коэффициента *Kp*. Если робот не смещается в сторону при движении, то значение ошибки является небольшим или вообще нулевым, в этом случае коэффициент *Ki* мал или равен 0. При поворотах значения ошибки изменяются сильнее и быстрее, поэтому для того, чтобы робот не сбился с пути, необходимо использовать относительно большой коэффициент *Kd*.

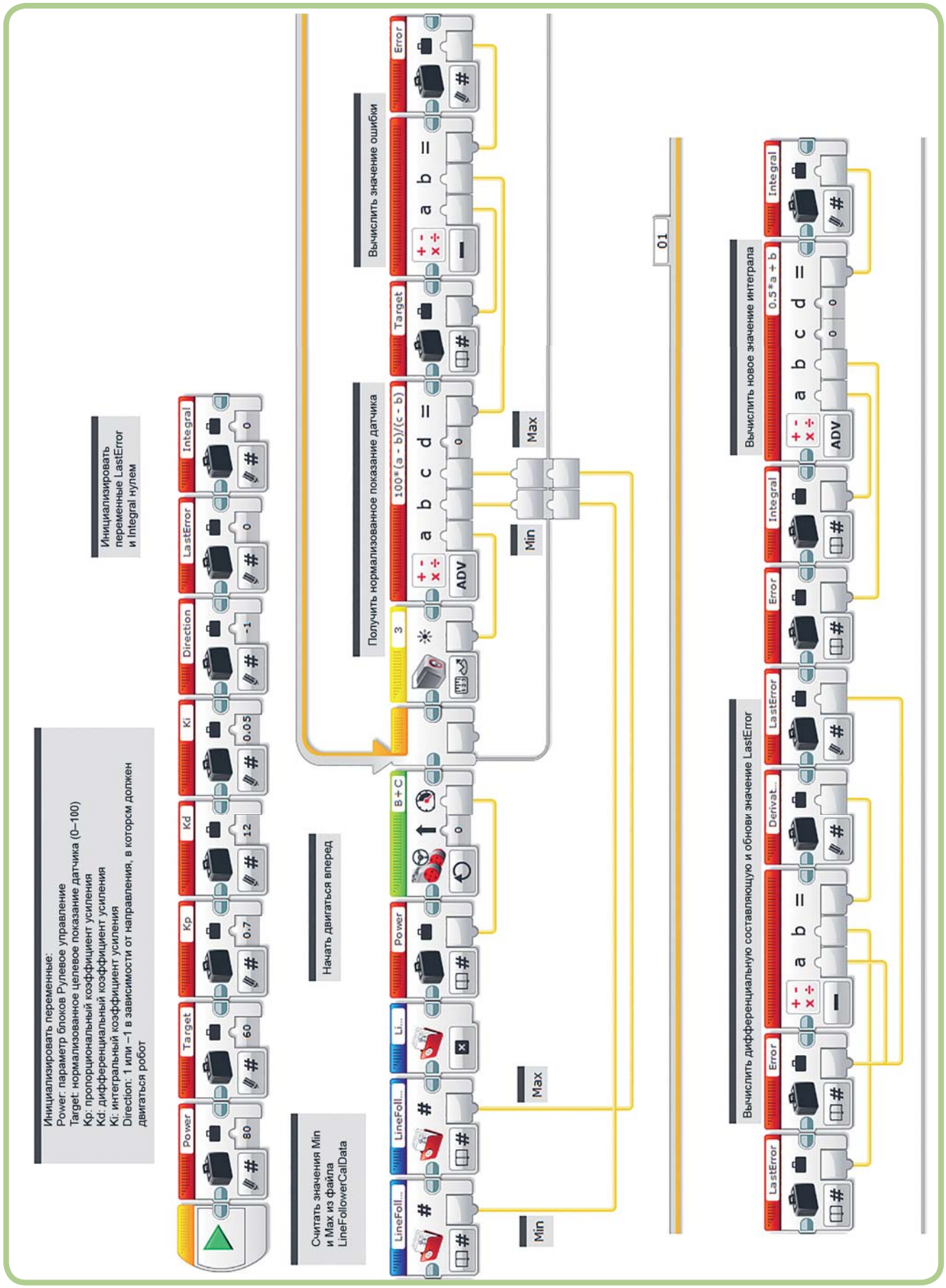


Рис. 19.17. Программа LineFollower с ПИД-регулятором, часть 1

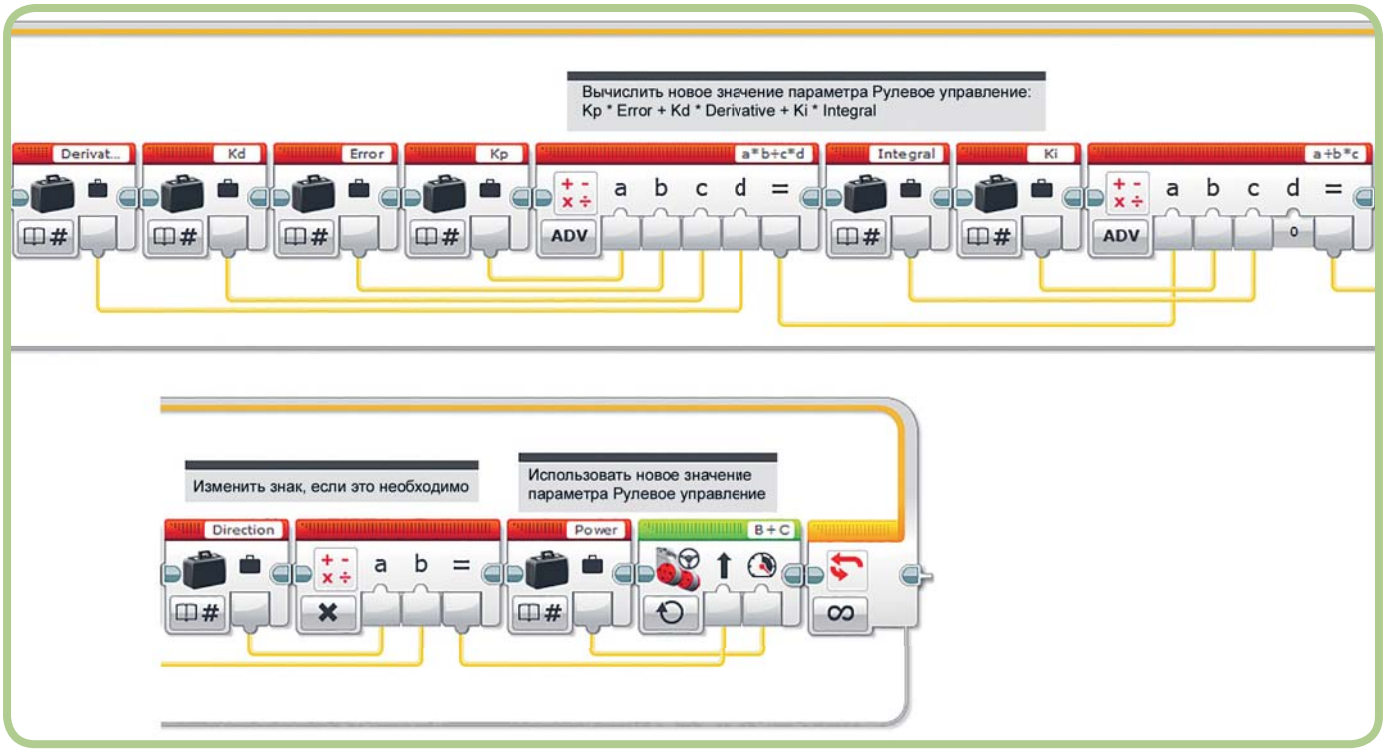


Рис. 19.18. Программа LineFollower с ПИД-регулятором, часть 2

Дальнейшее исследование

Попробуй выполнить перечисленные далее действия, чтобы еще лучше разобраться с движением вдоль линии и ПИД-регулятором:

1. Адаптируй программу *LineFollower* для разных траекторий и движения в разных направлениях по одной и той же линии (например, по часовой стрелке и против часовой стрелки по овалу). Посмотри, сможешь ли ты найти параметры, работающие в большинстве случаев, и настроить их с целью обеспечения небольшого улучшения в каждой ситуации.

2. Создай контейнер «Мой блок» из блоков, составляющих ПИД-регулятор. В качестве входных параметров используй целевое значение, нормализованное показание датчика, коэффициенты *Kp*, *Kd*, *Ki* и значение переменной Direction, а в качестве выходного параметра — результат вычисления.

3. Используй ПИД-регулятор в виде контейнера «Мой блок» в программе *WallFollower* вместо двухпозиционного регулятора.

4. Создай программу *RemoteFollower*, которая заставляет робота TriBot следовать за удаленным инфракрасным маяком. Используй значение **Направление маяка** (Beacon Heading) для регулирования параметра **Рулевое управление** (Steering) и значение **Приближение маяка** (Beacon Proximity) для регулирования скорости. Подсказка: для решения этой задачи потребуются два ПИД-регулятора.

Заключение

Написание программы для следования вдоль линии — это классическое упражнение по робототехнике, которое позволяет тебе отточить все навыки программирования EV3. На примере программы *LineFollowerCal* и изменений, внесенных в программу *LineFollower* из гл. 13, был продемонстрирован способ использования файла для хранения параметра программы. Это позволяет избежать жесткого кодирования значений, что делает программу более гибкой. Ты можешь использовать эту технику в любой программе, где требуется изменение целевых показаний датчика или других параметров.

В итоговой версии программы *LineFollower* используется алгоритм ПИД-регулирования для улучшения реагирования робота TriBot на изменения в направлении линии. Использование показаний датчика цвета и некоторых сложных концепций для определения того, насколько должен повернуться робот, позволяет ему двигаться быстрее и оставаться ближе к линии. Использование показаний датчиков для управления моторами робота является базовой частью многих программ, а эксперименты с различными алгоритмами управления представляют собой отличный способ расширения своих знаний в области робототехники при одновременном оттачивании навыков программирования.

A

Совместимость платформ NXT и EV3

Компания LEGO проделала замечательную работу, чтобы обеспечить слаженную работу модуля и программного обеспечения EV3 с существующим оборудованием NXT. На самом деле моторы и датчики NXT отлично сочетаются с набором EV3. Значит, при наличии набора NXT ты можешь объединить его с конструктором EV3, чтобы расширить свои возможности по сборке роботов. Для школ и команд FLL эта обратная совместимость означает, что инвестиции в продукты NXT не пропадут при переходе к использованию системы EV3.

Ты можешь легко использовать моторы и датчики NXT с модулем и программным обеспечением EV3, при этом все будет функционировать ожидаемым образом. Однако по большей части это работает только в одном направлении; датчики EV3 не совместимы с модулем NXT (хотя моторы EV3 будут с ним работать), а с помощью программного обеспечения NXT невозможно программировать модуль EV3. Среду EV3 можно использовать для программирования модуля NXT, но с некоторыми ограничениями.

Моторы

Мотор NXT очень похож на большой мотор EV3. Ты можешь использовать любой мотор с модулем и программным обеспечением EV3 или NXT. Средний мотор EV3 также может работать с модулем и программным обеспечением NXT.

Датчики

Все датчики NXT совместимы с модулем и программным обеспечением EV3. Блоки программирования EV3 могут работать с датчиками любой из этих систем; например, блок **Датчик цвета** (Color Sensor) работает с датчиками цвета NXT и EV3.

Программное обеспечение EV3 Home Edition по умолчанию не поддерживает ультразвуковой датчик и датчик звука. Если ты хочешь использовать любой из этих датчиков, загрузи для них блоки с сайта: **www.lego.com/en-us/mindstorms/downloads**. После загрузки блока выбери команду меню **Инструменты** (Tools) ⇒ **Импорт блоков** (My Block Import), чтобы открыть окно импорта и экспорта блоков (рис. А.1). Щелкни по кнопке **Просмотреть** (Browse) для выбора папки, в которую были загружены блоки. Затем выбери блоки из списка и щелкни по кнопке **Импорт** (Import). Новые блоки датчиков появятся на вкладке с блоками датчиков палитры программирования, а среди настроек блоков **Ожидание** (Wait), **Цикл** (Loop) и **Переключатель** (Switch) появятся параметры соответствующих датчиков.

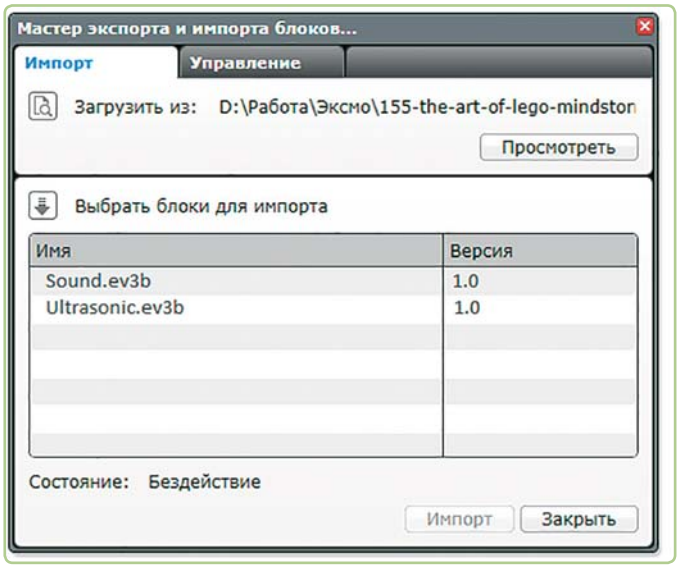


Рис. А.1. Окно импорта и экспорта блоков

Датчик цвета NXT работает с блоком **Датчик цвета** (Color Sensor) (и соответствующими режимами блоков **Ожидание** (Wait), **Цикл** (Loop) и **Переключатель** (Switch)) при использовании режимов **Яркость отраженного света** (Reflected Light Intensity) и **Яркость внешнего освещения** (Ambient Light Intensity). Однако при выборе для датчика цвета одного из режимов **Цвет** (Color) используется показание яркости внешнего освещения, в результате чего твоя программа может работать не так, как ты ожидаешь.

Отметим, что существует проблема с использованием датчиков NXT, суть которой в том, что устройство некоторых старых датчиков касания NXT из набора NXT 1.0 не позволяет им работать с модулем EV3. Чтобы выяснить, будет ли твой датчик касания работать, ты можешь использовать вкладку **Представление порта** (Port View). Убедись, что модуль EV3 подключен к программному обеспечению, и присоедини датчик. Если датчик отображается в окне **Представление порта** (Port View), он будет работать с модулем EV3.

Напомним, что датчики EV3 нельзя использовать с модулем NXT.

Программное обеспечение

Программное обеспечение NXT нельзя использовать для программирования модуля EV3, но ты можешь программировать модуль NXT с помощью программного обеспечения EV3 со следующими оговорками. В большинстве случаев с модулем NXT проще использовать программное обеспечение NXT, не EV3, исключением является только классная комната, где применяются наборы и NXT, и EV3. В этом случае проще, когда все используют одну и ту же версию программного обеспечения. Далее описаны нюансы использования программного обеспечения EV3 с модулем NXT:

– Для подключения модуля NXT к компьютеру следует использовать USB-кабель. Соединение Bluetooth работать не будет.
– Модуль NXT имеет экран меньшего размера по сравнению с модулем EV3 (100×64 против 178×128), поэтому многие изображения будут отображаться на нем некорректно; нижняя и правая стороны изображения могут быть обрезаны. Если модуль NXT подключен к программному обеспечению, то в окне предварительного просмотра блока **Экран** (Display) отобразится обрезанное изображение (рис. А.2).

Рис. А.2. Обрезанное изображение Big smile на экране модуля NXT

– При использовании модуля EV3 в блоке **Экран** (Display) нумерация пикселов начинается с верхнего левого угла направо (рис. А.3), а при использовании модуля NXT, нумерация начинается с нижнего левого угла вниз, как показано на рис. А.4. Это означает, что код для рисования необходимо скорректировать для работы с модулем NXT.

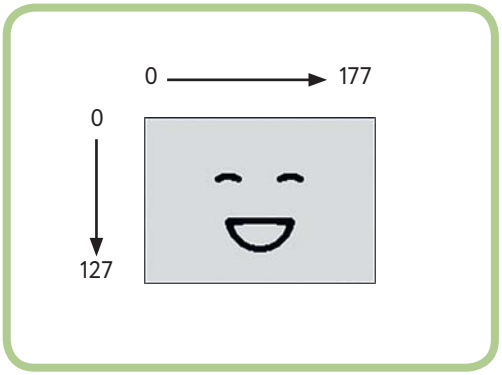


Рис. А.3. Нумерация пикселов при использовании модуля EV3

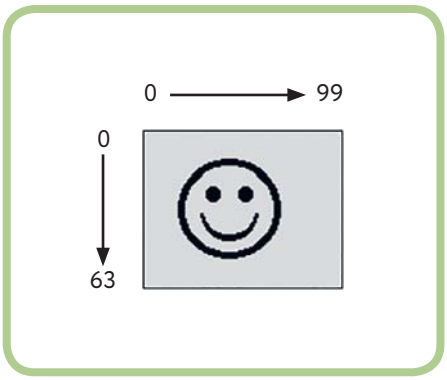


Рис. А.4. Нумерация пикселов при использовании модуля NXT

– Блок **Экран** (Display) не поддерживает параметры **Шрифт** (Font) и **Цвет** (Color).
– Блоки **Индикатор состояния модуля** (Brick Status Light), **Операции над массивом** (Array Operations), **Инвертирование мотора** (Invert Motor) и **Средний мотор** (Medium Motor) не поддерживаются. Если при использовании одного из этих блоков твой модуль NXT подключен к программному обеспечению, ты увидишь предупреждение, показанное на рис. А.5.

*Рис. А.5. Модуль NXT не поддерживает блок **Индикатор состояния модуля***

– Режимы **Дополнения** (Advanced) и **Показатель степени** (Exponent) блока **Математика** (Math) не поддерживаются.
– Блоки **Bluetooth-подключение** (Bluetooth Connection) и **Сообщения** (Messaging) не поддерживаются.
– Датчик цвета NXT не работает с блоком **Датчик цвета** (Color Sensor) и соответствующими режимами. Эту проблему можно обойти, используя блок **Датчик звука** (Sound Sensor) в режиме **dB** для измерения отраженного света и в режиме **dBa** для измерения внешнего освещения.

Б

Веб-сайты, посвященные набору EV3

В данном приложении приведен список веб-сайтов, содержащих полезную информацию о программировании EV3. Многие из них также содержат ссылки на другие ресурсы, которые могут тебя заинтересовать, например, на инструкции по сборке роботов EV3 или на сайты, посвященные более общим темам, связанным с робототехникой.

— **mindstorms.lego.com**
Официальный сайт LEGO MINDSTORMS содержит последние официальные новости и справочную информацию, касающуюся набора EV3. Кроме того, на нем можно найти множество проектов, предоставленных пользователями.

— **www.thenxtstep.com**
В блоге NXT STEP приведены новости и информация о событиях, связанных с роботами EV3.

— **www.mindboards.net**
На форуме этого сайта указано много полезной информации, в том числе решений большинства распространенных проблем, с которыми сталкиваются новые пользователи EV3. Этот сайт часто посещают многие продвинутые пользователи, у которых ты можешь получить ответы на свои вопросы.

— **forums.usfirst.org**
Здесь представлены форумы для участников конкурса FIRST LEGO League. Форумы по программированию содержат множество полезных сведений, которые пригодятся даже тем, кто (пока еще) не участвует в конкурсах по робототехнике.

— **www.legoengineering.com**
Сайт партнерства Образовательного центра по распространению инженерных наук при Университете Тафтса (CEEO) и LEGO Education. Его цель — использование системы LEGO MINDSTORMS для привлечения студентов к изучению науки, техники, инженерного дела и математики.

— **groups.google.com/forum/#!forum/legoengineering**
Группа Google, посвященная использованию робототехники, в том числе наборов EV3, в школе.

— **bricks.stackexchange.com**
Сайт для энтузиастов LEGO и еще один хороший ресурс для поиска ответов на вопросы и решений проблем, связанных с программированием.

Предметный указатель

U

V

W

А

Б

В

Г

Д

З

И

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Ц

Ч

Ш

Э

Я

Издание для досуга

ПОДАРОЧНЫЕ ИЗДАНИЯ. КОМПЬЮТЕР

Гриффин Терри

ИСКУССТВО ПРОГРАММИРОВАНИЯ LEGO MINDSTORMS EV3

Главный редактор *Р. Фасхутдинов*
Руководитель направления *В. Обручев*
Ответственный редактор *Е. Истомина*
Литературный редактор *Ю. Голобокова*
Младший редактор *А. Захарова*
Художественный редактор *К. Доброслов*
Компьютерная верстка *Э. Брегис*
Корректоры *А. Баскакова, Л. Макарова*

Страна происхождения: Российская Федерация
Шығарылған елі: Ресей Федерациясы

ООО «Издательство «Эксмо»
123308, Россия, город Москва, улица Зорге, дом 1, строение 1, этаж 20, каб. 2013.
Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru
Өндіруші: «ЭКСМО» АҚБ Баспасы,
123308, Ресей, қала Мәскеу, Зорге көшесі, 1 үй, 1 ғимарат, 20 қабат, офис 2013 ж.
Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru.
Тауар белгісі: «Эксмо»
Интернет-магазин : www.book24.ru

Интернет-магазин : www.book24.kz
Интернет-дүкен : www.book24.kz
Импортёр в Республику Казахстан ТОО «РДЦ-Алматы».
Қазақстан Республикасындағы импорттаушы «РДЦ-Алматы» ЖШС.
Дистрибьютор и представитель по приему претензий на продукцию,
в Республике Казахстан: ТОО «РДЦ-Алматы»
Қазақстан Республикасында дистрибьютор және өнім бойынша арыз-талаптарды
қабылдаушының өкілі «РДЦ-Алматы» ЖШС,
Алматы қ., Домбровский көш., 3«а», литер Б, офис 1.
Тел.: 8 (727) 251-59-90/91/92; E-mail: RDC-Almaty@eksmo.kz
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта: www.eksmo.ru/certification

Сведения о подтверждении соответствия издания согласно законодательству РФ
о техническом регулировании можно получить на сайте Издательства «Эксмо»
www.eksmo.ru/certification
Өндірген мемлекет: Ресей. Сертификация қарастырылмаған

Дата изготовления / Подписано в печать 19.01.2022.
Формат 60x84 1/8. Печать офсетная. Усл. печ. л. 31,73.
Тираж экз. Заказ

ISBN 978-5-04-095834-4

Москва. ООО «Торговый Дом «Эксмо»
Адрес: 123308, г. Москва, ул. Зорге, д.1, строение 1.
Телефон: +7 (495) 411-50-74. **E-mail:** reception@eksmo-sale.ru

По вопросам приобретения книг «Эксмо» зарубежными оптовыми покупателями обращаться в отдел зарубежных продаж ТД «Эксмо»
E-mail: **international@eksmo-sale.ru**

International Sales: International wholesale customers should contact Foreign Sales Department of Trading House «Eksmo» for their orders.
international@eksmo-sale.ru

По вопросам заказа книг корпоративным клиентам, в том числе в специальном оформлении, обращаться по тел.: +7 (495) 411-68-59, доб. 2261.
E-mail: **ivanova.ey@eksmo.ru**

Оптовая торговля бумажно-беловыми и канцелярскими товарами для школы и офиса «Канц-Эксмо»:
Компания «Канц-Эксмо»: 142702, Московская обл., Ленинский р-н, г. Видное-2, Белокаменное ш., д. 1, а/я 5. Тел./факс: +7 (495) 745-28-87 (многоканальный).
e-mail: **kanc@eksmo-sale.ru**, сайт: www.**kanc-eksmo.ru**

Филиал «Торгового Дома «Эксмо» в Нижнем Новгороде
Адрес: 603094, г. Нижний Новгород, улица Карпинского, д. 29, бизнес-парк «Грин Плаза»
Телефон: +7 (831) 216-15-91 (92, 93, 94). **E-mail**: reception@eksmonn.ru

Филиал ООО «Издательство «Эксмо» в г. Санкт-Петербурге
Адрес: 192029, г. Санкт-Петербург, пр. Обуховской обороны, д. 84, лит. «Е»
Телефон: +7 (812) 365-46-03 / 04. **E-mail**: server@szko.ru

Филиал ООО «Издательство «Эксмо» в г. Екатеринбурге
Адрес: 620024, г. Екатеринбург, ул. Новинская, д. 2щ
Телефон: +7 (343) 272-72-01 (02/03/04/05/06/08)

Филиал ООО «Издательство «Эксмо» в г. Самаре
Адрес: 443052, г. Самара, пр-т Кирова, д. 75/1, лит. «Е»
Телефон: +7 (846) 207-55-50. **E-mail**: RDC-samara@mail.ru

Филиал ООО «Издательство «Эксмо» в г. Ростове-на-Дону
Адрес: 344023, г. Ростов-на-Дону, ул. Страны Советов, 44А
Телефон: +7(863) 303-62-10. **E-mail**: info@rnd.eksmo.ru

Филиал ООО «Издательство «Эксмо» в г. Новосибирске
Адрес: 630015, г. Новосибирск, Комбинатский пер., д. 3
Телефон: +7(383) 289-91-42. E-mail: eksmo-nsk@yandex.ru

Обособленное подразделение в г. Хабаровске
Фактический адрес: 680000, г. Хабаровск, ул. Фрунзе, 22, оф. 703
Почтовый адрес: 680020, г. Хабаровск, А/Я 1006
Телефон: (4212) 910-120, 910-211. **E-mail**: eksmo-khv@mail.ru

Филиал ООО «Издательство «Эксмо» в г. Тюмени
Центр оптово-розничных продаж Cash&Carry в г. Тюмени
Адрес: 625022, г. Тюмень, ул. Пермякова, 1а, 2 этаж. ТЦ «Перестрой-ка»
Ежедневно с 9.00 до 20.00. Телефон: 8 (3452) 21-53-96

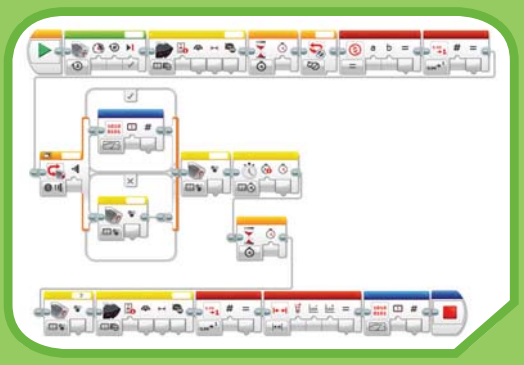
Республика Беларусь: ООО «ЭКСМО АСТ Си энд Си»
Центр оптово-розничных продаж Cash&Carry в г. Минске
Адрес: 220014, Республика Беларусь, г. Минск, проспект Жукова, 44, пом. 1-17, ТЦ «Outleto»
Телефон: +375 17 251-40-23; +375 44 581-81-92
Режим работы: с 10.00 до 22.00. **E-mail:** exmoast@yandex.by

Казахстан: «РДЦ Алматы»
Адрес: 050039, г. Алматы, ул. Домбровского, 3А
Телефон: +7 (727) 251-58-12, 251-59-90 (91,92,99). E-mail: RDC-Almaty@eksmo.kz

Украина: ООО «Форс Украина»
Адрес: 04073, г. Киев, ул. Вербовая, 17а
Телефон: +38 (044) 290-99-44, (067) 536-33-22. **E-mail**: sales@forsukraine.com

Полный ассортимент продукции ООО «Издательство «Эксмо» можно приобрести в книжных магазинах «Читай-город» и заказать в интернет-магазине: www.chitai-gorod.ru.
Телефон единой справочной службы: 8 (800) 444-8-444. Звонок по России бесплатный.

Интернет-магазин ООО «Издательство «Эксмо»
www.book24.ru
Розничная продажа книг с доставкой по всему миру.
Тел.: +7 (495) 745-89-14. E-mail: **imarket@eksmo-sale.ru**



ИСКУССТВО ПРОГРАММИРОВАНИЯ LEGO® MINDSTORMS® EV3

С этим красочным цветным наглядным руководством по программированию роботов LEGO® MINDSTORMS® EV3 вы научитесь создавать сложные программы из таких элементов, как блоки, шины данных, файлы и переменные. Вы познакомитесь с правилами программирования, освоите полезные стратегии отладки и навыки управления ресурсами, которые пригодятся при программировании на любом языке.

Все описанные в книге программы работают с одним универсальным тестовым роботом, которого вы создадите в самом начале и запрограммируете на то, чтобы он:

- Реагировал на разные условия окружающей среды и выполнял команды
- Находил выход из лабиринта
- Отображал на экране модуля EV3 рисунки, введенные с помощью датчиков и шин данных
- Играл в игры, используя массивы для сохранения рекорда

И многое другое.

Эта книга предназначена как для пользователей домашней, так и образовательной версий EV3, поэтому идеально подходит детям, родителям и учителям. Не важно, расположена ли ваша робототехническая лаборатория в гостиной или в классе, это руководство по программированию EV3 — то, что нужно!

Об авторе

Терри Гриффин уже более 20 лет работает инженером-программистом и создает программное обеспечение для управления различными машинами. Он работает в компании Carl Zeiss с гелиевым ионным микроскопом ORION — программирует его пользовательский интерфейс и разрабатывает ПО для высокоуровневого управления оборудованием.

Для обучения по этой книге вам понадобится набор MINDSTORMS® EV3 домашней или образовательной версии № 31313 или № 45544.

Всем известна любовь к конструкторам Lego как детей, так и взрослых. Конструкторы помогают развивать внимание, терпение, усидчивость, логическое мышление, моторику рук и многие другие качества. Мы живем в цифровую эпоху, где навык цифровых компетенций у детей является таким же важным, как умение читать и писать. Данное пособие является отличным путеводителем в захватывающий мир программирования лего-роботов, которое не оставит равнодушным ни детей, ни взрослых. В этой книге в игровой форме подается информация о крайне важном и нужном цифровом навыке будущего. Автор постарался сделать все для того, чтобы книга была по-настоящему универсальной и могла использоваться как в домашних условиях, так и для организации кружкового движения в общеобразовательных организациях.

Васильев Артем Игоревич, ректор Университета «Синергия»

no starch press
THE FINEST IN GEEK ENTERTAINMENT™
www.nostarch.com

БОМБОРА
ИЗДАТЕЛЬСТВО

БОМБОРА – лидер на рынке полезных и вдохновляющих книг.
Мы любим книги и создаем их, чтобы вы могли творить, открывать мир, пробовать новое, расти. Быть счастливыми. Быть на волне.

bomborabooks bombora.ru

СЕРИЯ: ПОДАРОЧНЫЕ ИЗДАНИЯ. КОМПЬЮТЕР

ISBN 978-5-04-095834-4