2 Создание демонстрационного 3D-проекта

В этой главе

- ✓ Знакомство с трехмерным координатным пространством.
- ✓ Размещение в сцене игрока.
- ✓ Создание сценария перемещения объектов.
- ✓ Реализация элементов управления персонажем в игре от первого лица.

Глава 1 завершилась описанием традиционного способа знакомства с новыми средствами программирования — написанием программы «Hello World!». Пришло время погрузиться в более сложный Unity-проект с элементами интерактивности и графикой. Вы поместите в сцену объекты и напишете код, позволяющий игроку перемещаться по ней. По сути, это будет аналог игры Doom без монстров (примерно как на рис. 2.1). Визуальный редактор Unity позволяет новым пользователям сразу приступить к сборке трехмерного прототипа без предварительного написания шаблонного кода (для таких вещей, как инициализация 3D-вида или установка цикла рендеринга).

Хотя на данном этапе высок соблазн немедленно заняться созданием сцены, особенно с учетом крайней простоты проекта, имеет смысл немного притормозить и продумать порядок своих действий, так как вы новичок.

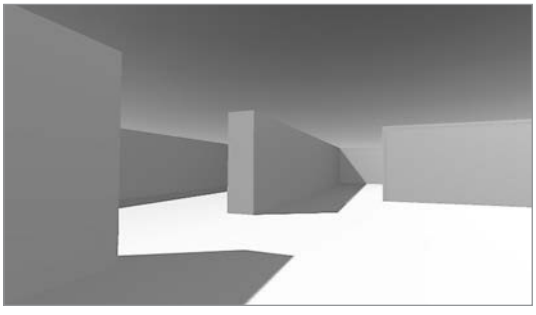


Рис. 2.1. Снимок экрана трехмерной демоверсии (по сути, это Doom без монстров)

ПРИМЕЧАНИЕ Помните, что проект для каждой главы можно скачать с сайта книги http://mng.bz/VBY5. Сначала откройте проект в Unity, а затем главную сцену (часто называемую просто Сценой, или Scene) для запуска и проверки. Пока вы учитесь, я рекомендую вам самостоятельно набирать весь код и использовать загруженный пример исключительно для справки.

2.1. ПОДГОТОВКА

Инструмент Unity дает новичкам возможность сразу приступить к работе, но перед тем, как заняться созданием сцены, оговорим пару моментов. Даже при работе с таким гибким инструментом, как Unity, следует четко представлять себе, что именно вы хотите получить в результате своих действий. Кроме того, необходимо понимать, как функционирует трехмерная система координат, в противном случае вы запутаетесь, как только попытаетесь расположить объект на сцене.

2.1.1. Планирование проекта

Перед тем как приступить к программированию, всегда нужно остановиться и спросить себя: «Что я собираюсь создать?». Проектирование игр — весьма обширная тема, которой посвящено множество книг. К счастью, для наших целей достаточно мысленно представлять структуру будущей сцены. Первые проекты будут несложными, чтобы лишние детали не мешали вам изучать принципы программирования, а задумываться о разработке более высокоуровневого проекта можно (и нужно!) только после того, как вы освоите основы разработки игр.

Вашим первым проектом станет создание сцены из типичного шутера от первого лица (First-person Shooter, FPS). Мы создадим комнату, по которой можно перемещаться, при этом игроки будут наблюдать окружающий мир с точки зрения их игрового персонажа и будут управлять этим персонажем посредством

мыши и клавиатуры. Все интересные, но сложные элементы готовой игры мы пока отбросим, чтобы сконцентрироваться на основной задаче — перемещениях в трехмерном пространстве. Рисунок 2.2 иллюстрирует план проекта — перечень действий, придуманный мной.

1. Разработка комнаты: создание пола, внешних и внутренних стен.
2. Размещение источников света и камеры.
3. Создание игрока как объекта (в том числе и фиксация камеры к его «голове»).
4. Написание скриптов движения: повороты при помощи мыши и перемещение при помощи клавиатуры.

Рис. 2.2. Сценарий трехмерного демонстрационного ролика

Пусть столь внушительный список действий вас не пугает! Кажется, что в этом разделе вам предстоит сделать много шагов, но Unity упрощает их. Разделы, посвященные скриптам перемещения, так велики только потому, что мы детально рассматриваем каждую строчку кода, чтобы полностью понять принципы программирования.

Мы начинаем с игры от первого лица, чтобы снизить требования к художественному оформлению, — в играх от первого лица «себя» игрок не видит, поэтому вполне допустимо придать персонажу форму цилиндра, на верхнем основании которого закреплена камера! Теперь осталось понять, как функционируют трехмерные координаты, чтобы вы смогли легко вставлять объекты в сцены в визуальном редакторе.

2.1.2. Трехмерная система координат

Сформулированный нами для начала простой план включает игровое пространство, средства обзора, элементы управления. Для их реализации требуется понимание того, как в трехмерных симуляциях задается положение и перемещение объектов. Те, кто раньше никогда не сталкивался с 3D-графикой, вполне могут этого не знать.

Все сводится к числам, указывающим положение точки в пространстве. Сопоставление этих чисел с пространством происходит через оси системы координат. На школьных уроках математики вы наверняка видели и использовали оси *X* и *Y* (рис. 2.3) для присваивания координат различным точкам на листе бумаги. Это так называемая *декартова система координат*.

Две оси дают вам двумерные координаты, все точки которой лежат в одной плоскости. Трехмерное пространство задается уже тремя координатными осями. Так как ось *X* располагается на странице горизонтально, а ось *Y* — вертикально, третья ось должна как бы «протыкать» страницу, располагаясь перпендикулярно осям *X* и *Y*. Рисунок 2.4 демонстрирует оси *X*, *Y* и *Z* для трехмерного координатного пространства. У всех элементов, расположенных в сцене (игрок, стена и т. п.), будут координаты *X*, *Y* и *Z*.

На вкладке Scene в Unity вы видите значок трех осей. На панели Inspector можно задать три числа, необходимых для позиционирования объекта. Координаты в трехмерном пространстве вы будете использовать не только при написании кода, определяющего местоположение объектов, но и для указания смещений как значений сдвига вдоль каждой из осей.

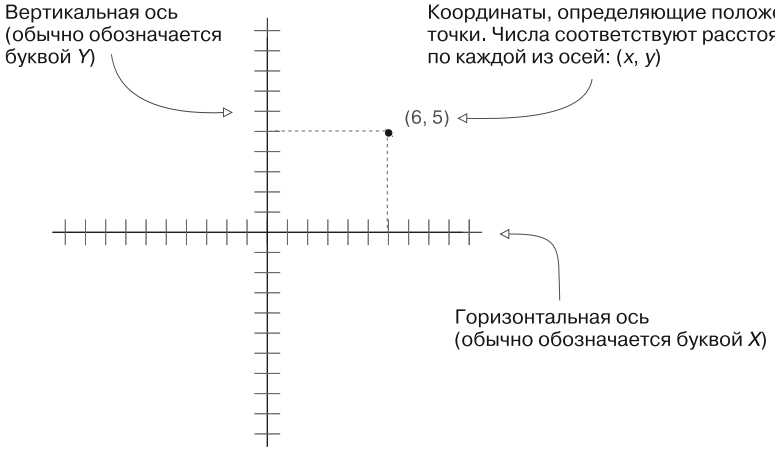


Рис. 2.3. Координаты по осям *X* и *Y* определяют положение точки в двумерном пространстве

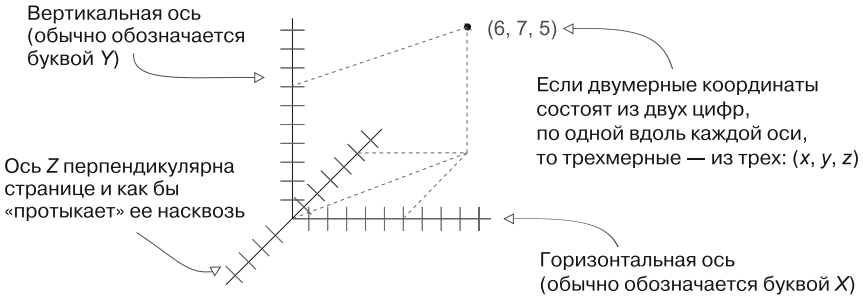
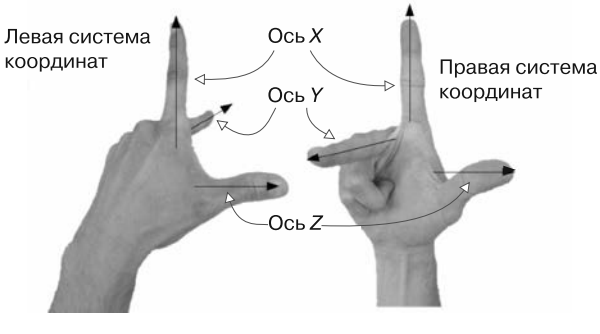


Рис. 2.4. Координаты по осям *X*, *Y* и *Z* определяют точку в трехмерном пространстве

ПРАВАЯ И ЛЕВАЯ СИСТЕМЫ КООРДИНАТ

Положительное и отрицательное направления каждой оси выбираются произвольным образом, в обоих случаях система координат прекрасно работает. Главное, используя инструмент для обработки 3D-графики (инструменты анимации, разработки и т. п.), всегда выбирать одну и ту же систему координат.

Впрочем, практически всегда ось *X* указывает вправо, а ось *Y* — вверх; разница между инструментами в основном состоит в том, что где-то ось *Z* «выходит» из страницы, а где-то «входит» в нее. Эти два варианта называют «правой» и «левой» системами координат. Как показано на рисунке, если большой палец расположить вдоль оси *X*, а указательный — вдоль оси *Y*, средний палец задаст направление оси *Z*.



У левой и правой рук ось *Z* ориентирована в разных направлениях

В Unity, как и во многих других приложениях для работы с компьютерной графикой, используется левосторонняя система координат. Однако существует множество инструментов, в которых применяется и правосторонняя (например, OpenGL). Помните об этом, чтобы не растеряться, столкнувшись с другими направлениями координатных осей.

Теперь, когда у вас есть не только план действий, но и представление о том, как с помощью координат задать положение объекта в трехмерном пространстве, приступим к работе над сценой.

2.2. НАЧАЛО ПРОЕКТА: РАЗМЕЩЕНИЕ ОБЪЕКТОВ

Итак, начнем с создания объектов и размещения их в сцене. Первыми будут статичные декорации — пол и стены. Затем выберем место для источников света и камеры. Последним создадим игрока — это будет объект, к которому вы добавите скрипты, перемещающие его по сцене. Рисунок 2.5 демонстрирует вид редактора после завершения работы.

Давайте создадим наш новый проект (мы рассматривали этот процесс в главе 1). Выберите New в Unity Hub (или File ▸ New Project в редакторе) и затем в появившемся окне введите имя проекта. Заполнение сцены мы начнем с создания наиболее очевидных объектов.

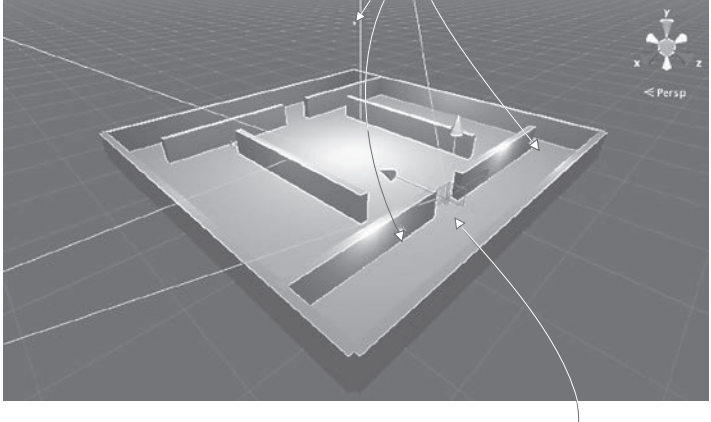


Рис. 2.5. Сцена в редакторе с полом, стенами, источниками света, камерой и игроком

2.2.1. Декорации: пол, внешние и внутренние стены

В расположенном в верхней части экрана меню GameObject наведите указатель мыши на строчку 3D Object, чтобы открыть выпадающее меню. Выберите в нем вариант Cube (позднее вы поработаете и с другими фигурами, такими как Sphere

и Capsule). Отредактируйте имя, положение и масштаб появившегося в сцене куба таким образом, чтобы получить пол; значения, которые следует присвоить параметрам этого объекта на панели Inspector, показаны на рис. 2.6 (для превращения куба в пол его нужно растянуть).

ПРИМЕЧАНИЕ Единицы измерения положения могут выбираться произвольным образом, главное, чтобы они были одинаковы для всех элементов сцены. Чаще всего в качестве единицы измерения фигурирует метр, и я обычно использую именно его, но бывают случаи, когда я выбираю футы. Мне даже доводилось видеть людей, которые считали, что размеры в сцене измеряются в дюймах!

Повторите описанную последовательность для создания внешних стен комнаты. Можно каждый раз задействовать новый куб или копировать существующий объект, используя стандартные сочетания клавиш. Двигайте, поворачивайте и изменяйте размеры стен, чтобы получить показанный на рис. 2.5 периметр. Экспериментируйте с различными значениями (например, 1, 4, 50 для полей Scale) или воспользуйтесь инструментами преобразований, с которыми вы познакомились в разделе 1.2.2 (напоминаю, что перемещения и повороты в трехмерном пространстве называют *преобразованиями*).

СОВЕТ Не забудьте про средства навигации из главы 1, позволяющие рассматривать сцену под разными углами или менять ее масштаб для обзора с высоты птичьего полета. При этом нажатие клавиши F вернет вас к просмотру выделенного в данный момент объекта.

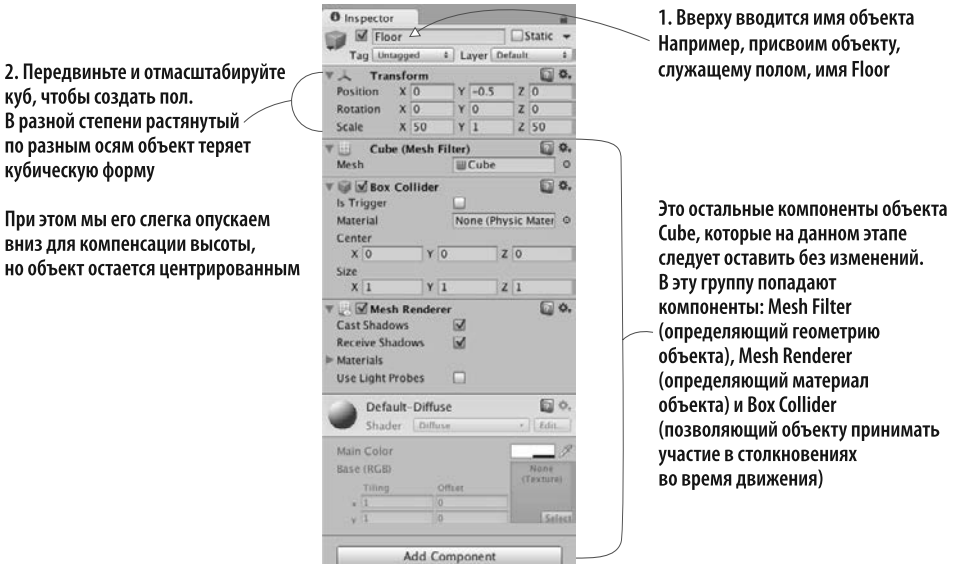


Рис. 2.6. Панель Inspector для пола

Когда внешние стены будут установлены, создайте внутренние (располагайте их как угодно) для перемещения между ними; идея состоит в том, чтобы создать коридоры и препятствия для обхода, как только вы напишете код для передвижения. Точные значения Transform будут зависеть от того, как вы повернете и отмасштабируете кубы, а также от того, как объекты связаны между собой во вкладке Hierarchy. Если вам нужен пример, из которого можно скопировать рабочие значения, загрузите образец проекта и посмотрите на стены в нем.

СОВЕТ Связи между объектами устанавливаются простым перетаскиванием объектов друг на друга на вкладке Hierarchy. Объект, к которому присоединены другие объекты, называется родительским (parent); объекты, присоединенные к другим объектам, называются дочерними (children). Перемещение (поворот или масштабирование) родительского объекта сопровождается аналогичным преобразованием всех его потомков.

ОПРЕДЕЛЕНИЕ *Корневой объект* (тесно связанный с понятиями родительских и дочерних объектов) — это объект в основании иерархии, который сам не имеет родительского объекта. Таким образом, все корневые объекты являются родителями, но не все родители являются корневыми объектами.

Вы также можете создавать пустые игровые объекты, которые будут использоваться для организации сцены. В меню GameObject выберите Create Empty. Связывая видимые объекты с корневым объектом, можно свернуть их список иерархии. К примеру, на рис. 2.7 демонстрируется ситуация, когда все стены являются потомками пустого корневого объекта (с названием Building). Содержимое вкладки Hierarchy в этом случае имеет упорядоченный вид.

ПРЕДУПРЕЖДЕНИЕ Прежде чем привязывать к нему какие-либо дочерние объекты, обязательно сбросьте параметры преобразования (Position (расположение) и Rotation (поворот) на 0, 0, 0, а Scale (масштабирование) — на 1, 1, 1) пустого корневого объекта, чтобы избежать любых проблем с расположением дочерних объектов.

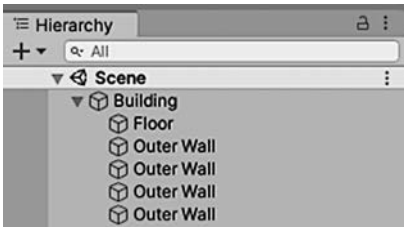


Рис. 2.7. Панель Hierarchy, показывающая, что стены и пол являются потомками пустого объекта

Не забудьте сохранить измененную сцену, если вы этого еще не сделали. Теперь в сцене появилась комната, но нам еще нужно разместить в ней источники света. Поэтому давайте займемся этим вопросом.

ЧТО ТАКОЕ GAMEOBJECT

Все объекты сцены представляют собой экземпляры класса `GameObject` аналогично тому, как все компоненты сценариев наследуются от класса `MonoBehaviour`. Данный факт становится более наглядным, если пустому объекту присвоить имя `GameObject`. Впрочем, даже если этот объект будет называться Floor, Camera или Player, суть дела не изменится.

На самом деле `GameObject` представляет собой всего лишь контейнер для набора компонентов. Его основным назначением является обеспечение некоего объекта, к которому можно присоединять класс `MonoBehaviour`. Как все это будет выглядеть в сцене, зависит от добавленных к объекту `GameObject` компонентов. К примеру, куб получается добавлением компонента `Cube`, сфера — добавлением компонента `Sphere`, и т. п.

2.2.2. Источники света и камеры

Как правило, трехмерные сцены освещаются направленным источником света, к которому добавляется набор точечных осветителей. Начать имеет смысл с направленного источника. Он может по умолчанию присутствовать в сцене, но если его нет, наведите указатель мыши на строку Light в меню GameObject и выберите вариант Directional Light.

ТИПЫ ОСВЕТИТЕЛЕЙ

Вы можете создать различные типы источников света, которые освещают область. Три основных типа: точечный источник, прожектор и направленный источник.

В точечном источнике (point light) все лучи начинаются в одной точке и распространяются во всех направлениях, как лампочка. Яркость света увеличивается по мере приближения к источнику за счет концентрации лучей.

В прожекторе (spot light) лучи также исходят из одной точки, но распространяются в пределах ограниченного конуса. В текущем проекте мы с прожекторами работать не будем, но осветители данного типа повсеместно используются для подсвечивания отдельных частей уровня.

В направленном источнике света (directional light) лучи распространяются равномерно и параллельно друг другу, одинаково освещая все элементы сцены. Это аналог солнца.

Испускаемый направленным осветителем луч не зависит от местоположения источника, значение имеет только его направление, поэтому его можно поместить

в произвольную точку сцены. Я рекомендую установить направленный источник света над комнатой, чтобы он выглядел как солнце и не мешал вам при работе с остальными фрагментами сцены. Поверните источник света и посмотрите, как это повлияет на освещенность комнаты; для получения нужного эффекта попробуйте слегка повернуть его относительно осей *X* и *Y*.

На панели Inspector вы найдете параметр Intensity (рис. 2.8), который управляет яркостью света. Если бы данный направленный осветитель был в сцене единственным, его яркость имело бы смысл увеличить, но так как мы добавим несколько точечных источников света, его можно оставить тусклым. Например, присвойте параметру Intensity значение `0.6`. Этот свет также должен иметь легкий желтый оттенок, как у солнца, в то время как другой свет будет белым.

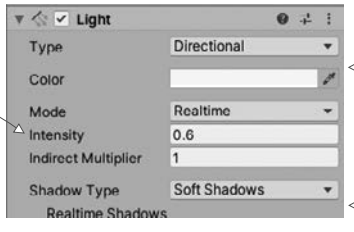


Рис. 2.8. Настройки направленного источника света на панели Inspector

Теперь по уже знакомой вам схеме создайте несколько точечных осветителей, поместив их в темные места комнаты, чтобы все стены были освещены. Источников не должно быть слишком много, иначе пострадает производительность. Достаточно расположить по одному в каждом верхнем углу стены и один высоко над сценой (например, со значением 18 координаты *Y*). Это придаст освещению комнаты некое разнообразие.

Обратите внимание, что у точечных источников света на панели Inspector есть дополнительный параметр `Range` (рис. 2.9). Он управляет дальностью распространения лучей. Если направленный источник света равномерно освещает всю сцену, то яркость точечного уменьшается по мере удаления от него. Поэтому для стоящего на полу источника этот параметр может быть равен 18, а для размещенного высоко необходимо увеличить до 40, иначе он не сможет осветить всю комнату. Установите интенсивность на `0.8` для света у пола, в то время как у находящегося высоко — `0.4`, чтобы они давали дополнительный свет, который заполнит пространство.

Чтобы игрок мог наблюдать за происходящим, требуется еще один тип объекта — камера. Впрочем, пустая сцена содержит основную камеру, которой мы можем воспользоваться. Если вам когда-нибудь потребуется создать дополнительные камеры (например, для разделения экрана в многопользовательских играх),

выберите команду Camera в меню GameObject. Расположим нашу камеру над игроком, чтобы наблюдать сцену его глазами.

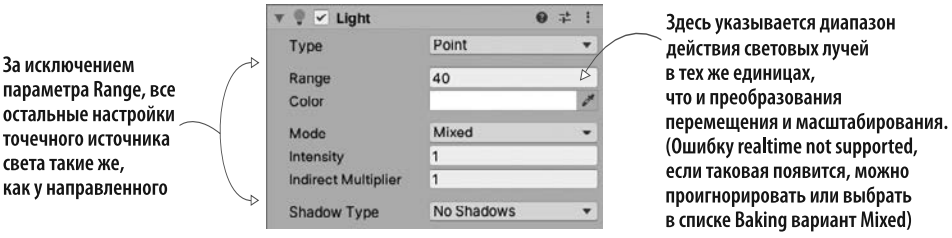


Рис. 2.9. Настройки точечного источника света на панели Inspector

2.2.3. Коллайдер и точка обзора игрока

В этом проекте игрока будет представлять обычный примитив. В меню GameObject (наведите курсор на строку 3D Object) выберите вариант Capsule. Появится цилиндрическая фигура — это и есть наш игрок. Сместите объект вверх, сделав его координату *Y* равной 1,1 (половина высоты объекта плюс еще немного, чтобы избежать перекрывания с полом). Теперь наш игрок может произвольным образом перемещаться вдоль осей *X* и *Z* при условии, что он остается внутри комнаты и не касается стен. Присвойте объекту имя Player.

На панели Inspector вы увидите, что созданному нами объекту назначен капсульный коллайдер. Это очевидный вариант для объекта Capsule, точно так же как объект Cube по умолчанию обладает коллайдером Box. Но так как наша капсула — игрок, ее компоненты должны слегка отличаться от компонентов большинства объектов. Удалите капсульный коллайдер, щелкнув на значке меню справа от имени компонента (рис. 2.10), а затем на Remove Component. Коллайдер — это зеленая сетка вокруг объекта, поэтому после удаления этого компонента она исчезнет.

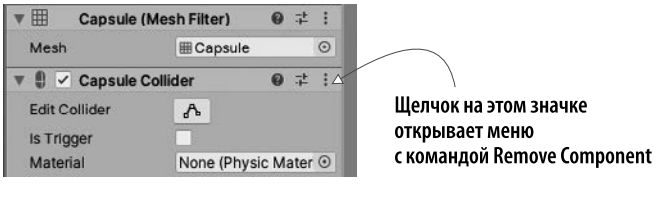


Рис. 2.10. Удаление компонента на панели Inspector

Вместо капсульного коллайдера мы назначим объекту *контроллер персонажа* (Character controller). В нижней части панели Inspector вы найдете кнопку Add Component. Щелчок на ней открывает меню с перечнем типов доступных

компонентов. В разделе Physics и находится нужная нам строка Character Controller. Именно этот компонент позволит объекту вести себя как персонаж.

Для завершения настройки игрока осталось сделать всего один шаг — подключить к нему камеру. Как уже упоминалось в подразделе 2.2.1, вы можете перетаскивать объекты и класть их друг на друга на вкладке Hierarchy. Проделайте эту операцию, перетащив камеру на капсулу, чтобы присоединить ее к игроку. Затем расположите ее таким образом, чтобы она располагалась на уровне глаз игрока (я предлагаю указать позицию со значениями `0`, `0.5`, `0`). При необходимости сбросьте параметр вращения камеры на `0`, `0`, `0` (если вы вращали капсулу, он будет выключен).

Итак, в сцене присутствуют все необходимые объекты. Осталось написать код перемещения игрока.

2.3. ДВИГАЕМ ОБЪЕКТЫ: СЦЕНАРИЙ, АКТИВИРУЮЩИЙ ПРЕОБРАЗОВАНИЯ

Чтобы заставить игрока перемещаться по сцене, нам потребуются сценарии движения, которые будут привязаны к игроку. Напоминаю, что компонентами называются модульные фрагменты функциональности, добавляемые к объектам, а скрипты — это разновидность компонентов. Именно они будут реагировать на действия с клавиатурой и мышью, но для начала мы заставим игрока поворачиваться на месте. Так вы наглядно увидите, как применять преобразования в коде. Надеюсь, вы помните, что преобразования бывают трех видов: перенос, поворот и изменение размера; вращение объекта на месте соответствует преобразованию поворота. Но вы должны знать об этой задаче еще кое-что, кроме того, что она «сводится к изменению ориентации объекта».

2.3.1. Схема программирования движения

Анимация (например, вращение) представляет собой смещение объекта в каждом кадре на небольшое расстояние, а затем последующее многократное воспроизведение всех кадров. Само по себе преобразование происходит мгновенно, в отличие от видимого движения, растянутого во времени. Но последовательное применение набора преобразований приводит к появлению визуального перемещения объекта, совсем как набор рисунков в кинеографе. Этот принцип проиллюстрирован на рис. 2.11.

Напомню вам, что компоненты сценария снабжены запускающимся в каждом кадре методом `Update()`. Поэтому, чтобы заставить куб вращаться вокруг своей оси, следует добавить в метод `Update()` код, который будет поворачивать его на небольшой угол кадр за кадром. Звучит понятно, верно?

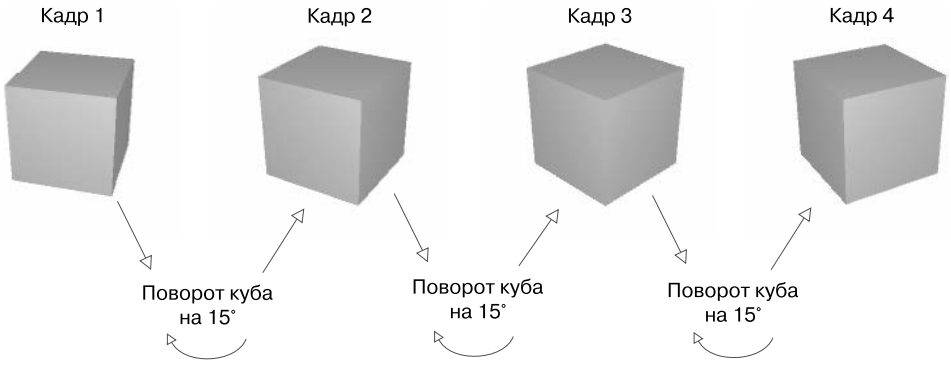


Рис. 2.11. Возникновение движения: циклический процесс преобразования статичных изображений

2.3.2. Написание кода

Применим рассмотренную концепцию на практике. Создайте новый скрипт (вы можете это сделать через меню Assets, выбрав Create и нажав C# script), присвойте ему имя Spin и напишите код из листинга 2.1 (не забудьте после ввода листинга сохранить файл!).

Листинг 2.1. Приводим объект во вращение

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Spin : MonoBehaviour {
    public float speed = 3.0f;

    void Update() {
        transform.Rotate(0, speed, 0);
    }
}
```

Вставьте классы Unity в этот сценарий

Объявление общедоступной переменной для скорости вращения

Поместите сюда команду Rotate, чтобы она запускалась в каждом кадре

Для добавления компонента скрипта к игроку перетащите сценарий с вкладки Project на строку Player на вкладке Hierarchy. Нажмите кнопку Play, и вы увидите, как все придет в движение, — вы написали код с вашей первой анимацией! В основном этот код — шаблон по умолчанию, к которому добавлены две строки. Посмотрим, что именно они делают.

Прежде всего мы добавили переменную для скорости в верхнюю часть определения класса. Скорость вращения определяется как переменная, а не как константа из-за способа отображения общедоступных переменных в Unity. Разберем эту переменную подробнее.