2

ДВИЖЕНИЕ И РИСОВАНИЕ

Вы уже ориентируетесь в интерфейсе. Теперь можно использовать больше инструментов программирования на Scratch. Вот что вам предстоит сделать в этой главе:

- познакомиться с командами разделов **Движение** и **Перо**;
- анимировать спрайты и передвигать их по **Сцене**;
- рисовать геометрические узоры и создавать игры;
- узнать, почему клонирование спрайтов — такой ценный инструмент.

Самое время надеть шляпу художника и нырнуть с головой в мир компьютерной графики!

Использование команд движения

Если вы хотите делать игры или другие анимированные программы, чтобы перемещать спрайты по **Сцене**, нужно использовать блоки из раздела **Движение**. Более того, вам нужно будет давать спрайтам команду переместиться в конкретную точку на **Сцене** или повернуть в определенном направлении.

Абсолютное движение

На рис. 1.4 мы видели на **Сцене** прямоугольную систему координат 480 × 360 с центром в точке (0, 0). В Scratch есть четыре команды абсолютного

движения (**перейти в**, **плыть к**, **установить x в** и **установить y в**), которые дают вам возможность сообщить спрайту, куда ему переместиться.

> *Если вы хотите узнать больше об этих или любых других блоках, используйте окошко* **Подсказка**, *которое находится справа от панели* **Скрипты**. *Если вы не видите его, кликните по знаку вопроса в верхнем правом углу редактора проектов.*

Чтобы продемонстрировать, как работают эти команды, представим, что вы хотите, чтобы спрайт Ракета на рис. 2.1 попал в имеющую форму звезды цель, расположенную в точке (200, 150). Самый очевидный способ — использовать блок **перейти**, как показано на рисунке справа. Ось *x* показывает, куда спрайту нужно переместиться по горизонтали, а ось *y* — по вертикали.

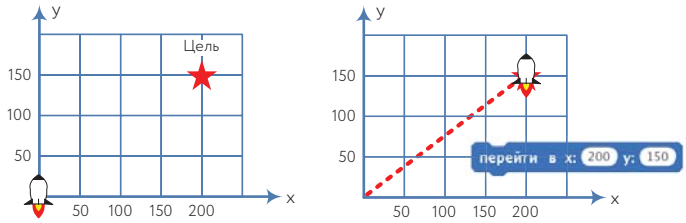


Рис. 2.1. При помощи блока **перейти** *вы можете переместить спрайт в любую точку на* **Сцене**

Ракета не повернется носом к цели, но будет двигаться по невидимой прямой, соединяющей ее исходное местоположение, точку (0, 0), с точкой (200, 150). Вы можете заставить ракету замедлить движение, если используете команду **плыть к**. Она практически идентична **перейти**, но позволяет устанавливать время, которое понадобится ракете, чтобы достичь цели.

Другой способ попасть в цель — независимо друг от друга изменить координаты *x* и *y* спрайта-ракеты при помощи блоков **установить x в** и **установить y в**, как показано на рис. 2.2. Помните, как вы использовали блок **установить x в**, когда делали игру Pong (см. рис. 1.20)?

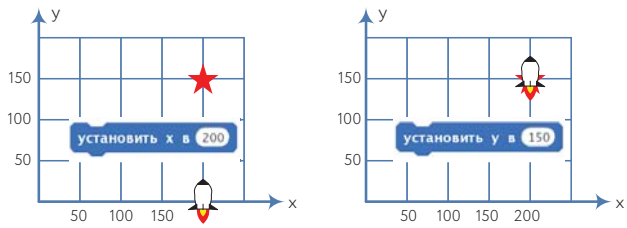
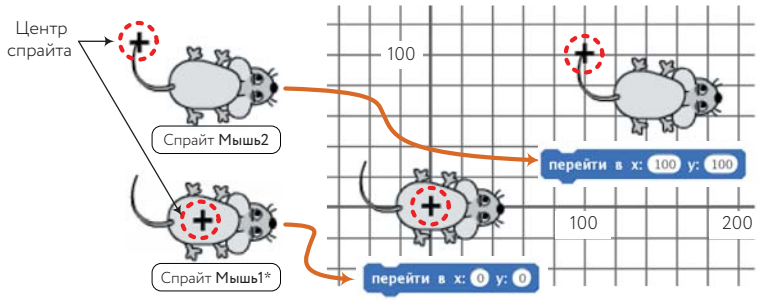


Рис. 2.2. Вы можете независимо изменить координаты x и y спрайта

Вы всегда можете видеть текущие координаты спрайта в правом верхнем углу поля скриптов. Если вы хотите, чтобы эта информация отображалась на **Сцене**, используйте блоки-репортеры **положение x** и **положение y**. Поставьте галочки в чекбоксах около них, чтобы увидеть их значения на **Сцене**.

Команды движения работают относительно центра спрайта, который вы можете установить в графическом редакторе. Например, если отправить спрайт в точку (100, 100), он переместится так, что его центр окажется в точке (100, 100), как показано на рис. 2.3. Поэтому, если вы рисуете или импортируете костюм для спрайта, который планируете перемещать, обратите внимание на его центр!

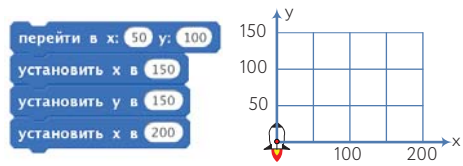


* В библиотеке спрайтов Mouse1.

Рис. 2.3. Команды движения относительно центра спрайта

УПРАЖНЕНИЕ 2.1

Список координат спрайта-ракеты после выполнения каждой из команд приведенного ниже скрипта.



Относительное движение

А теперь посмотрите внимательно на сетку координат на рис. 2.4, где показаны другие спрайты Ракета и Цель. На этот раз вы не видите координат, поэтому точное расположение спрайтов вам неизвестно. Если бы вам нужно было объяснить ракете, как попасть в цель, вы могли бы сказать: «Сделай три шага, потом поверни направо и сделай еще два шага».

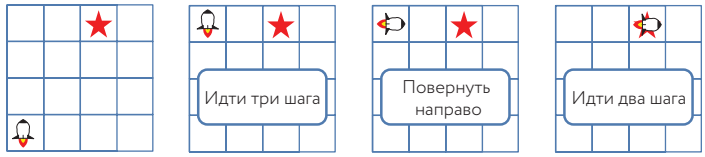


Рис. 2.4. Вы можете перемещать спрайт по **Сцене**, *используя команды относительного движения*

Идти и **повернуть** — *команды относительного движения*. Например, первая команда «идти» сверху заставляет ракету двигаться вверх, а вторая отправляет ее направо. Движение зависит от текущего *направления* спрайта. На рис. 2.5 изображены направления движения в Scratch.

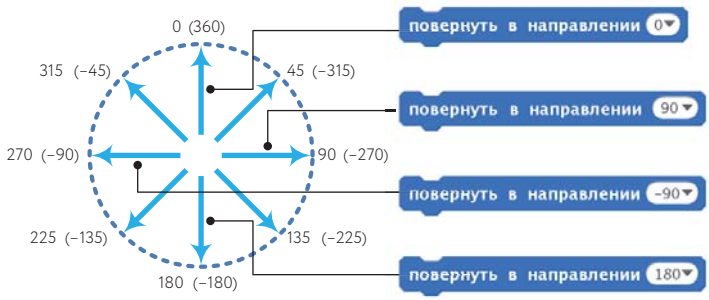


Рис. 2.5. В Scratch 0 — вверх, 90 — вправо, 180 — вниз и –90 — влево

Вы можете повернуть спрайт к конкретной цели (курс) с помощью команды **повернуть в направлении**. Чтобы выбрать направление, кликните по указывающей вниз стрелке и выберите нужный вариант из выпадающего меню. Чтобы выбрать другое направление, вставьте нужное значение в белое редактируемое поле. Можно даже использовать отрицательные значения! (Например, если написать 45 или –315, спрайт в обоих случаях повернет на северо-восток.)

Актуальное направление спрайта отражено в поле информации о нем. Также можете поставить галочку в чекбоксе напротив блока **направление** *(в разделе* **Движение***), чтобы увидеть направление на* **Сцене**.

Теперь, когда вы знаете, как в Scratch работает направление, посмотрим на команды относительного движения (**идти**, **изменить x на**, **изменить y на** и **повернуть**). Начнем с команд **идти** и **повернуть**, которые работают с учетом актуального положения спрайта, как показано на рис. 2.6.

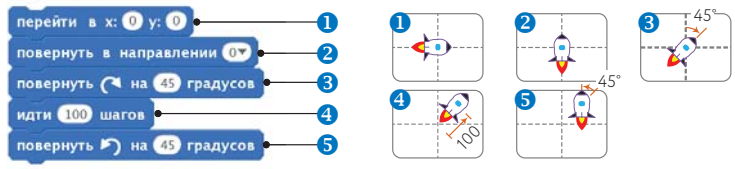


Рис. 2.6. Простой скрипт, который демонстрирует использование команд **идти** *и* **повернуть**

Во-первых, блок **перейти** ❶ двигает Ракету так, что ее центр оказывается совмещенным с центром **Сцены**. Второй командный блок ❷ направляет спрайт вверх, а третий ❸ поворачивает спрайт на 45° по часовой стрелке. Затем спрайт перемещается на 100 шагов ❹ по своему текущему направлению, прежде чем повернуть на 45° по часовой стрелке ❺, чтобы он остановился, будучи направленным вверх.

НАПРАВЛЕНИЯ И КОСТЮМЫ

Команда **повернуть в направлении** совершенно не в курсе костюма спрайта. Возьмем для примера два спрайта с рисунка слева. При помощи графического редактора мы нарисовали костюм птицы ориентированным направо, а костюм насекомого — вверх. Как вы думаете, что будет, если использовать команду **повернуть в направлении 90** (повернуться направо) для каждого из спрайтов? Можно предположить, что насекомое станет смотреть в правую сторону, но на самом деле ни один из спрайтов не пошевелится. Хотя 90° обозначается как «право», на деле это направление относится к *исходной ориентации костюма* в графическом редакторе. Так что если в нем насекомое ориентировано наверх, оно по-прежнему будет смотреть вверх после того, как вы скажете ему повернуться на 90°. Если вы хотите, чтобы ваш спрайт отреагировал на команду, как показано на рис. 2.5, нужно в графическом редакторе нарисовать костюм спрайта, ориентированный вправо (как костюм-птица на рисунке сверху).

Иногда вам, возможно, понадобится лишь переместить спрайт из его текущей позиции по горизонтали или вертикали. И здесь приходят на помощь блоки **изменить x на** и **изменить y на**. Скрипт на рис. 2.7 демонстрирует использование этих блоков.

После того как спрайт-ракета передвинулся в центр **Сцены**, первая команда **изменить x на 50** ❶ добавляет 50 к его координате *x* и он отправляется еще на 50 шагов направо. Следующая команда ❷, **изменить y на 50**, меняет на 50 координату *y*, в результате спрайт перемещается еще на 50 шагов вверх. Другие команды работают так же. Попробуйте проследить движение спрайта на рис. 2.7, чтобы найти его окончательный пункт назначения.

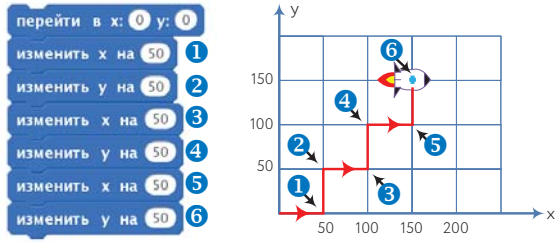
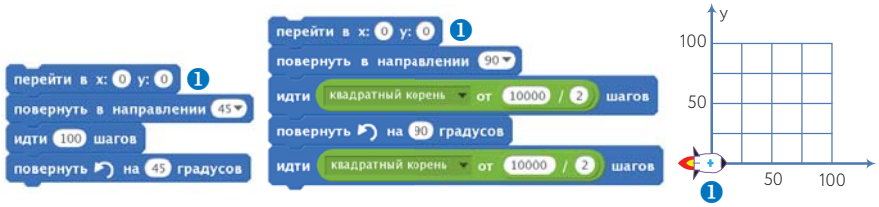


Рис. 2.7. Проведите ваш спрайт извилистой тропой при помощи **изменить x на** *и* **изменить y на**

УПРАЖНЕНИЕ 2.2

Найдите конечную позицию ракеты (*x*, *y*) после выполнения ею каждого из скриптов, показанных ниже. Какую математическую теорему можно использовать, чтобы доказать, что эти два скрипта эквивалентны?



Другие команды движения

TennisBallChaser .sb2

Осталось познакомиться всего с четырьмя командами движения: **повернуть к**; второй тип блока **перейти**; **если на краю, оттолкнуться** и **стиль вращения**. Вы уже знаете о стилях вращения и видели в действии команду **если на краю, оттолкнуться** в главе 1 (см. рис. 1.13). Чтобы посмотреть, как работают остальные две команды, создадим простую программу, в которой кот будет гоняться за теннисным мячиком, как показано на рис. 2.8.

Рис. 2.8. Программируем кота, чтобы он бегал за мячиком

* В библиотеке спрайтов Cat2 и Ball.

Как вы видите, в программе два спрайта (мы назвали их Кот и Мяч*) и два скрипта. Когда вы кликнете по зеленому флажку, спрайт Мяч будет следовать за курсором мыши. Спрайт Кот постоянно направлен в сторону мяча и движется за ним при помощи команды **плыть**. Попробуйте самостоятельно создать эту программу, чтобы посмотреть, как она работает. Блок **всегда** вы найдете в разделе **Управление**, а блоки **мышка по x** и **мышка по y** — в разделе **Сенсоры**. Готовая программа находится в файле *TennisBallChaser.sb2*. Ниже мы подробно разберем раздел **Перо** и научимся заставлять спрайты оставлять видимые следы своего передвижения.

Команды раздела Перо и программа Easy Draw

EasyDraw.sb2

Команды движения, которые вы использовали раньше, позволяют вам перемещать спрайт в любую точку на **Сцене**. Разве не было бы здорово увидеть путь, по которому перемещается ваш спрайт? Тут может помочь перо в Scratch.

У каждого спрайта есть невидимое перо, которое может быть либо поднято, либо опущено. Если оно опущено, спрайт по мере передвижения будет рисовать. В остальных случаях он передвигается, не оставляя следов. Команды из раздела **Перо** позволяют вам контролировать размеры пера, его цвет и тень.

УПРАЖНЕНИЕ 2.3

Откройте в Scratch окошко **Подсказки**, кликните по иконке с домиком, а затем по перу, чтобы получить краткое описание каждой команды **Пера**. Скрипты внизу показывают большую часть этих команд. Воссоздайте эти скрипты, запустите и опишите эффект от каждого из них. Не забудьте опустить перо спрайта вниз, прежде чем запустить скрипты. (Блок **повторить** вы найдете в разделе **Управление**.)

Давайте подробнее изучим некоторые команды пера и создадим простую программу для рисования картинок, передвигая и переворачивая спрайт на **Сцене** клавишами со стрелками. Одно нажатие на клавишу со стрелкой вверх (↑) переместит спрайт вперед на 10 шагов. Нажатие

клавиши со стрелкой вниз (↓) — на 10 шагов назад. Каждое нажатие клавиши со стрелкой вправо (→) повернет спрайт вправо на 10°, а нажатие клавиши со стрелкой влево (←) — на 10° влево.

Так, например, чтобы повернуть спрайт на 90°, как показано на рис. 2.9, вам нужно нажать клавиши со стрелкой направо или налево девять раз.

Сначала создайте новый проект в Scratch. Замените костюм кота на то, что точно показывает, когда спрайт указывает вверх, вниз, направо или налево. Хорошо подойдут костюмы beetle или cat2 (из раздела **Животные**), но вы можете выбрать любой понравившийся вам костюм. В закладке **Костюмы** нажмите кнопку **Выбрать костюм из библиотеки** и выберите подходящий костюм.

Рис. 2.9. Программа Easy Draw в действии

Теперь добавьте вашему спрайту все скрипты, показанные на рис. 2.10. Вы можете создать четыре блока **когда клавиша нажата** из блока **когда клавиша пробел нажата** в разделе **События**. Кликните по черной стрелочке в блоке и выберите клавишу со стрелкой, которая вам нужна.

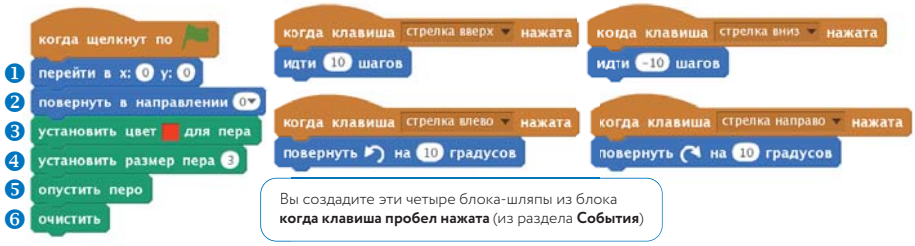


Рис. 2.10. Скрипты для программы Easy Draw

Когда вы кликаете по зеленому флажку, спрайт начинает двигаться к центру **Сцены** ❶ и поворачивается вверх ❷. Затем устанавливаются цвет ❸ и размер ❹ пера, скрипт опускает перо вниз ❺, чтобы подготовиться к рисованию. После этого программа стирает со **Сцены** все старые рисунки ❻.

Чтобы очистить сцену и начать новый рисунок, нужно кликнуть по зеленому флажку. Используйте клавиши со стрелками, чтобы нарисовать любую форму, какую захотите. Как вы думаете, какую форму создаст последовательность ↑→↑→↑→...?

УПРАЖНЕНИЕ 2.4

Добавьте еще одну функцию, чтобы перо становилось шире, когда нажата клавиша *W*, и тоньше, когда нажата *N*. Подумайте о других способах усовершенствовать программу и попробуйте их внедрить.

Сила повторения

Пока наши программы были простыми, но когда вы начнете писать более длинные скрипты, вам будет нужно повторять одни и те же блоки несколько раз подряд. Дублирование скриптов может сделать программу длиннее и запутаннее, с ней станет труднее экспериментировать. Если вам, например, нужно изменить одну цифру, придется вносить изменения в каждую копию блока. Команда **повторить** из раздела **Управление** поможет вам избежать этой проблемы.

DrawSquare.sb2

Предположим, вы хотите нарисовать квадрат, как показано на рис. 2.11 (слева). Вы можете скомандовать спрайту следовать этим повторяющимся инструкциям.

1. Пройди некоторое расстояние и повернись на 90° против часовой стрелки.
2. Пройди некоторое расстояние и повернись на 90° против часовой стрелки.
3. Пройди некоторое расстояние и повернись на 90° против часовой стрелки.
4. Пройди некоторое расстояние и повернись на 90° против часовой стрелки.

Рис. 2.11. Квадрат (слева) и скрипт для рисования его (справа), использующий последовательность команд **идти** *и* **повернуть**

На рис. 2.11 вы также видите скрипт, выполняющий эти инструкции. Обратите внимание, что в нем четыре раза повторяются команды **идти 100 шагов** и **повернуть на 90 градусов**. Мы можем избежать использования одних и тех же блоков при помощи блока **повторить**, который запускает команды внутри себя столько раз, сколько вы ему скажете, как показано на рис. 2.12. Используя блок **повторить**, вы также можете сделать свои инструкции гораздо более легкими для понимания.

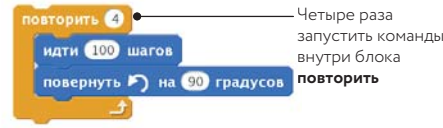


Рис. 2.12. Используем блок ***повторить****, чтобы нарисовать квадрат*

Квадрат, который вы нарисовали при помощи скрипта на рис. 2.11, зависит от того, куда ориентирован ваш спрайт в тот момент, когда вы запускаете скрипт. Эта концепция проиллюстрирована на рис. 2.13. После того как вы нарисовали квадрат, спрайт возвращается в свою исходную позицию.

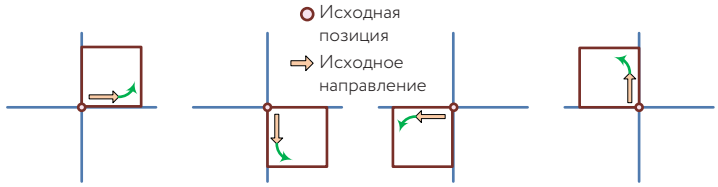


Рис. 2.13. Исходное направление спрайта меняет расположение квадрата

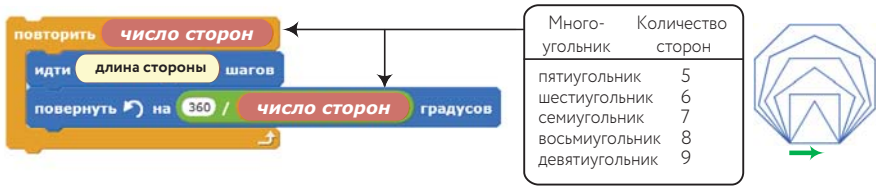
УПРАЖНЕНИЕ 2.5

Polygon.sb2

Вы можете модифицировать скрипт с рис. 2.12 так, чтобы он рисовал другие правильные многоугольники. Модифицированный скрипт имеет показанную ниже форму. Вместо «количества сторон» и «длины стороны» подставьте любые целые числа, чтобы обозначить желаемый многоугольник и контролировать его размеры. Рисунок показывает также шесть многоугольников с одинаковой длиной стороны, которые были нарисованы при помощи этого скрипта. Спрайт стартовал из своей исходной позиции в направлении, указанном зеленой стрелкой.

Откройте файл *Polygon.sb2* и запустите его, используя разные цифры для «количества сторон».

Что происходит, когда это число становится большим? Это должно дать вам представление о том, как рисовать круги.



Многоугольник	Количество сторон
пятиугольник	5
шестиугольник	6
семиугольник	7
восьмиугольник	8
девятиугольник	9

Вращающиеся квадраты

RotatedSquares.sb2

Вы можете создавать потрясающие картины, повторяя один и тот же узор в определенной последовательности.

Вот, например, скрипт на рис. 2.14 создает симпатичный узор, вращая и рисуя квадрат 12 раз подряд (блоки, опускающие и поднимающие перо, здесь не показаны).

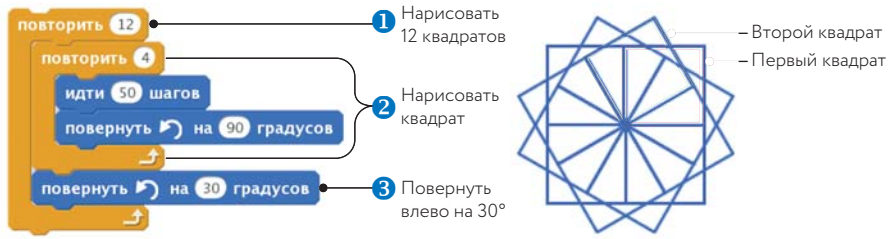


Рис. 2.14. Рисуем вращающийся квадрат

Внешний блок **повторить** ❶ работает 12 раз. За каждый повтор он рисует квадрат ❷, а потом делает поворот на 30° влево ❸, чтобы приготовиться рисовать следующий.

УПРАЖНЕНИЕ 2.6

Обратите внимание, что (12 повторов) × (30° на каждый повтор) = 360°. Как вы думаете, что произойдет, если вы измените в программе числа на 4 повтора по 90°? А как насчет 5 и 72°? Поэкспериментируйте с количеством повторов и значением угла поворота и посмотрите, что получится.

Исследуем печать

Windmill.sb2

Из предыдущего раздела вы узнали, как использовать блоки **повернуть** и **повторить**, чтобы превращать простые формы в сложные узоры. А что если вы захотите вращать более сложные формы? Вместо того чтобы рисовать простейшие формы при помощи команд **идти** и **повернуть**, вы можете создать новый костюм в графическом редакторе и использовать блок **печать**, чтобы нарисовать на **Сцене** несколько копий костюма. Чтобы проиллюстрировать эту технику, напишем программу, которая будет рисовать ветряную мельницу (рис. 2.15).

Мы нарисовали форму флага в графическом редакторе (см. рис. 2.15, слева) и использовали его в качестве костюма нашего спрайта. Мы установили центр костюма в основании флага так, чтобы вращать его вокруг этой точки.

Рис. 2.15. Команда **печать** *позволит вам легко создавать сложные геометрические узоры*

Скрипт для рисования ветряной мельницы показан в центре рис. 2.15.

Блок **повторить** выполняется восемь раз. Каждый раз на **Сцене** отпечатывается копия костюма, прежде чем спрайт повернется на 45° влево. Чтобы этот скрипт работал, вам нужно использовать блок **стиль вращения** с установленным значением **кругом**. Тогда флаг будет переворачиваться при вращении.

DrawingGeometricShapes.pdf в пакете дополнительных ресурсов (которые вы можете скачать с http://nostarch.com/learnscratch/) дает детальное представление о рисовании геометрических фигур, таких как прямоугольники, параллелограммы, ромбы, трапеции и многоугольники, а также научит вас создавать красивые рисунки из многоугольников.

УПРАЖНЕНИЕ 2.7

Блок **изменить цвет эффект на** (из раздела **Внешность**) позволит вам использовать такие графические эффекты, как цвет, завихрение или рыбий глаз. Откройте файл *Windmill.sb2* и добавьте эту команду в блок **повторить**. Поэкспериментируйте с другими графическими эффектами, чтобы получить классные узоры. Чтобы блок работал, цвет флага в графическом редакторе может быть любым, кроме черного.

Проекты Scratch

В этом разделе мы создадим две короткие программы, которые должны углубить ваше понимание уже знакомых разделов **Движение** и **Перо**. Фоны и спрайты есть в файлах проектов, поэтому мы можем сосредоточиться на скриптах, которые нужны для того, чтобы эти программы работали. Дополнительные материалы вы сможете найти в файле *BonusApplications.pdf* (*http://nostarch.com/learnscratch/*).

Некоторые из этих скриптов содержат блоки-команды, с которыми вы еще не сталкивались. Но не переживайте, если вы чего-то не поймете. В следующих главах я всё объясню.

Собери деньги

Money_NoCode.sb2

Наша первая программа — простая игра, в которой пользователь должен перемещать спрайт при помощи стрелок на клавиатуре, чтобы собрать максимальное количество мешков с золотом. Как показано на рис. 2.16, мешки появляются в случайных точках координатной сетки. Если игрок не успевает схватить мешок за три секунды, тот перемещается в другое место.

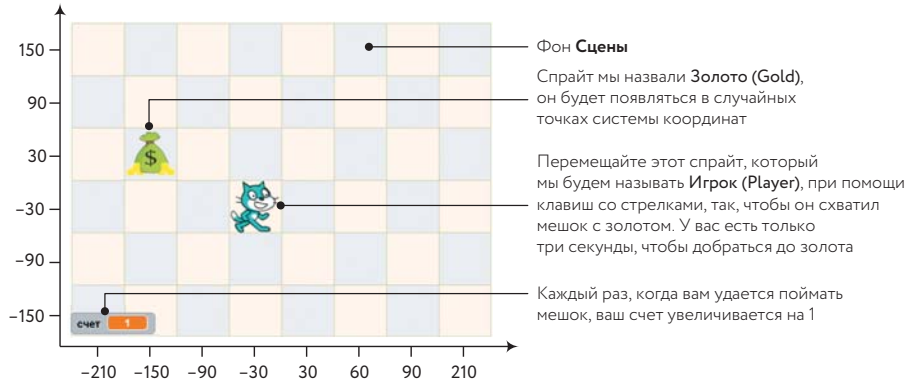


Рис. 2.16. Помогите коту схватить как можно больше мешков с золотом!

Откройте файл *Money_NoCode.sb2*. Здесь нет скриптов, но вы сейчас их создадите, а в этом файле есть еще кое-что полезное.

Оси координат, показанные на рис. 2.16, были добавлены, чтобы помочь вам понять цифры, которые используются в скриптах. Вернитесь к этому рисунку, когда нужно будет освежить в памяти картину передвижения спрайта.

Начнем с написания скриптов для спрайта Игрок, как показано на рис. 2.17.

Когда игрок кликает по зеленому флажку, спрайт перемещается в точку (–30, –30) ❶ и поворачивается вправо ❷. Остальные четыре скрипта отвечают на использование клавиш со стрелками. Когда нажимается клавиша со стрелкой, соответствующий скрипт меняет направление спрайта ❸, проигрывает короткий звук (при помощи блока **играть звук** ❹ из раздела **Звуки**) и передвигает спрайт на 60 шагов ❺. Спрайт отскакивает от краев **Сцены** ❻, если это необходимо. Поскольку 60 шагов соответствуют одному квадрату в системе координат на рис. 2.16, каждый раз, когда вы нажимаете клавишу со стрелкой, спрайт перемещается на один квадрат.

Рис. 2.17. Скрипты для спрайта Игрок

Вы обратили внимание на то, что четыре скрипта, управляемых клавишами со стрелками, на рис. 2.17 практически одинаковые? В главе 4 вы узнаете, как избежать такого дублирования кода.

Протестируйте эту часть игры. У вас должно получаться передвигать игрока по всей **Сцене** при помощи клавиш со стрелками. Если все работает, мы перейдем к спрайту Золото, скрипт которого показан на рис. 2.18.

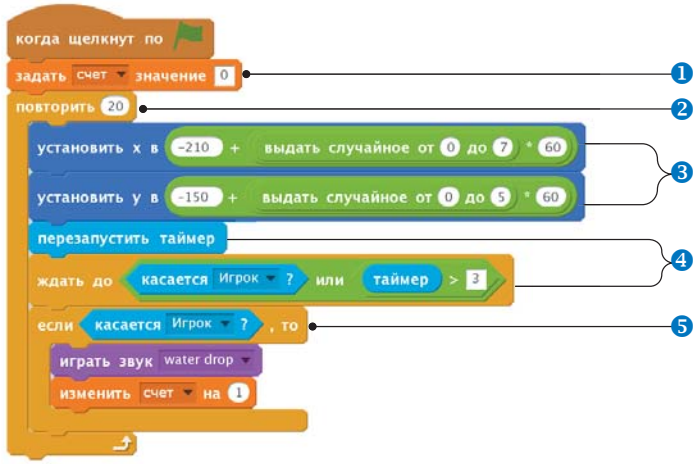


Рис. 2.18. Скрипт спрайта Золото

Как и спрайт Игрок, этот спрайт тоже начинает двигаться с момента нажатия на зеленый флажок. Он передвигает по **Сцене** мешок с золотом. Он также при помощи переменной под названием счет (которую

я создал для вас в разделе **Данные**) отслеживает, сколько мешков было собрано.

Переменные позволяют сохранять информацию, чтобы использовать ее позже в наших программах. Вы узнаете все о переменных в главе 5.

Поскольку игра только началась и пока нет никаких мешков, мы установим счет 0 ❶. Затем мы запустим цикл, который будет повторяться 20 раз ❷, чтобы показать игроку в общей сложности 20 мешков (или можете выбрать любое число). После каждого цикла мешок с золотом будет появляться в другом случайном месте ❸, давать игроку немного времени, чтобы схватить его ❹, и увеличивать счет, если игроку это удалось ❺. Нам нужно, чтобы мешок появлялся в одном из 48 квадратов сцены в случайном порядке. Как вы видите на рис. 2.16, координата *x* мешка может быть такой: –210, –150, –90, ..., 210. Эти цифры расположены на расстоянии 60 шагов друг от друга, так что вы сможете вычислить координату *x* начиная с –210:

$$x = -210 + (0 \times 60)$$
$$x = -210 + (1 \times 60)$$
$$x = -210 + (2 \times 60)$$
$$x = -210 + (3 \times 60)$$

Похожее выражение применимо и к координате *y*.

Мы можем устанавливать координату *x* мешка, сгенерировав случайное число от 0 до 7, умножая его на 60 и прибавляя к результату –210. Рис. 2.19 показывает детально алгоритм создания блока **установить x в** в нашем скрипте. Блок **установить y в** устроен похожим образом.

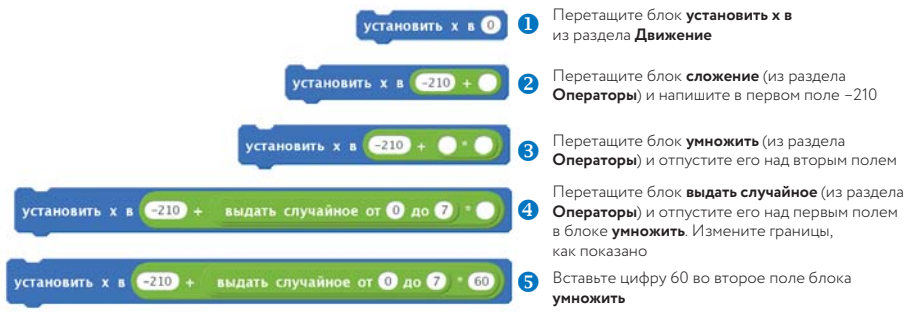


Рис. 2.19. Создаем блок **установить x в**, *как на рис. 2.18*

Появившись в случайном месте, мешок с золотом даст игроку три секунды на то, чтобы схватить его. (Вы можете изменить это время, сделав игру сложнее или проще.) Чтобы отслеживать время, скрипт первым делом обнуляет встроенный таймер. Затем он ждет, пока либо игрок схватит мешок, дотронувшись до него, либо выйдет время. Когда одно из этих условий выполняется, блок **ждать до** позволит скрипту перейти к блоку **если**. Подробное описание того, как создать блок **ждать до**, показано на рис. 2.20.

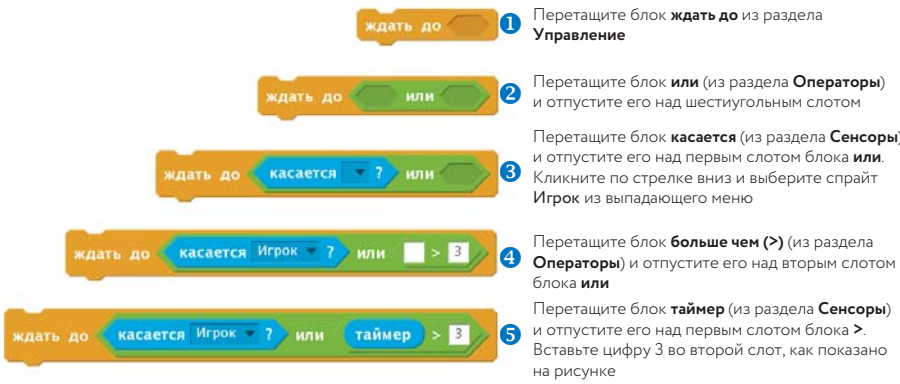


Рис. 2.20. Создание блока ***ждать до*** *в скрипте с рис. 2.18*

Блоки внутри блока ***если*** *будут выполнены, только если условие, указанное вами, верно. В главе 6 подробно объяснена работа этого блока, но пока вы достаточно знаете, чтобы использовать его и с его помощью добавлять в программу новые функции.*

Если игрок дотронется до мешка, команда внутри блока **если/то** запустится. В этом случае блок **играть звук** издаст звук water drop, а блок **изменить счет на 1** (из раздела **Данные**) добавит к счету одно очко.

Игра готова. Кликните по зеленому флажку, чтобы протестировать ее!

ТАЙМЕР SCRATCH

В среде Scratch есть таймер, который фиксирует, сколько времени прошло с момента начала работы. Когда вы открываете Scratch в своем браузере, таймер ставится на 0 и дальше считает время с точностью до десятых секунды, пока интерфейс открыт. Блок **таймер** (из раздела **Сенсоры**) содержит текущие показания таймера. Чекбокс рядом с блоком позволяет показать/скрыть таймер на **Сцене**. Блок **перезапустить таймер** обнуляет показания таймера, и отсчет времени начинается снова. Таймер продолжает работать, даже если проект остановлен.

Поймай яблоки

CatchApples_NoCode.sb2

Теперь рассмотрим игру «Поймай яблоки» (Catch Apples), показанную на рис. 2.21. В ней яблоки появляются в случайных горизонтальных позициях вверху **Сцены** в случайные моменты времени и падают вниз. Игрок должен передвигать тележку и ловить яблоки, прежде чем они коснутся земли. За каждое пойманное яблоко он получает 1 балл.

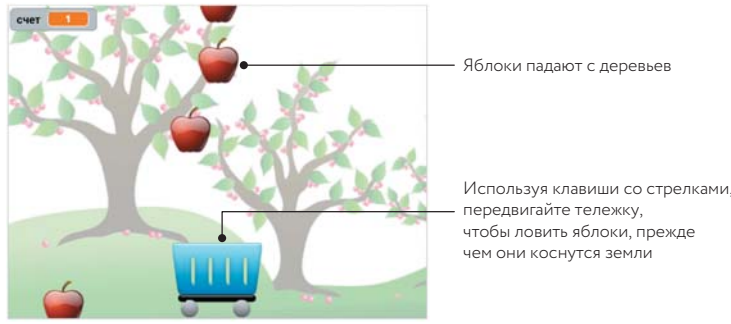


Рис. 2.21. Игра «Поймай яблоки»

Кажется, что в такой игре нужно много спрайтов с практически одинаковыми скриптами. Ведь яблок падает много. Однако в случае со средой Scratch 2 это не так. Благодаря функции *клонирования* вы можете легко создать много копий одного спрайта. В игре будет один спрайт-яблоко (Apple) и столько его клонов, сколько нам захочется.

Откройте файл *CatchApples_NoCode.sb2*, в котором содержится заготовка для нашей игры без скриптов. Чтобы было интереснее, в эту заготовку также была включена переменная под названием **счет** (она создана в разделе **Данные**), которую мы будем использовать, чтобы вести учет пойманных яблок. Но для начала вы сделаете скрипт для спрайта-тележки (Cart), как показано на рис. 2.22.

Рис. 2.22. Скрипт для спрайта-тележки

Когда зеленый флажок нажат, мы размещаем тележку внизу **Сцены** по центру. Затем скрипт постоянно проверяет состояние клавиш со стрелками и соответственно передвигает тележку. Методом проб и ошибок я выбрал число 30, а вы можете его изменить по собственному усмотрению.

Займемся клонированием. Для начала добавьте спрайту-яблоку скрипт с рис. 2.23. Он тоже начинает работать, когда нажат зеленый флажок.

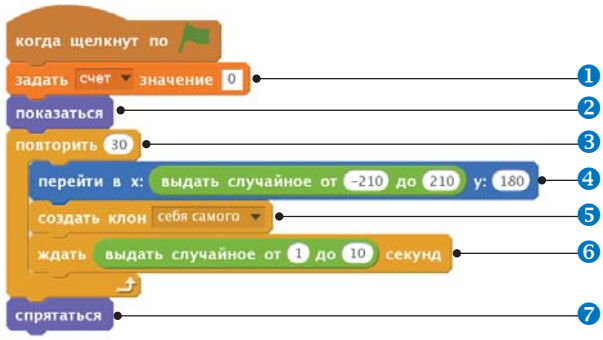


Рис. 2.23. Первый скрипт спрайта-яблока

Пока мы ни одного яблока не поймали, поэтому скрипт устанавливает переменную счет на 0 ❶. Затем он делает спрайт видимым при помощи блока **показаться** из раздела **Внешность** ❷. После чего он запускает блок **повторить**, который повторяется 30 раз, чтобы упало 30 яблок ❸.

При каждом повторе спрайт-яблоко приходит в случайную горизонтальную позицию в верхней части **Сцены** ❹. Затем он отдает блоку **создать клон** (из раздела **Управление**) команду клонировать себя самого ❺, ждет непродолжительный случайный отрезок времени ❻ и начинает следующий раунд блока **повторить**. После завершения 30 раундов блока **повторить** скрипт скрывает спрайт-яблоко при помощи блока **спрятаться** ❼ из раздела **Внешность**.

Если вы теперь запустите игру, кликнув по зеленому флажку, 30 яблок начнут случайным образом появляться вверху **Сцены** и оставаться там — потому что мы не объяснили им, что им нужно делать. И вот тут как раз появляется следующий скрипт для спрайта-яблока (рис. 2.24).

Благодаря блоку **когда я начинаю как клон** ❶ (из раздела **Управление**) каждый клон будет выполнять скрипт, показанный на этом рисунке. Каждое яблоко двигается вниз на 10 шагов ❷ и проверяет, было ли оно поймано или пропущено тележкой. Если клон определяет, что коснулся тележки ❸, это означает, что его поймали. Тогда он увеличивает счет и удаляет себя (потому что для него больше нет работы). Если клон падает ниже тележки ❹, игрок промахнулся. В этом случае клон проигрывает другой

звук, прежде чем удалить себя. Если клон не был ни пойман, ни пропущен, он еще падает и блок **всегда** запускается по второму кругу.

Рис. 2.24. Второй скрипт для яблока

Теперь, когда наши яблоки знают, как им нужно падать, игра готова! Протестируйте ее, кликнув по зеленому флажку. Если вы хотите поэкспериментировать, можете изменить время ожидания между клонированием разных яблок и скорость движения тележки. Нет ли у вас идей, как изменить уровень сложности игры?

И еще о клонированных спрайтах

Любой спрайт может создать копию себя или другого спрайта при помощи блока **создать клон**. (**Сцена** тоже может клонировать спрайты при помощи этого же блока.) Клонированный спрайт наследует состояние оригинала на момент клонирования: позицию и направление, костюм, статус видимости, цвет и размер пера, графические эффекты и т. д. (рис. 2.25).

Рис. 2.25. Клон наследует характеристики оригинала

Клоны также наследуют скрипты спрайта-оригинала, как показано на рис. 2.26. Здесь спрайт-оригинал создает два клона. Когда вы нажимаете **пробел**, все три спрайта (оригинал и два клона) поворачиваются на 15° вправо, потому что они выполняют скрипт **когда клавиша пробел нажата**.

Рис. 2.26. Клоны наследуют скрипты оригинала

Всегда будьте особенно внимательны, когда используете блок **создать клон** в скрипте, который не начинается с зеленого флажка. Иначе у вас может получиться больше спрайтов, чем вы планировали. Посмотрите внимательно на программу, показанную на рис. 2.27. Когда вы в первый раз нажмете **пробел**, появится клон, и после этого в программе будет два спрайта (оригинал и клон).

Рис. 2.27. Клонирование в ответ на нажатие клавиши

Если вы нажмете **пробел** во второй раз, в программе будет четыре спрайта. Почему? В ответ на нажатие клавиши спрайт-оригинал создаст клон, но первый клон тоже среагирует и создаст еще один (клон клона). Нажмите **пробел** в третий раз — и у вас будет восемь спрайтов. Количество клонов растет в геометрической прогрессии!

Вы можете исправить это, клонируя только те спрайты, скрипты которых начинаются с блока **когда щелкнут по**  . Они выполняются только спрайтом-оригиналом.

Итоги

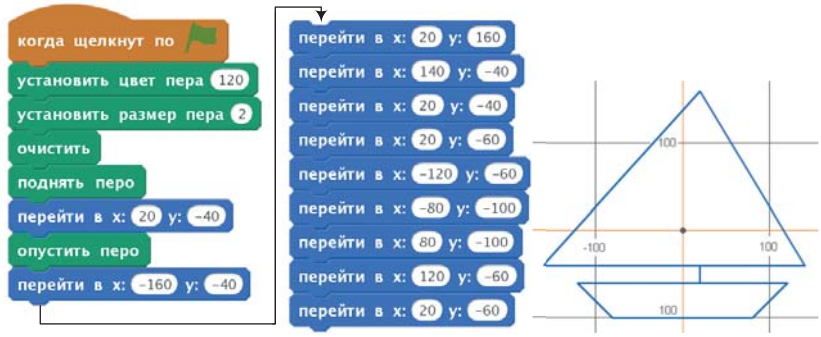
Из этой главы вы узнали, как перемещать спрайты на конкретные позиции на **Сцене** при помощи команд абсолютного движения. Затем вы использовали команды относительного движения, чтобы перемещать спрайты относительно их собственного местоположения и направления.

После этого вы создали несколько отличных рисунков при помощи команд **Пера**. Рисуя различные фигуры, вы открыли для себя возможности блока **повторить**, который позволяет создавать более короткие и более эффективные скрипты. Вы также узнали о команде **печать** и стали использовать ее вместе с блоком **повторить**, чтобы легко создавать сложные узоры.

В конце этой главы вы создали две игры и узнали о функции клонирования. В следующей главе вы будете пользоваться разделами **Внешность** и **Звук**, чтобы создавать еще более увлекательные программы.

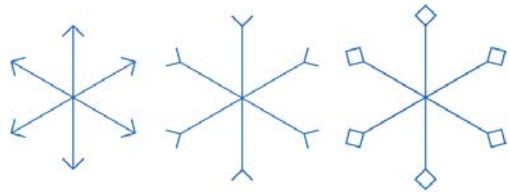
Задания

1. Объясните, как работает такой скрипт. Запишите координаты *x* и *y* для всех углов фигуры.



2. Напишите скрипт, чтобы соединить между собой точки в каждой из этих фигур и определить их окончательную форму:
   а) (30, 20), (80, 20), (80, 30), (90, 30), (90, 80), (80, 80), (80, 90), (30, 90), (30, 80), (20, 80), (20, 30), (30, 30), (30, 20);
   б) (–10, 10), (–30, 10), (–30, 70), (–70, 70), (–70, 30), (–60, 30), (–60, 60), (–40, 60), (–40, 10), (–90, 10), (–90, 90), (–10, 90), (–10, 10).

3. Напишите скрипт, чтобы нарисовать каждый из узоров, приведенных ниже.



издательство
МАНН, ИВАНОВ И ФЕРБЕР
Максимально полезные книги