6 Базовый двумерный платформер

- ✓ Непрерывное движение спрайтов.
- ✓ Анимация на базе листов спрайтов.
- ✓ Физические явления в двумерном пространстве (столкновения, гравитация).
- ✓ Управление камерой в сайд-скроллерах.

Продолжим знакомство с двумерной функциональностью Unity на примере еще одной игры. Фундаментальные концепции мы рассмотрели в предыдущей главе, теперь воспользуемся ими, чтобы построить более сложную игру, а именно: создадим основную функциональность двумерного *платформера*. Это распространенный тип двумерных игр, самым известным представителем которого является классическая игра *Super Mario Brothers*: персонаж, показываемый сбоку, бежит и прыгает по платформам на фоне перемещающейся сцены.

Результат, который мы хотим получить, показан на рис. 6.1.

Во время работы над этим проектом вы познакомитесь с такими концепциями, как перемещение игрока влево и вправо, воспроизведение анимации спрайтов и добавление возможности прыгать. Рассмотрим мы и вещи, характерные именно для платформеров, например односторонние и движущиеся платформы. Переход от шаблона к полноценной игре по большей части сводится к многократному повторению всех этих вещей.

Первым делом нам нужен новый двумерный проект. Мы уже создавали такие проекты в предыдущей главе. Выберите команду New в окне, которое появляется после запуска Unity, или команду New Project в меню File. В открывшемся окне выберите вариант 2D. Создайте для нового проекта две папки с именами Sprites и Scripts. В них мы будем складывать различные ресурсы. Еще можно настроить камеру, как мы это делали в предыдущей главе, но в данном случае достаточно уменьшить значение поля Size до 4. В этом проекте точной настройки не требуется, но для окончательной версии игры нужно скорректировать размер поля зрения камеры.

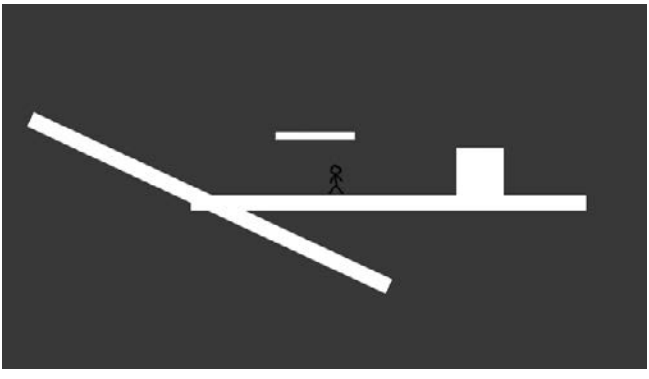

Рис. 6.1. Итоговый вид игры, которую мы создадим в этой главе

СОВЕТ Чтобы значок камеры в центре экрана не мешал работе, его нужно скрыть. В верхней части вкладки Scene откройте меню Gizmos и щелкните на значке Icon в строке Camera.

Сохраните пустую сцену, чтобы создать соответствующий набор ресурсов (в процессе работы не забывайте периодически щелкать на кнопке Save). Теперь давайте приступим к созданию графических ресурсов.

6.1. Создание графических ресурсов

Перед началом программирования функциональности нашего платформера в проект следует импортировать несколько изображений и вставить эти спрайты в сцену (надеюсь, вы помните, что в двумерных играх изображения называются *спрайтами*, а в трехмерных — *текстурами*). Так как мы просто знакомимся с принципом создания платформеров, в игре будет присутствовать управляемый игроком персонаж, перемещающийся по практически пустой сцене. Соответственно, достаточно пары спрайтов для платформ и для персонажа. Рассмотрим их по очереди, так как с этими простыми картинками связан ряд неочевидных моментов.

6.1.1. Стены и пол

Нам нужно пустое белое изображение. Пример проекта для этой главы содержит файл blank.png; скопируйте его оттуда в свой проект и поместите в папку Sprites. В параметрах импорта на панели Inspector убедитесь, что это действительно спрайт, а не текстура. Для двумерных проектов тип изображения выбирается автоматически, но проверить в любом случае имеет смысл.

По сути, мы будем строить геометрическую модель сцены, как это делалось в главе 4, просто на этот раз в режиме 2D. Поэтому основой послужат не сетки, а спрайты, но мы точно так же разместим в сцене пол и стены, формируя пространство, по которому будет перемещаться персонаж.

Чтобы получить пол, перетащите в сцену спрайт `blank`, как показано на рис. 6.2. Расположите его в точке с координатами `0.15`, `-1.27`, `0`, а в поля масштаба введите

значения `50`, `2`, `1`. Присвойте спрайту имя Floor. Перетащите в сцену еще один пустой спрайт, введите в поля масштаба значения `6`, `6`, `1` и поставьте на пол справа, указав координаты `2`, `-0.63`, `0`. Присвойте ему имя Block.

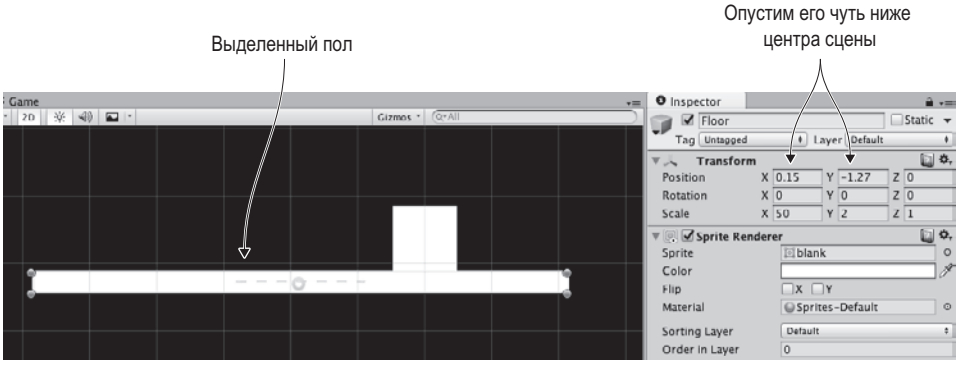

Рис. 6.2. Выбираем положение пола

Как видите, все очень просто. Пол и блок уже на месте, теперь нужно добавить в сцену персонаж.

6.1.2. Импорт листов спрайтов

Теперь нам требуется спрайт, изображающий игрока, поэтому скопируйте из папки с примером проекта файл stickman.png. На этот раз это не одно изображение, а объединенный набор спрайтов. Он показан на рис. 6.3 и представляет собой кадры двух вариантов анимации: бездействия и цикла ходьбы. Вдаваться в детали процесса анимации мы не будем, скажем только, что термины *бездействие* (idle) и *цикл* (cycle) повсеместно используются разработчиками игр. Бездействие относится к небольшим движениям стоящей на месте фигурки, а цикл представляет собой непрерывно повторяющуюся анимацию.

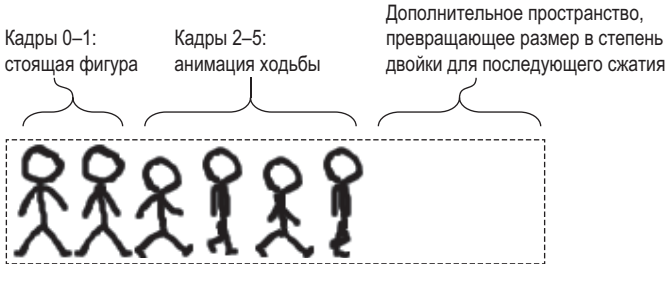

Рис. 6.3. Лист спрайтов Stickman — 6 кадров подряд

В предыдущей главе упоминалось, что изображение может представлять собой набор соединенных друг с другом спрайтов. В этом случае оно называется *листом спрайтов* (sprite sheet) и состоит из кадров анимации. В Unity листы спрайтов фигурируют на

вкладке Project как единый ресурс, но щелчок на расположенной сбоку стрелке позволяет увидеть все входящие в их состав изображения, как показано на рис. 6.4.

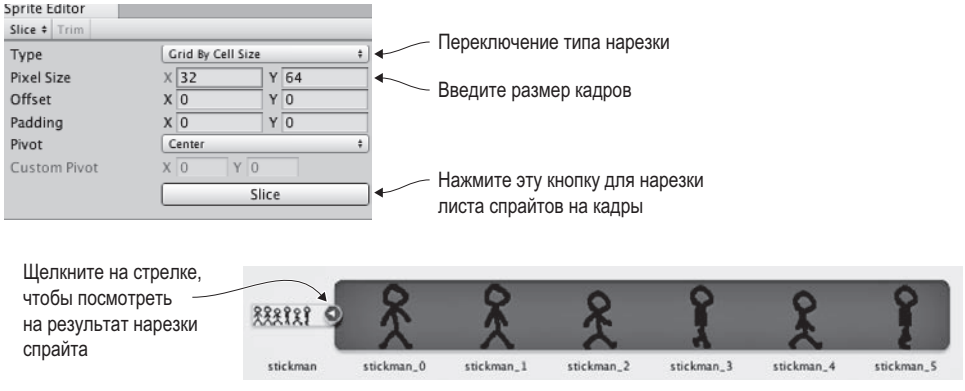

Рис. 6.4. Нарезка листа спрайтов на кадры

Перетащите изображение stickman.png в папку Sprites, чтобы выполнить его импорт, и внесите изменения в настройки импорта на панели Inspector. В меню Sprite Mode выберите вариант Multiple, а затем щелкните на кнопке Sprite Editor, чтобы открыть окно редактора. Откройте меню Slice в верхнем левом углу редактора и выберите в раскрывающемся списке Type вариант Grid By Cell Size (как показано на рис. 6.4). В поля Pixel Size введите значения `32` и `64` соответственно (это размер одного кадра в листе спрайтов) и щелкните на кнопке Slice. Лист распадется на отдельные кадры. Щелкните на кнопке Apply, чтобы сохранить сделанные изменения, и закройте редактор кадров.

Лист спрайтов превратился в набор кадров. Щелкните на стрелке, чтобы их увидеть. Перетащите один спрайт (например, самый первый) в сцену, поставьте на пол в центре и присвойте ему имя Player. В нашей игре появился персонаж!

6.2. Смещение персонажа вправо и влево

Теперь, когда в сцене появилась вся необходимая графика, можно приступить к программированию перемещений персонажа. Первым делом следует добавить персонажу пару компонентов. Как упоминалось в предыдущих главах, сымитировать объект, подчиняющийся законам физики, можно только при наличии у этого объекта компонента `Rigidbody`. Наш персонаж должен подчиняться законам физики (например, испытывать действие силы тяжести). Кроме того, потребуется компонент `Collider`, определяющий границы объекта при распознавании столкновений. Важно понимать разницу между этими компонентами: `Collider` определяет форму, на которую будут действовать законы физики, а `Rigidbody` указывает симулятору физики, на какие объекты он должен действовать.

ПРИМЕЧАНИЕ Эти компоненты существуют по отдельности (хотя они тесно связаны), потому что многие объекты, для которых не требуется имитация физической среды, *принимают участие* в столкновениях с объектами, *которые должны* подчиняться законам физики.

Есть еще один тонкий момент, о котором не следует забывать. В Unity для двумерных игр существует отдельная система имитации физической среды. Соответственно, в этой главе будут использоваться компоненты из раздела Physics 2D, а не из раздела Physics.

Выделите игрока и щелкните на кнопке Add Component на панели Inspector. Выберите в появившемся меню команду Physics 2D > Rigidbody 2D, как показано на рис. 6.5. Еще раз щелкните на кнопке Add Component и выберите уже команду Physics 2D > Box Collider 2D. Теперь слегка отредактируем компонент `Rigidbody`. Выберите в меню Collision Detection вариант Continuous. Затем установите флажок Freeze Rotation Z (обычно симулятор физической среды пытается поворачивать объекты в процессе перемещения, но персонаж нашей игры ведет себя немного по-другому), а в поле Gravity Scale введите значение 0 (позднее мы вернем исходное значение, но пока сила тяжести нам не требуется).

Осталось написать сценарий, который будет контролировать перемещения персонажа.

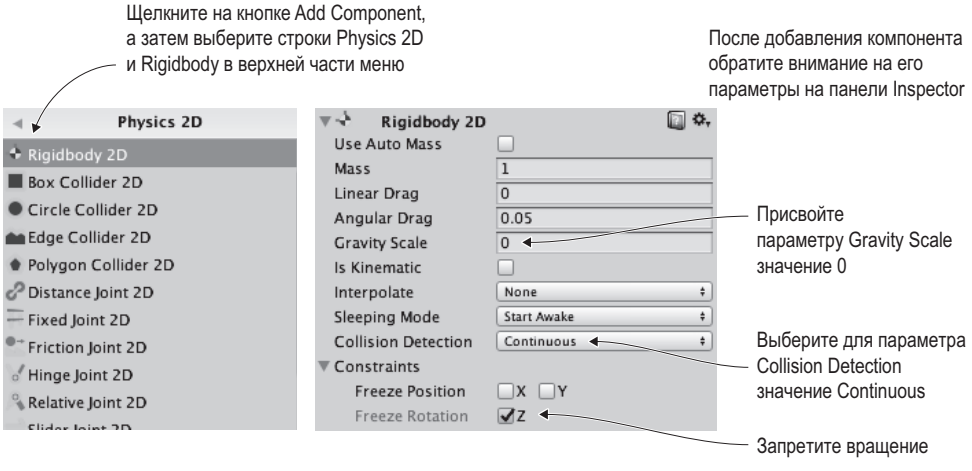

Рис. 6.5. Добавление и настройка компонента Rigidbody 2D

6.2.1. Элементы управления клавиатурой

Первым делом заставим персонажа перемещаться из стороны в сторону. Вертикальные перемещения также играют в платформерах важную роль, но ими мы займемся позднее. Создайте в папке Scripts сценарий с именем PlatformerPlayer и перетащите его на объект `Player`.

Добавьте в него код следующего листинга.

Листинг 6.1. Сценарий PlatformerPlayer для перемещений при помощи клавиш со стрелками

```
using UnityEngine;
using System.Collections;

public class PlatformerPlayer : MonoBehaviour {
  public float speed = 250.0f;
```

```
  private Rigidbody2D _body;

  void Start() {
    _body = GetComponent<Rigidbody2D>();
  }

  void Update() {
    float deltaX = Input.GetAxis("Horizontal") * speed * Time.deltaTime;
    Vector2 movement = new Vector2(deltaX, _body.velocity.y);
    _body.velocity = movement;
  }
}
```

Нужно, чтобы к объекту GameObject был прикреплен этот второй компонент.

Задаем только горизонтальное движение; сохраняем заданное вертикальное смещение.

Нажмите кнопку Play и посмотрите, как персонаж реагирует на кнопки со стрелками. Основное отличие от кода движения из предыдущих глав состоит в том, что новый код действует на компонент `Rigidbody2D`, а не на контроллер персонажа. Компонент `Character Controller` используется в трехмерном режиме, а для двумерных игр применяется компонент `Rigidbody`. Обратите внимание: движение применяется к переменной `velocity`, то есть к скорости этого компонента, а не к его положению.

СОВЕТ По умолчанию при нажатии кнопок со стрелками в Unity объект не сразу начинает двигаться с указанной скоростью, а постепенно разгоняется до нее. Но это слишком вялое движение для платформера. Для более быстрого управления увеличьте значения параметров Sensitivity и Gravity для горизонтального ввода до 6. Доступ к этим настройкам открывается командой Project Settings > Input из меню Edit. Они находятся в свитке Horizontal на панели Inspector.

Программирование движения по горизонтали практически завершено! Осталось научиться распознавать столкновения.

6.2.2. Столкновения со стенами

В настоящий момент персонаж спокойно проходит сквозь блок, ведь ни у пола, ни у блока нет коллайдера и сквозному движения ничто не мешает. Чтобы устранить эту недоработку, добавьте к объектам `Floor` и `Block` компонент `Box Collider 2D`. По очереди выделите каждый объект, щелкните на кнопке Add Component на панели Inspector и выберите в разделе Physics 2D строку Box Collider 2D.

Готово! Нажмите кнопку Play и убедитесь, что теперь персонаж останавливается перед блоком. Как и в случае перемещения игрока в главе 2, при попытках непосредственно редактировать положение перемещаемого объекта распознавание столкновений невозможно. Встроенный в Unity модуль распознавания столкновений работает, когда мы начинаем двигать добавленные объекту компоненты имитации физической среды. Другими словами, при изменениях переменной `Transform.position` распознавание столкновений игнорируется, поэтому в сценарии движения мы работаем с переменной `Rigidbody2D.velocity`.

К менее примитивным объектам добавить коллайдер чуть сложнее, но не намного. Даже если фигура не похожа на прямоугольник, можно взять прямоугольный коллайдер и окружить им форму, изображающую препятствие. Существуют и другие варианты

коллайдеров, в том числе и в виде определяемых пользователем наборов многоугольников. Принцип их применения к объектам сложной формы показан на рис. 6.6.

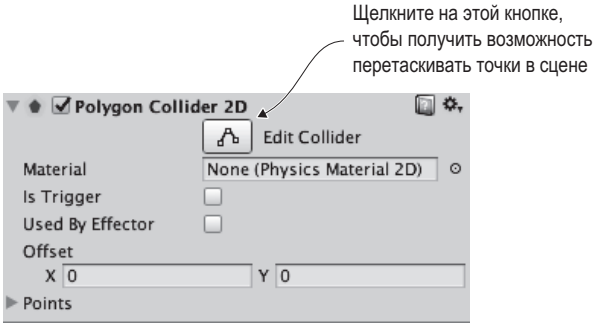

Рис. 6.6. Переход в режим редактирования формы полигонального коллайдера с помощью кнопки Edit Collider

С распознаванием столкновений мы разобрались, теперь давайте заставим персонажа двигаться в процессе перемещения.

6.3. Анимация спрайтов

После импорта изображение stickman.png было превращено в набор кадров для последующей анимации. Пришло время *запустить* эту анимацию, чтобы персонаж перестал скользить, а начал бегать в разные стороны.

6.3.1. Система анимации Mecanim

В главе 4 я уже упоминал, что система анимации в Unity называется *Mecanim*. Она позволяет визуально настроить для персонажа комплексную сеть анимационных клипов и обойтись минимумом кода при управлении этими клипами. Чаще всего ее применяют для трехмерных персонажей (эта тема будет детально рассмотрена в следующих главах), но работает она и с двумерными персонажами.

В основе системы анимации лежат два вида ресурсов: анима*ционные* клипы и анима*торные* контроллеры. Клипы представляют собой отдельные циклы анимации, в то время как контроллер — это сеть, определяющая, когда будет воспроизводиться каждый клип. Это диаграмма *конечного автомата* (state machine), а состояния на ней соответствуют различным анимационным клипам. Контроллер перемещается между состояниями, реагируя на окружающую среду, и в каждом состоянии воспроизводится свой клип.

Оба ресурса — анимационный клип и аниматорные контроллер — Unity создает автоматически в момент перетаскивания двумерной анимации в сцену. Разверните кадры нашего спрайта, как показано на рис. 6.7, выделите кадры 0–1, перетащите их в сцену и укажите в открывшемся окне имя stickman_idle.

На вкладке с ресурсами появился клип stickman_idle и контроллер `stickman_0`; переименуйте контроллер в stickman. Вы только что создали анимацию для стоящего на месте