Майк Гейг

Разработка игр на Unity® 2018

за 24 Часа


Мировой
компьютерный
бестселлер

Mike Geig

Unity® 2018 Game Development in 24 Hours

Майк Гейг

Разработка игр на Unity® 2018

за 24 Часа

БОМБОРА™
Москва 2020

УДК 004.9
ББК 77.056с.я92
Г27

MIKE GEIG
UNITY 2018 GAME DEVELOPMENT IN 24 HOURS, SAMS TEACH YOURSELF

Authorized translation from the English language edition, entitled UNITY 2018 GAME DEVELOPMENT IN 24 HOURS, SAMS TEACH YOURSELF, 3rd Edition by MIKE GEIG, published by Pearson Education, Inc, publishing as Sams Publishing,

Гейг, Майк.

Г27 Разработка игр на Unity 2018 за 24 часа / Майк Гейг ; [перевод с английского М. А. Райтмана]. — Москва : Эксмо, 2020. — 464 с. — (Мировой компьютерный бестселлер. Геймдизайн).

ISBN 978-5-04-105963-7

Unity — невероятно мощный игровой движок, популярный среди разработчиков игр как профессионального, так и любительского уровня. Автор книги — Майк Гейг, руководитель направления информационно-разъяснительной работы в компании Unity Technologies.

Эта книга призвана ускорить обучение читателя и дать ему возможность как можно быстрее начать работу с Unity и освоить принципы игрового программирования.

Вы не только приобретете теоретические знания об игровом движке Unity, но и создадите небольшое портфолио игр, которое пригодится вам в будущем.

УДК 004.9
ББК 77.056с.я92

ISBN 978-5-04-105963-7

Оглавление

Предисловие

Unity — невероятно мощный игровой движок, популярный среди разработчиков игр как профессионального, так и любительского уровня. Эта книга призвана ускорить обучение читателя и дать ему возможность как можно быстрее начать работу с Unity (в течение примерно 24 уроков, если быть точным) и освоить основные принципы игрового программирования. В отличие от других книг, затрагивающих только определенные темы или описывающих создание одной конкретной игры, эта охватывает широкий спектр тем, и, кроме того, в ней описана процедура создания четырех игр. Выгода налицо! Закончив читать книгу, вы не только приобретете теоретические знания об игровом движке Unity, но и создадите небольшое портфолио игр, которое пригодится вам в будущем.

Для кого предназначена эта книга

Эта книга предназначена для тех, кто желает научиться использовать игровой движок Unity. Будь вы студент или эксперт по разработке — вы найдете на этих страницах что-нибудь интересное для себя. Книга не требует от вас предварительного наличия каких-либо знаний или опыта в разработке игр, так что, даже если это ваша первая попытка познать искусство создания игр, волноваться не о чем. Не торопитесь и получайте удовольствие. Вы и сами не заметите, как начнете учиться.

Как организована эта книга

Как и другие самоучители из этой серии, книга разделена на 24 главы, и на проработку каждой из них потребуется примерно один час. В книге представлены следующие главы.

- Час 1. «Введение в Unity». В этом часе мы научимся основам работы с различными компонентами игрового движка Unity.
- Час 2. «Игровые объекты». В часе 2 вы узнаете, как использовать главные элементы игрового движка Unity — игровые объекты. Мы также изучим системы координат и преобразования координат.
- Час 3. «Модели, материалы и текстуры». В этом часе мы научимся работать с графическими ресурсами Unity и применять к материалам шейдеры

и текстуры. Вы также узнаете, как впоследствии применять эти материалы к различным трехмерным объектам.

- Час 4. «Ландшафт и элементы окружающей среды». В часе 4 вы научитесь делать игровые миры, используя встроенную систему ландшафта Unity. Главное не бояться запачкать руки и углубиться в работу, создавая уникальные и невероятные пейзажи.
- Час 5. «Источники света и камеры». В часе 5 мы подробно поговорим об источниках света и камерах.
- Час 6. «Игра первая: «Великолепный гонщик»». Пришло время создать первую игру! В часе 6 вы сделаете игру «Великолепный гонщик», пустив в ход все приобретенные знания.
- Час 7. «Скрипты, часть 1». В часе 7 мы перейдем к созданию скриптов в Unity. Если вы никогда не занимались программированием — не волнуйтесь. Мы рассмотрим материал медленно и подробно, чтобы вы могли начать изучение с основ.
- Час 8. «Скрипты, часть 2». Мы продолжаем изучать тему, начатую в часе 7, но теперь сосредоточимся на более сложных задачах.
- Час 9. «Столкновения». Мы поговорим о взаимодействии и столкновении предметов, которые в современных видеоиграх встречаются повсеместно. Вы узнаете о физических и триггерных столкновениях. Кроме того, мы создадим физические материалы, чтобы придать объектам новые свойства.
- Час 10. «Игра вторая: „Шар хаоса“». Пора создать еще одну игру! В этом часе мы сделаем игру «Шар хаоса». Название говорящее, так как мы будем использовать различные столкновения, физические материалы и цели. Для игры потребуется и стратегическое мышление, и быстрая реакция!
- Час 11. «Префабы». Префабы позволяют многократно повторять одинаковые игровые объекты. В часе 11 вы научитесь создавать и редактировать префабы.
- Час 12. «Инструменты создания 2D-игр». В часе 12 мы освоим встроенный в Unity инструментарий для создания 2D-игр, а также научимся работать со спрайтами и физикой Box2D.
- Час 13. «Тайлмапы в 2D-играх». В часе 13 вы узнаете, как создавать сложные двухмерные миры с помощью простых спрайтовых тайлов.
- Час 14. «Пользовательские интерфейсы». В этом часе вы узнаете, как использовать встроенную в Unity систему пользовательских интерфейсов, и в качестве примера мы создадим меню для одной из игр.

- Час 15. «Игра третья: „Капитан Бластер“». Пришло время третьей игры! В этом часе мы создадим игру «Капитан Бластер» — аркадную ретрострелялку с космическими кораблями, астероидами и тому подобным.
- Час 16. «Системы частиц». Настала пора изучить эффекты частиц. В этой главе мы поэкспериментируем с системой частиц движка Unity, позволяющей создавать нереально крутые эффекты, и попробуем применить их к своим проектам.
- Час 17. «Анимация». В часе 17 вы узнаете об анимациях и системах анимации Unity. Мы поработаем с 2D- и 3D-анимацией и некоторыми инструментами для создания анимации.
- Час 18. «Аниматоры». Час 18 посвящен Mecanim — системе анимации в Unity. Вы узнаете, как использовать инструменты работы с состояниями и как смешивать анимации.
- Час 19. «Система Timeline». В этом часе вы узнаете, как собрать воедино множество анимаций игровых объектов с помощью системы **Timeline**.
- Час 20. «Игра четвертая: „Бег с препятствиями“». Наша четвертая игра называется «Бег с препятствиями». На ее примере мы рассмотрим новый способ прокрутки фонов и покажем, как можно реализовать его и тем самым расширить возможности игры.
- Час 21. «Работа со звуком». В часе 21 мы перейдем к созданию звуковых эффектов окружающей среды. Мы изучим 2D- и 3D-звук и узнаем, чем они отличаются.
- Час 22. «Разработка для мобильных устройств». В этом часе мы рассмотрим создание игр для мобильных устройств. Мы научимся использовать встроенный в мобильное устройство акселерометр и сенсорный дисплей.
- Час 23. «Доработка и развертка». Пора узнать, как сделать игру многоуровневой и как переносить данные между уровнями-сценами! Мы также поговорим о настройках развертывания игр и об игровом процессе.
- Час 24. «Финальные штрихи». Пора вспомнить весь пройденный нами путь в изучении Unity. В этом часе мы поговорим о том, чего мы уже успели достичь и куда можно двигаться дальше.

Благодарю за то, что прочитали предисловие! Я надеюсь, вы получите удовольствие от этой книги и узнаете из нее много нового. Удачи в вашем путешествии с игровым движком Unity!

Файлы примеров

В файлах примеров содержатся листинги программ из каждой главы с авторскими комментариями, все сторонние графические ассеты (текстуры, шрифты, модели), а также аудиоматериалы. Для доступа к сопроводительным файлам перейдите по адресу **http://addons.eksmo.ru/it/Unity.zip**.

Об авторе

Майк Гейг — руководитель направления информационно-разъяснительной работы в компании Unity Technologies. Он содействует процессу демократизации разработки игр путем создания и внедрения эффективных учебных материалов. Майк имеет опыт разработчика инди-игр, преподает в университете, а также пишет книги. Он геймер в душе и старается сделать разработку интерактивных развлечений веселее и доступнее для людей с любыми способностями. Мама, передаю тебе привет!

Посвящение

Кара, я решил дождаться третьего издания, чтобы, наконец, посвятить эту книгу тебе. Теперь, когда ты скажешь: «Ты никогда не посвящал мне книгу», я могу парировать: «Ты в этом уверена?» Спасибо за то, что была и остаешься моей опорой.

Благодарности

Премного благодарен всем, кто помог мне написать эту книгу.

В первую очередь, спасибо тебе, Кара, за твою поддержку на моем пути. Я не знаю, что мы будем говорить, когда книга выйдет в печать, но, скорее всего, ты в любом случае окажешься права. Люблю тебя, детка.

Линк и Люк! Мы должны слегка отстать от мамочки на некоторое время. Она вот-вот выбьется из сил.

Благодарю моих родителей. Теперь, когда я и сам отец, я понимаю, как тяжело вам было не придушить или не прибить меня. Спасибо, что оставили меня в живых.

Спасибо Анджелине Джоли. Благодаря ее роли в захватывающем фильме «Хакеры» (1995 год) я решил освоить компьютер. Вы даже не представляете, какое влияние вы имели на десятилетних детей в те времена. Вы — нечто!

Изобретателю вяленого мяса: история, может, и забыла ваше имя, но точно не забыла ваш продукт — и я его обожаю. Спасибо!

Спасибо Майклу Ву — за согласие стать не только техническим редактором этой книги, но и другим Майком в нашем подкасте Mikes' Video Game Podcast (мне необходимо было засветиться там).

Спасибо, Лора, за то, что уговорила меня написать эту книгу. Также благодарю тебя за то, что угостила меня обедом в GDC. Я чувствую, что этот обед, лучший из всех трех приемов пищи, в немалой степени помог мне закончить книгу.

И наконец, следует поблагодарить компанию Unity Technologies. Если бы вы не создали игровой движок Unity, эта книга была бы очень странной и вводила в недоумение.

1-Й ЧАС

Введение в Unity

Что вы узнаете в этом часе

- Как установить Unity
- Как создать новый проект или открыть существующий
- Как использовать редактор Unity
- Как перемещаться по сцене в Unity

Этот урок подготовит вас к полноценной работе в среде Unity. Мы приведем обзор различных типов лицензий Unity, а затем поможем установить тот, который вы выберете. Также в этом уроке вы узнаете, как создавать новые проекты и открывать существующие. Далее вы запустите мощнейший редактор Unity и исследуете его компоненты. И наконец, вы научитесь ориентироваться на сцене с помощью мыши и клавиатуры. Этот урок практический, так что загрузите Unity, пока будете читать, и приступайте к работе.

Установка Unity

Перед началом использования среды Unity необходимо загрузить и установить ее. На сегодня процедура установки программного обеспечения довольно проста и понятна, и Unity в этом смысле не исключение. Перед началом установки обратите внимание на три доступных варианта лицензии Unity: Unity Personal, Unity Plus и Unity Pro. Unity Personal — это бесплатная лицензия, в которой есть все необходимое, чтобы выполнить все примеры и проекты из данной книги. На самом деле Unity Personal достаточно для создания игр на коммерческой основе, вплоть до дохода 100 тыс. долларов США в год! Если вы сможете зарабатывать больше или хотите получить доступ к расширенным возможностям Unity Plus или Unity Pro (в основном касающимся работы в команде), вы всегда можете выполнить обновление в будущем.

ПРИМЕЧАНИЕ

Unity Hub

На момент публикации этой книги полным ходом идет разработка нового способа доступа к установке Unity и созданию проектов. Unity Hub — это централизованный лаунчер для различных ваших работ. К сожалению, он пока не готов настолько, чтобы включать его в данную книгу, но это вовсе не означает, что наш первый урок не имеет смысла. Вы можете установить Unity и создавать игры так, как описано здесь, если хотите. Если же вы решите использовать Unity Hub, то общие моменты, касающиеся установки и проектов, все еще будут в силе. Как говорится, «упаковка новая, а вкус все так же великолепен!»

Загрузка и установка Unity

В данном уроке предполагается, что вы используете лицензию Unity Personal. Если вы выберете версию Plus или Pro, то процедура установки будет очень похожа, за исключением момента, когда нужно выбрать тип лицензии. Когда будете готовы начать загрузку и установку Unity, выполните следующие действия.

1. Загрузите дистрибутив Unity со страницы загрузки **unity3d.com/ru/get-unity/download** и следуйте инструкциям, чтобы загрузить программу установки.

2. Запустите программу установки и следуйте инструкциям на экране, точно так же, как и при установке любого другого программного обеспечения.

3. При появлении показанного на рис. 1.1 окна не забудьте установить флажки **Unity 2018** и **Standard Assets**. Можно также установить демонстрационный проект и поддержку любых других платформ, которые вам нужны, если у вас достаточно места на жестком диске. На работу с данной книгой это никак не повлияет.

4. Выберите каталог установки Unity. Если вы не знаете, что следует выбрать, мы рекомендуем оставить параметры по умолчанию. Unity потребуется некоторое время, чтобы загрузить необходимые файлы. Во время загрузки вы будете видеть индикатор выполнения (рис. 1.2).

5. Если у вас уже есть учетная запись Unity, вам может быть предложено авторизоваться под своей учетной записью. Если у вас нет учетной записи Unity, следуйте инструкциям, чтобы создать ее. Убедитесь, что у вас есть доступ к электронной почте, так как вам потребуется подтвердить ее адрес. Вот и все! Установка Unity завершена.

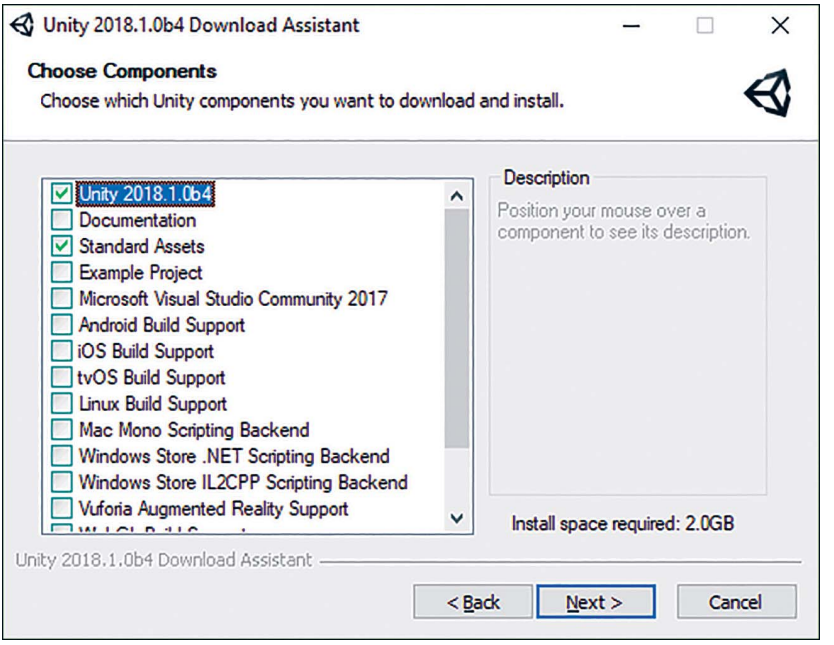


Рис. 1.1. Выбор компонентов для установки

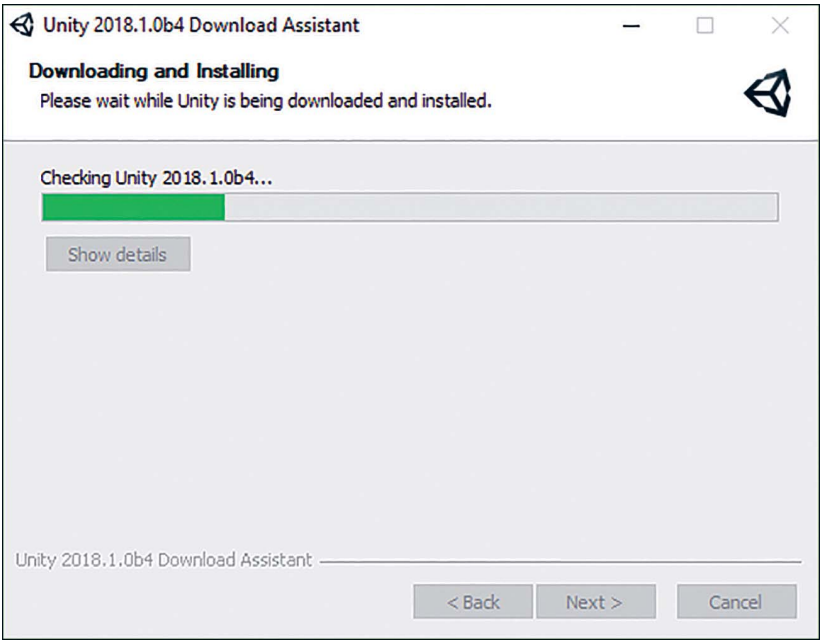


Рис. 1.2. Подождите, пока Unity загрузится на ваш компьютер

ПРИМЕЧАНИЕ

Поддерживаемые операционные системы и системные требования

Чтобы использовать Unity, вам нужен компьютер под управлением операционной системы Windows или macOS. Хотя есть версия редактора, которая работает в среде Linux, она не входит в список поддерживаемых ОС. Ваш компьютер также должен соответствовать приведенным ниже минимальным системным требованиям (требования взяты с сайта Unity на момент публикации книги).

- Windows 7 SP1+, Windows 8 или Windows 10 (только 64-разрядные версии); macOS10.9+.
- Обратите внимание, что Unity не тестировалась на серверных версиях Windows и macOS.
- Видеокарта с поддержкой DirectX 10 (версия шейдеров 4.0).
- Центральный процессор, поддерживающий набор инструкций SSE2 (как большинство современных процессоров).

Обратите внимание, что это минимальные требования.

ПРИМЕЧАНИЕ

Ссылки в Интернете

Все URL-адреса, приведенные в данной книге, актуальны на момент ее публикации. Однако интернет-адреса иногда меняются. Если окажется, что приведенные ссылки больше не работают, вы можете подыскать подходящие источники информации самостоятельно.

Знакомство с редактором Unity

Теперь, когда вы установили Unity, можно начать изучать редактор Unity. Редактор Unity является визуальным компонентом, который позволяет создавать игры по принципу What You See Is What You Get — «Что видишь, то и получишь». Поскольку бóльшая часть работы подразумевает именно взаимодействие с редактором, многие говорят Unity, имея в виду «редактор Unity». В этом разделе мы рассмотрим все элементы редактора Unity и то, как они сочетаются друг с другом в процессе создания игр.

Диалоговое окно Project

При первом запуске Unity вы увидите диалоговое окно **Project** (рис. 1.3). Вы можете использовать его, чтобы открыть последние проекты, просмотреть созданные проекты или начать новые.

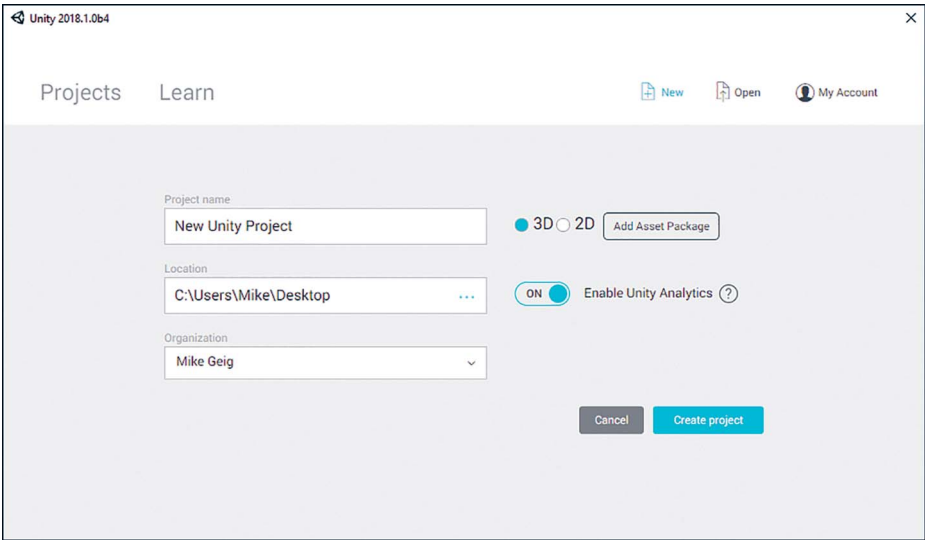


Рис. 1.3. Диалоговое окно **Project** (приведена версия для Windows, версия для macOS выглядит аналогично)

Если вы уже создали проект в Unity, то он будет открываться сразу при каждом запуске редактора. Чтобы вернуться к диалоговому окну **Project**, выберите команду меню **File** ⇒ **New Project**, чтобы открыть диалоговое окно **Create New Project,** или команду меню **File** ⇒ **Open Project**, чтобы открыть диалоговое окно **Open Project**.

СОВЕТ

Открытие диалогового окна Project

Вы можете выбрать действие для Unity при запуске: открытие диалогового окна **Project** или последнего проекта. Для этого выберите пункт меню **Edit** ⇒ **Preferences** (в macOS: **Unity** ⇒ **Preferences**) и измените положение флажка **Load Previous Project on Startup**.

ПРАКТИКУМ

Создание первого проекта

Теперь вы готовы создать первый проект. Обязательно запомните расположение файлов, которое зададите для проекта, чтобы потом легко найти его. На рис. 1.4 показано диалоговое окно создания проекта. Выполните следующие действия.

Запустите редактор Unity и найдите диалоговое окно **New Project** (если Unity открывает какой-то проект, выберите команду меню **File** ⇒ **New Project**).

Выберите расположение для вашего проекта. Рекомендуется создать папку *Unity Projects* и хранить все работы в одном месте. Если вы не знаете, куда пристроить проект, оставьте расположение по умолчанию.

Назовите ваш проект **Hour 1 TIY**. Unity создаст папку с таким же именем в том каталоге, который вы выбрали.

Установите переключатель в положение **3D**, а кнопку **Add Asset Package** и **Enable Unity Analytics** пока не трогайте.

Нажмите кнопку **Create Project**.

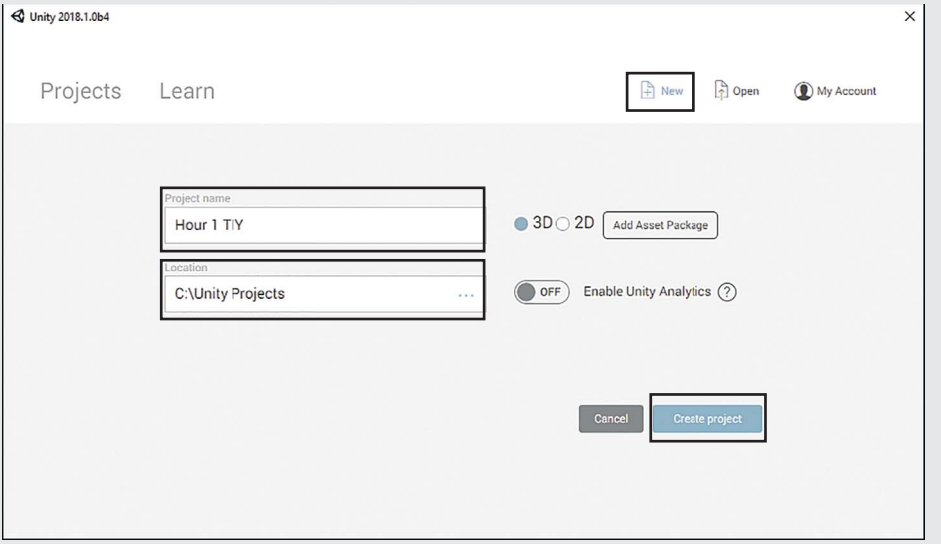


Рис. 1.4. Настройки первого проекта

ПРИМЕЧАНИЕ

Что такое 2D, 3D, пакеты и аналитика?

Вам, возможно, любопытно, для чего нужны остальные элементы управления в диалоговом окне **New Project**. Переключатель **2D/3D** позволяет выбрать тип игры, которую вы планируете создать. Не беспокойтесь, если сделаете неправильный выбор или не уверены в нем. Эти опции влияют лишь на настройки редактора и могут быть изменены в любой момент. Кнопка **Add Asset Package** позволяет автоматически добавить в новый проект файлы, которые вы обычно используете в своих проектах. И, наконец, **Enable Unity Analytics** — очень мощный инструмент, который подключает вас к сервису Unity Analytics. С его помощью вы можете генерировать данные об аудитории вашей игры. Это невероятно крутой сервис для разработчиков игр, но нам он пока не требуется.

Интерфейс Unity

Сейчас мы всего лишь установили Unity и ознакомились с диалоговым окном **Project**. Теперь настало время копнуть глубже. При первом открытии нового

проекта Unity вы увидите набор серых окон (называемых *панелями*), и интерфейс будет выглядеть довольно аскетично (рис. 1.5). Не волнуйтесь — скоро здесь все будет гораздо круче. В следующем разделе мы рассмотрим каждую из этих панелей последовательно. Но сначала давайте поговорим о структуре интерфейса в целом.

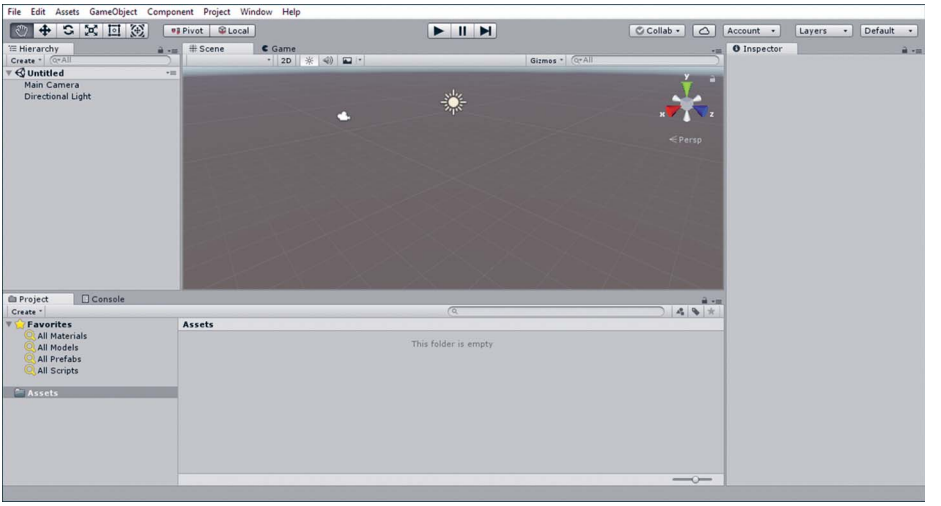


Рис. 1.5. Интерфейс среды Unity

В первую очередь среда Unity позволяет точно определить, как вы хотите работать. Все панели можно перемещать, соединять, дублировать или изменять. Например, нажав и удерживая кнопку мыши на слове **Hierarchy**, вы выбираете панель **Hierarchy** и, перетащив ее к панели **Inspector** (справа), можете соединить их в одну панель с двумя вкладками. Вы можете также установить курсор на границу между панелями и, нажав и удерживая кнопку мыши, перетаскиванием изменить их размер. Почему бы вам не воспользоваться случаем и не поэкспериментировать с панелями, разместив все на свой вкус?

Если в итоге у вас получится рабочая среда, которая вам не нравится, вы можете быстро и легко переключиться на вид по умолчанию, выбрав команду меню **Window** ⇒ **Layouts** ⇒ **Default Layout**. Экспериментируя с панелями, попробуйте несколько других стандартных рабочих сред. (Я фанат среды **Wide**.) Если вы создадите рабочую среду, которая вам пришлась по вкусу, можете сохранить ее, выбрав пункт меню **Window** ⇒ **Layouts** ⇒ **Save Layout** (для этой книги задействована пользовательская рабочая среда под названием **Pearson**). После того как вы сохранили пользовательскую рабочую среду, вы всегда можете вернуться к ней, если случайно что-то измените.

ПРИМЕЧАНИЕ

Выбор правильной рабочей среды

Нет двух одинаковых людей, и точно так же нет двух одинаковых рабочих сред. Хорошая среда поможет вам создавать проекты и сильно упростит вам жизнь. Не забудьте выделить время, чтобы поэкспериментировать с настройками рабочей среды и выбрать ту, которая лучше подойдет. Вам предстоит многое сделать с Unity, так что стоит настроить среду так, чтобы было удобно.

Дублирование панелей — тоже довольно простая процедура. Достаточно щелкнуть правой кнопкой мыши на *вкладке* любой панели (например, вкладке **Inspector** на рис. 1.6), установить указатель мыши на кнопку **Add Tab**, и вы увидите список панелей (рис. 1.6). Вы спросите — а зачем нужно дублировать панель? Предположим, что в процессе работы вы слишком увлеклись и закрыли панель. Повторное добавление вкладки позволит вернуть ее. Кроме того, обратите внимание на возможность создания нескольких копий панели **Scene**. Каждый вид сцены может совпадать с конкретным элементом или осью в пределах вашего проекта. Если вы хотите увидеть, как это работает, попробуйте шаблон **4 Split**, выбрав пункт меню **Window** ⇒ **Layouts** ⇒ **4 Split**. (Если вы создали настройку среды, которая вам нравится, не забудьте сохранить ее, прежде чем пробовать вариант **4 Split.**)

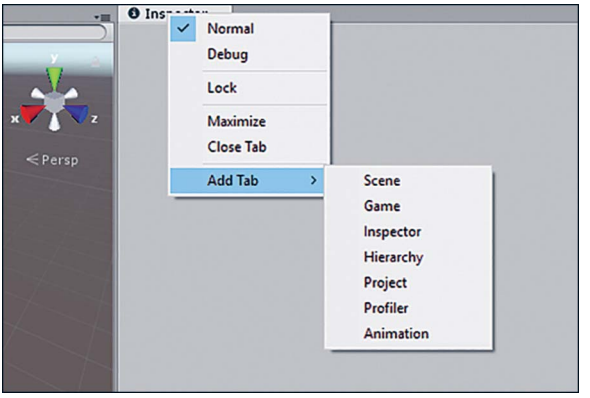


Рис. 1.6. Добавление новой вкладки

Теперь давайте без лишних слов взглянем поближе на некоторые панели.

Панель Project

Все, что создано в рамках проекта (файлы, скрипты, текстуры, модели и так далее), можно найти на панели **Project** (рис. 1.7). Это панель, на которой отображаются все ассеты и структура проекта. При создании нового проекта вы увидите

одну папку с именем **Assets**. Если вы заглянете в папку на жестком диске, где сохранили проект, то найдете там папку *Assets*. Дело в том, что Unity отображает на панели **Project** реальные папки на жестком диске. Если создать файл или папку в Unity, соответствующий файл или папка появится в проводнике (и наоборот). Вы можете перемещать элементы на панели **Project** путем перетаскивания. Это позволяет размещать элементы внутри папок или реорганизовывать свой проект «на лету».

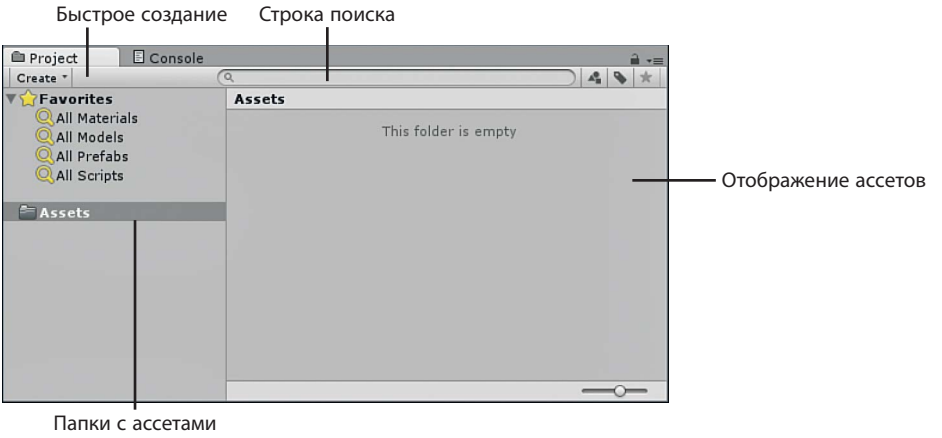


Рис. 1.7. Панель **Project**

ПРИМЕЧАНИЕ

Ассеты и объекты

Ассет — это любой элемент, который существует в папке **Assets**. Все текстуры, меши, звуковые файлы, скрипты и т. д. считаются ассетами. В противоположность этому игровой объект — это объект, который является частью сцены или уровня. Вы можете создавать ассеты из игровых объектов, а также игровые объекты из ассетов.

ВНИМАНИЕ!

Перемещение ассетов

Unity поддерживает связи между различными ассетами, задействованными в проектах. Поэтому перемещение или удаление элементов за пределами Unity может вызвать проблемы. Как правило, лучшего всего управлять ассетами внутри Unity.

При выборе папки на панели **Project** содержимое папки отобразится в разделе **Assets** справа. Как показано на рис. 1.7, папка **Assets** сейчас пуста, поэтому справа ничего не отображается. Если хотите создать ассет, вы можете легко сделать это,

открыв раскрывающийся список **Create**. Он позволяет добавлять в проект всевозможные ассеты и папки.

СОВЕТ

Структура проекта

Структура чрезвычайно важна при управлении проектами. По мере того как ваши проекты будут становиться объемнее, количество ассетов станет расти, и находить что-то конкретное окажется все труднее и труднее. Вы можете избежать множества разочарований, используя несколько простых правил структурирования.

- Для каждого типа объекта (сцены, скрипты, текстуры и так далее) следует создать отдельную папку.
- Все ассеты должны находиться в папках.
- Если вы используете вложенные папки, убедитесь, что в такой структуре есть смысл. Папки должны иметь конкретные, а не неопределенные или обобщенные имена.

Соблюдение этих простых правил существенно скажется на вашей работе.

Кнопки **Favorites** позволяют вам быстро выбрать все ассеты определенного типа, чтобы просмотреть их. При нажатии на одну из кнопок **Favorites** (например, **All Models**) или поиска по имени вы можете сузить круг отображаемых ассетов в разделах **Assets** и **Assets Store**. Выбрав пункт **Asset Store**, вы можете просматривать ассеты, которые соответствуют вашим критериям поиска, в магазине Unity Asset Store (рис. 1.8). Вы можете еще больше ограничить результаты, выбрав платные или бесплатные ассеты. Это фантастическая возможность, поскольку она позволяет подключить в проект нужные вам ассеты, не выходя из интерфейса Unity.

Рис. 1.8. Поиск в магазине Unity Asset Store

Панель Hierarchy

Панель **Hierarchy** (рис. 1.9) во многих отношениях похожа на панель **Project**. Разница заключается в том, что на панели **Hierarchy** отображаются все элементы текущей сцены, но не всего проекта. При создании проекта в Unity вы по умолчанию увидите новую сцену, в которой есть только два элемента: игровые объекты **Main Camera** и **Directional Light**. Когда вы будете добавлять элементы на сцену, они появятся на панели **Hierarchy**. Как и на панели **Project**, вы можете использовать меню **Create**, чтобы быстро добавлять элементы на сцену, искать их с помощью строки поиска и перетаскивать, чтобы организовывать и вкладывать их в другие.

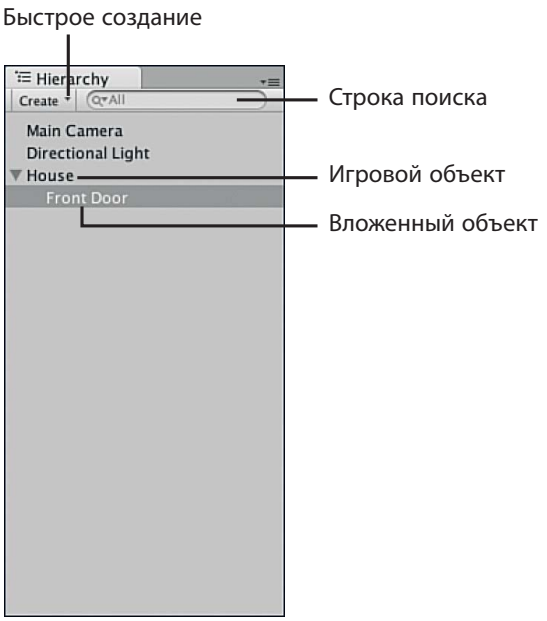


Рис. 1.9. Панель **Hierarchy**

СОВЕТ

Вложенные объекты

Вложение — это определение отношений между двумя (и более) элементами. На панели **Hierarchy** вы можете щелкнуть и перетащить один элемент на другой, чтобы «вложить» его. Произойдет формирование т. н. отношений родитель/ребенок (или предок/потомок). В этом случае объект сверху является родительским, а любые объекты под ним — дочерними. Визуально вложенность объекта отображается в виде отступа. Как вы узнаете позже, вложенность объектов на панели **Hierarchy** может влиять на их поведение.

СОВЕТ

Сцены

Сценой в Unity называется то, что вам может быть уже известно как «уровень» или «карта». По мере разработки проекта Unity у каждой коллекции объектов и поведения должна появиться своя сцена. Поэтому, если вы создаете игру со снежным уровнем и уровнем джунглей, это должны быть отдельные сцены. Вы увидите, что термины «сцена» и «уровень» взаимозаменяемы, если поищете ответы в Интернете.

СОВЕТ

Организация сцены

Первое, что вы должны сделать при работе с новым проектом в Unity, это создать папку **Scenes** под папкой **Assets** на панели **Project**. Таким образом, все ваши сцены (или уровни) будут храниться в одном месте. Обязательно присваивайте сценам описательные имена. «Сцена1» — это, безусловно, круто, но, когда у вас их уже 30, такое неопределенное имя может запутать.

Панель Inspector

На панели **Inspector** отображаются все свойства выбранного элемента. Выберите любой ассет или объект на панели **Project** или **Hierarchy**, и на панели **Inspector** автоматически отобразится соответствующая информация.

На рис. 1.10 вы видите панель **Inspector** после того, как на панели **Hierarchy** выбран объект **Main Camera**.

Давайте разберем некоторые функции.

- Если сбросить флажок рядом с именем объекта, он становится отключенным и не отображается в проекте.
- Раскрывающиеся списки (например, Tag и Layer, о которых мы подробнее поговорим позже) используются для выбора опций из заранее определенного набора.
- Поля ввода, раскрывающиеся списки и ползунки позволяют изменять значения, и эти изменения мгновенно автоматически отражаются на сцене, даже если игра запущена!
- Каждый игровой объект является контейнером для различных компонентов (например, Transform, Camera и Audio Listener на рис. 1.10). Вы можете отключить эти компоненты, сбросив соответствующие флажки, или удалить их, щелкнув правой кнопкой мыши и выбрав пункт Remove component.
- Вы можете добавить компоненты, нажав кнопку Add Component.

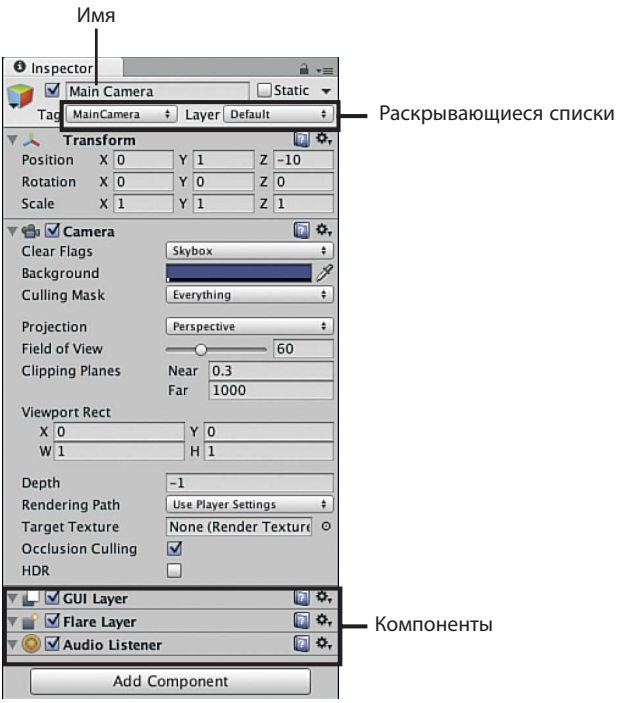


Рис. 1.10. Панель **Inspector**

ВНИМАНИЕ!

Изменение свойств при запущенной сцене

Возможность изменять свойства объекта и сразу видеть изменения на запущенной сцене — очень мощный инструмент. Он позволяет «на лету» настраивать такие моменты, как скорость движения, высота прыжков, сила столкновения и т. д., и все это без остановки и повторного запуска игры. Однако любые изменения, внесенные в свойства объекта во время работы сцены, будут отменены, когда сцена закончится. Если вы внесли изменения, и они вам понравились, обязательно помните, что потребуется установить их снова, когда сцена завершится.

Панель Scene

Панель **Scene** — наиболее важный компонент интерфейса, с которым вы работаете, так как она позволяет непосредственно увидеть создаваемую игру (см. рис. 1.11). С помощью мыши и клавиатуры вы можете передвигаться внутри сцены и помещать объекты там, где хотите. Это дает высочайший уровень контроля.

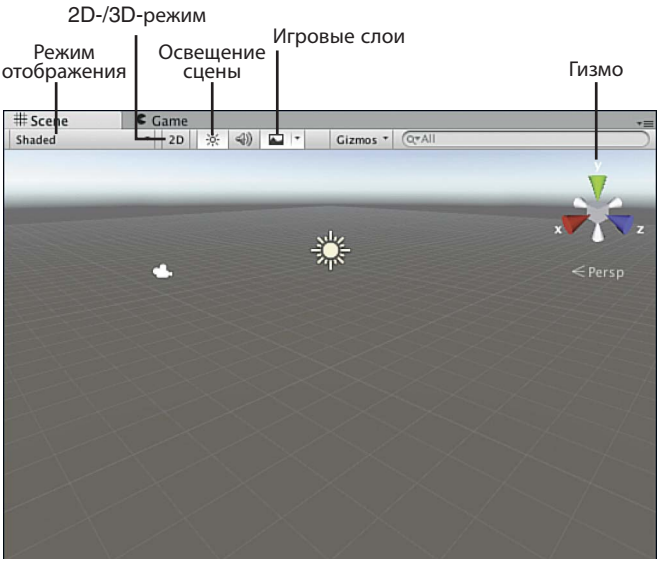


Рис. 1.11. Панель **Scene**

В скором времени вы узнаете, как перемещаться в пределах сцены, но сейчас давайте сосредоточимся на элементах управления, входящих в состав панели **Scene**.

- Режим отображения. Эта опция управляет тем, как отрисовывается сцена. Значение по умолчанию — Shaded предполагает, что объекты будут отрисованы с полноцветными текстурами.
- 2D/3D. Этот элемент управления позволяет переключаться с трехмерного вида на двухмерный. Обратите внимание, что в двухмерном виде не отображается гизмо (вы узнаете о нем позже в этом уроке).
- Освещение сцены. Этот элемент управления определяет, как освещаются объекты на панели Scene — общим освещением по умолчанию или только источниками, которые размещены в пределах сцены. По умолчанию включается встроенное общее освещение.
- Режим звука. Эта кнопка включает/выключает источник звука на панели Scene.
- Игровые слои. Этот элемент управления определяет, будут ли отображаться такие элементы, как небо, туман и другие эффекты, на панели Scene.
- Раскрывающийся список Gizmos. Этот элемент управления позволяет выбрать, какие гизмо — индикаторы, которые помогают при визуальной отладке и настройке — отображаются на панели Scene. Элемент также определяет, будет ли отображаться сетка размещения.

- **Гизмо**. Этот элемент управления показывает, в каком направлении вы сейчас смотрите, и позволяет расположить сцену по осям.

ПРИМЕЧАНИЕ

Гизмо

Гизмо сцены предоставляет много возможностей на панели **Scene**. Как вы видите, гизмо имеет индикаторы **X**, **Y** и **Z**, которые совпадают с тремя осями. Благодаря этому вы можете легко определить, в какую сторону смотрите на сцене. Вы узнаете больше об осях и трехмерном пространстве в часе 2. Гизмо также дает возможность контролировать выравнивание сцены. Если нажать на одну из осей гизмо, вид сцены сразу привяжется к этой оси и расположится в направлении, например, «сверху» или «слева». При нажатии на куб в центре гизмо позволяет переключаться между режимами **Iso** и **Persp** (Изометрия и Перспектива).

Режим **Iso** — это трехмерный вид без применения перспективы. Режим **Persp** — это трехмерный вид с применением перспективы. Попробуйте изменить режим и посмотрите, как это влияет на отображение сцены. Вы заметите, что значок выглядит как параллельные линии в режиме изометрии и как расходящиеся линии в режиме перспективы.

Панель Game

Последнее, что нам нужно рассмотреть, — это панель **Game**. По сути, панель **Game** позволяет вам «испытывать» игру внутри редактора и выполнить тем самым полное моделирование текущей сцены. Все элементы и механика игры на данной панели выводятся так же, как если бы игра была собрана полноценно. На рис. 1.12 приведен пример панели **Game**. Обратите внимание, что, хотя кнопки **Play**, **Pause** и **Step** не относятся к панели **Game**, они используются для управления этой панелью, поэтому тоже приведены на рисунке.

СОВЕТ

Если панель Game скрыта

Если панель **Game** скрыта за панелью **Scene** или вкладка **Game** вовсе отсутствует, не беспокойтесь. Как только вы нажмете кнопку **Play**, вкладка **Game** тут же появится в редакторе и выведет вам запущенную игру.

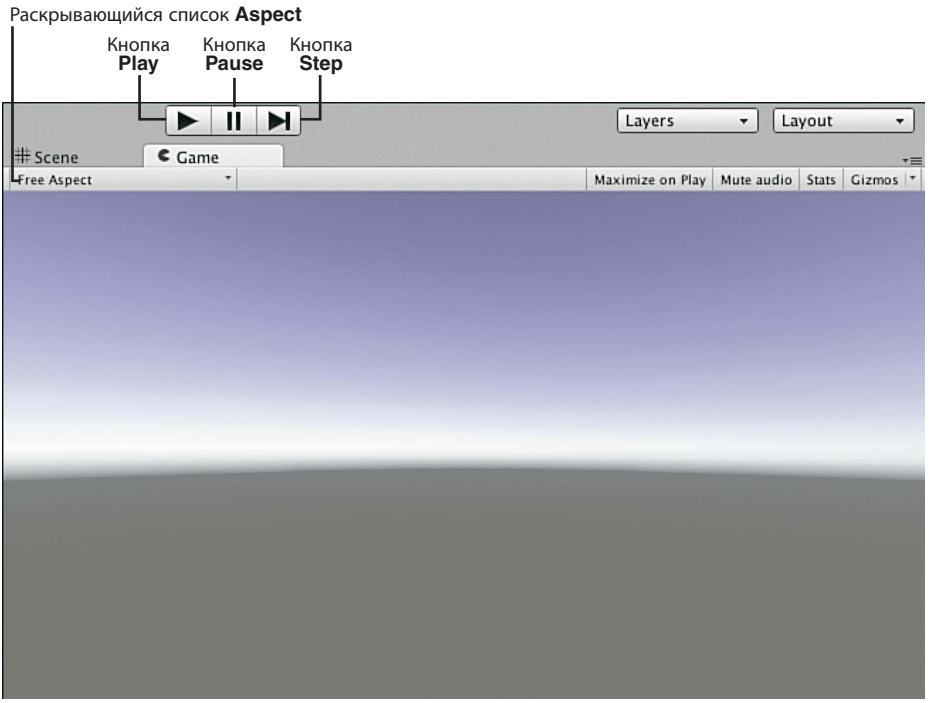


Рис. 1.12. Панель **Game**

На панели **Game** есть ряд элементов управления, которые используются для тестирования игры.

- Play. Кнопка Play позволяет воспроизвести текущую сцену. Все элементы управления, анимация, звуки и эффекты будут активированы. Здесь игра будет работать примерно так же, как если бы она была запущена автономно за пределами Unity (например, на компьютере или мобильном устройстве). Чтобы остановить игру, еще раз нажмите кнопку Play.
- Pause. Кнопка Pause приостанавливает выполнение игры, запущенной на панели Game. Игра сохраняет свое состояние и после возобновления продолжится с того же момента, на котором остановилась во время паузы. При повторном нажатии кнопки Pause игра возобновится.
- Step. Кнопка Step на панели Game может быть использована, когда игра приостановлена. Нажатие кнопки Step воспроизводит один кадр игры. Это позволяет вам пошагово выполнять игру и отлаживать любые возникающие проблемы. При нажатии кнопки Step во время выполнения игра приостанавливается.
- Aspect. В этом раскрывающемся списке вы можете выбрать соотношение сторон для панели Game во время выполнения игры. По умолчанию

установлена опция Free Aspect (Свободное соотношение), но вы можете изменить эту настройку, выбрав формат именно той платформы, для которой разрабатываете игру.

- Maximize on Play. Эта кнопка определяет, будет ли панель Game разворачиваться на весь размер окна редактора при запуске игры. По умолчанию опция выключена, и игра воспроизводится только в пределах панели Game.
- Mute Audio. Эта кнопка отключает звук во время игры. Весьма удобная опция, если сидящий рядом с вами человек устал слушать одни и те же звуки, пока вы тестируете игру!
- Stats. Эта кнопка определяет, будет ли выводиться на экран статистика рендеринга во время выполнения игры. Данные статистики могут быть полезны для измерения эффективности выполнения сцены. Опция по умолчанию отключена.
- Gizmos. Это одновременно и кнопка, и раскрывающийся список. Кнопка позволяет выбрать, будут ли отображаться гизмо во время выполнения игры. По умолчанию они не отображаются. Раскрывающийся список (открывается нажатием на маленькую стрелку) позволяет включить или выключить конкретные гизмо.

ПРИМЕЧАНИЕ

Запуск, пауза и остановка

Поначалу вам, возможно, будет трудно понять, что означают термины «Запуск», «Пауза» и «Остановка». Когда игра не выполняется на панели **Game**, она считается остановленной. В этом состоянии игровое управление не работает и игра не может воспроизводиться. При нажатии кнопки **Play** игра начинает выполняться, то есть запускается. Слова «Игра», «Выполнение» и «Запуск» в этом контексте означают одно и то же. Если игра запущена и нажата кнопка **Pause**, то игра приостанавливается, сохраняя свое текущее состояние, то есть она находится в режиме паузы. Разница между паузой (приостановкой) и остановкой игры состоит в том, что после паузы игра возобновляет выполнение с того момента, на котором была приостановлена, а после остановки начинает выполняться с самого начала.

Важная деталь: панель инструментов

Панель инструментов крайне важна в редакторе Unity. На рис. 1.13 показаны ее компоненты.

- Инструменты преобразования объектов: эти кнопки позволяют управлять игровыми объектами. В дальнейшем мы рассмотрим их назначение более подробно. Обратите особое внимание на кнопку с пиктограммой в виде руки. Это инструмент Hand, и его описание будет приведено в этом часе.

- Переключатели гизмо: переключатели позволяют управлять отображением гизмо на сцене. Пока мы не будем их трогать.
- Элементы управления панели Game: кнопки, используемые для управления панелью Game.
- Управление учетными записями и сервисами: эти кнопки позволяют управлять учетной записью Unity, которую вы используете, а также подключенными к вашему проекту сервисами.
- Раскрывающийся список настройки слоев: с его помощью вы можете выбрать, какие слои будут отображаться на панели Scene. По умолчанию отображаются все имеющиеся слои. Пока не будем трогать этот раскрывающийся список. Работу со слоями мы более подробно рассмотрим в часе 5.
- Раскрывающийся список настройки макета: этот раскрывающийся список позволяет быстро изменить макет редактора.

Рис. 1.13. Панель инструментов

Навигация на панели Scene в Unity

Панель **Scene** позволяет вам подробно контролировать процесс создания игры. Возможность размещать и изменять объекты с визуальным отображением, бесспорно, крута, но, если вы не можете передвигаться внутри сцены, пользы будет мало. В этом разделе описано несколько различных способов изменить свою позицию при навигации по сцене.

СОВЕТ

Изменение масштаба

Независимо от того, какой метод навигации вы используете, колесико мыши всегда служит для изменения масштаба в пределах сцены. По умолчанию сцена масштабирует изображение относительно центра панели **Scene**. Если вы зажмете клавишу **Alt** во время прокрутки, то масштабирование будет выполняться относительно точки, в которой в данный момент находится указатель мыши. Самое время попробовать!

Инструмент Hand

Инструмент **Hand** (клавиша **Q**) дает возможность легкого передвижения по панели с помощью мыши (рис. 1.14). Этот инструмент особенно полезен, если вы используете мышь с одной кнопкой (так как другие методы требуют наличия двух

кнопок мыши). В таблице 1.1 кратко описан каждый элемент управления инструмента **Hand**. О других кнопках рядом с инструментом **Hand** мы поговорим позже, их пока трогать не нужно.

Рис. 1.14. Инструмент **Hand**

ТАБЛ. 1.1. Элементы управления инструмента Hand

Действие	Результат
Перетаскивание	Перемещение камеры по сцене
Перетаскивание с зажатой клавишей Alt	Вращение камеры вокруг указанной точки
Перетаскивание правой кнопкой мыши с зажатой клавишей Ctrl (или клавишей ⌘ в macOS)	Масштабирование камеры

Список сочетаний клавиш Unity можно найти по адресу **docs.unity3d.com/ru/current/Manual/UnityHotkeys.html**.

ВНИМАНИЕ!

Различные камеры

Работая в Unity, вы будете использовать два типа камер. Первая — стандартный игровой объект «камера». Обратите внимание, что такая на вашей сцене уже есть по умолчанию. Вторая камера скорее виртуальная, она определяет ваш обзор на панели **Scene**. В данном руководстве, говоря «камера», мы будем иметь в виду именно второй тип. Игровым объектом «камера» вы не управляете так, как вашим обзором.

Режим Flythrough

Режим **Flythrough** позволяет перемещаться по сцене, используя традиционную систему управления «от первого лица». Он прекрасно знаком тем, кто играет в игры с видом «от первого лица» (например, шутеры). Если вы не из их числа, то к этому режиму придется слегка привыкнуть. Но, когда освоитесь с ним, вы полюбите его.

Чтобы перейти в режим **Flythrough**, удерживайте нажатой правую кнопку мыши, когда указатель мыши находится над панелью **Scene**. Все действия, перечисленные в таблице 1.2, выполняются с зажатой правой кнопкой мыши.

ТАБЛ. 1.2. Управление в режиме Flythrough

Действие	Результат
Перемещение мыши	Поворот камеры, дающий ощущение, словно вы «осматриваете» сцену
Нажатие клавиш **W**, **A**, **S** и **D**	Эти клавиши используются для перемещения по сцене в направлениях «вперед», «влево», «назад» и «вправо» соответственно
Нажатие клавиш **Q** и **E**	Эти клавиши используются для перемещения по сцене в направлениях «вверх», «вниз» соответственно
Удерживание клавиши **Shift** с нажатием клавиш **W**, **A**, **S** и **D** или **Q** и **E**	Дает тот же эффект, что и буквенные клавиши, но движение выполняется быстрее. Считайте это кнопкой «Бег»

СОВЕТ

Быстрые команды

Есть много способов, позволяющих точно управлять вашим перемещением по сцене. Но иногда вам просто надо быстро переместиться куда-то. В этом случае вам будут полезны так называемые быстрые команды. Если вы хотите моментально перейти к объекту на сцене или увеличить его, можете сделать это, выделив нужный объект на панели **Hierarchy** и нажав клавишу **F** (сокращенно от Frame Select — выбор кадра). Сцена тут же привяжется к выбранному объекту. Вы можете достичь того же эффекта двойным щелчком мыши по любому объекту на панели **Hierarchy**. Есть и другая быстрая команда, которую вы уже знаете: гизмо сцены позволяет моментально привязать камеру к любой оси. Таким образом, вы можете рассмотреть объект под нужным углом без необходимости вручную крутить камеру сцены. Попрактикуйтесь с этими командами, и перемещаться по сцене станет проще простого!

СОВЕТ

Дополнительные средства обучения

При открытии диалогового окна **New Project** или **Open Project** вы наверняка обратили внимание на кнопку **Learn** (рис. 1.15). Нажав ее, вы увидите новые учебные материалы Unity. Они прекрасно дополнят ваши знания и определенно стоят вашего времени, если вы хотите побольше попрактиковаться в основах работы с Unity (хотя это несколько предвзятое суждение, потому что я сам участвовал в разработке материалов).

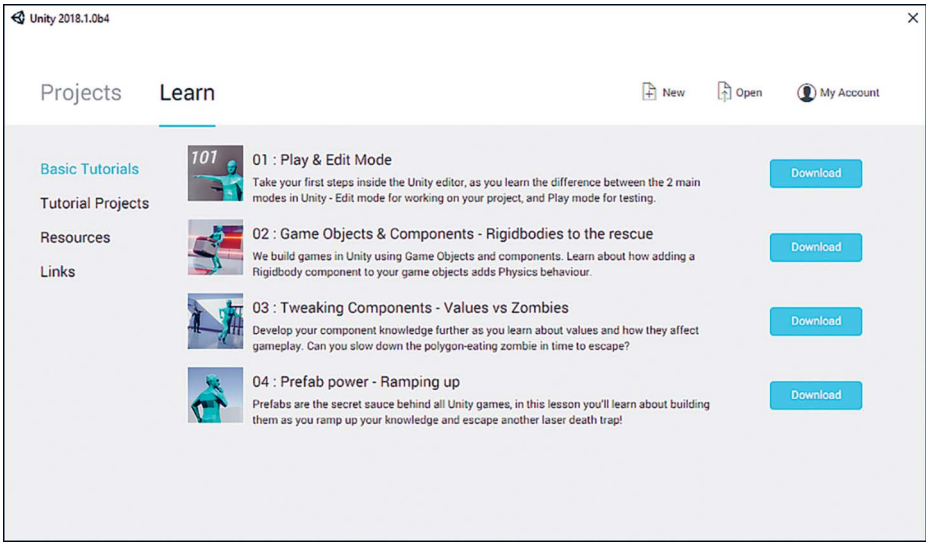


Рис. 1.15. Раздел **Learn** диалогового окна **Project**

Резюме

В этом часе вы познакомились с игровым движком Unity. Вы начали с загрузки и установки среды Unity. Затем вы научились открывать и создавать проекты. После этого вы узнали обо всех панелях, из которых состоит редактор Unity. Вы также узнали, как перемещаться по панели **Scene**.

Вопросы и ответы

Вопрос: **Ассеты и игровые объекты — это одно и то же?**

Ответ: Не совсем. Разница состоит в том, что ассет имеет соответствующий файл или группу файлов на жестком диске, а игровой объект — нет. Ассет может содержать игровой объект, а может и не содержать.

Вопрос: **В среде много различных элементов управления и опций. Я должен сразу запомнить их все?**

Ответ: Вовсе нет. Элементы управления и параметры, настроенные по умолчанию, охватывают бóльшую часть возможных ситуаций. По мере того как вы будете осваивать Unity, вы узнаете больше о различных доступных элементах управления. Этот урок предназначен для того, чтобы показать вам общую картину и ознакомить вас с Unity.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Верно или нет: вы должны приобрести Unity Pro, чтобы создавать коммерческие игры.
2. Какая панель позволяет визуально манипулировать объектами на сцене?
3. Верно или нет: вы всегда должны перемещать файлы ассетов из среды Unity, а не использовать файловый менеджер операционной системы.
4. Верно или нет: при создании нового проекта вы должны добавить в него все ассеты, которые, как вам кажется, отлично подойдут к нему.
5. Какой режим активируется в панели **Scene** путем удерживания правой кнопки мыши?

Ответы

1. Нет. Вы можете создавать игры с лицензией Unity Personal или Unity Plus.
2. Панель **Scene**.
3. Верно. Это помогает Unity следить за ассетами.
4. Нет. Такой подход занимает много места и замедляет ваш проект.
5. Режим **Flythrough**.

Упражнение

Уделите некоторое время, чтобы отработать на практике изученные вами в этом часе понятия. На данном этапе важно закрепить базовое понимание редактора Unity, потому что все, что вы узнаете дальше в книге, так или иначе связано с этим.

Итак, выполните следующие шаги.

1. Создайте новую сцену, выбрав команду меню **File** ⇒ **New Scene** или нажав сочетание клавиш **Ctrl+N** (⌘+**N** в macOS).
2. Создайте папку на панели **Project**, щелкнув правой кнопкой по строке **Assets** и выбрав команду **Create** ⇒ **Folder**. Назовите созданную папку **Scenes**.

3. Сохраните сцену, выбрав команду меню **File** ⇒ **Save Scene** или нажав сочетание клавиш **Ctrl+S** (⌘+**S** в macOS). Не забудьте поместить сцену в созданную вами папку **Scenes** и присвоить ей описательное имя.
4. Добавьте на сцену куб. Вы можете сделать это одним из трех способов.
   - В раскрывающемся списке GameObject в верхней части редактора выберите пункт 3D Object ⇒ Cube.
   - Выберите пункт Create ⇒ 3D Object ⇒ Cube на панели Hierarchy.
   - Щелкните правой кнопкой мыши на панели Hierarchy и выберите пункт 3D Object ⇒ Cube.
5. Выберите вновь добавленный куб на панели **Hierarchy** и поэкспериментируйте с его свойствами на панели **Inspector**.
6. Попрактикуйтесь в навигации по сцене, используя режим **Flythrough**, инструмент **Hand** и сочетания клавиш. Используйте куб в качестве ориентира.

2-Й ЧАС

Игровые объекты

Что вы узнаете в этом часе

- Как работать с двумерной и трехмерной системами координат
- Как работать с объектами игры
- Как работать с преобразованиями

Игровые объекты — основные компоненты проекта Unity. Каждый элемент, который находится на сцене, — это игровой объект или образован от него. В этом часе вы изучите игровые объекты в Unity. Однако сначала вы должны получить представление о двумерной и трехмерной системах координат. Затем мы начнем работу со встроенными в Unity игровыми объектами, а завершим этот час изучением различных преобразований для них. Данная информация лежит в основе всего дальнейшего понимания книги. Уделите время этому разделу и проработайте его как следует.

Измерения и системы координат

За красочной картинкой любой видеоигры всегда стоят математические конструкции. Все свойства, движения и взаимодействия — это в итоге обычные числа. К счастью для вас, большая часть подготовительной работы уже давно выполнена. Математики в течение многих столетий трудились над тем, чтобы обнаружить, изобрести и упростить различные процессы, и теперь благодаря им вы можете легко создавать игры с современным программным обеспечением. Может показаться, что объекты расположены в пространстве случайным образом, но на самом деле каждая игра имеет координатные оси и каждый объект располагается в системе координат (или в сетке).

Что такое D в 3D

Как мы говорили ранее, в каждой игре есть несколько измерений. Наиболее распространены, скорее всего, знакомые вам двумерная и трехмерная системы

(2D и 3D, где D от слова Dimension — измерение). Двумерная система плоская. Здесь у вас есть только вертикальные и горизонтальные элементы (или, иначе говоря, направления вверх, вниз, влево и вправо). Tetris, Pong и Pac Man — отличные примеры 2D-игр. Трехмерная система похожа на двумерную, но в ней, очевидно, на одно измерение больше. В трехмерной системе у вас есть не только горизонталь и вертикаль (направления вверх, вниз, влево и вправо), но еще и глубина (вперед и назад). На рис. 2.1 наглядно показано различие между двумерной плоскостью и трехмерным кубом. Обратите внимание, как добавление оси глубины заставляет трехмерный куб словно «выпирать».

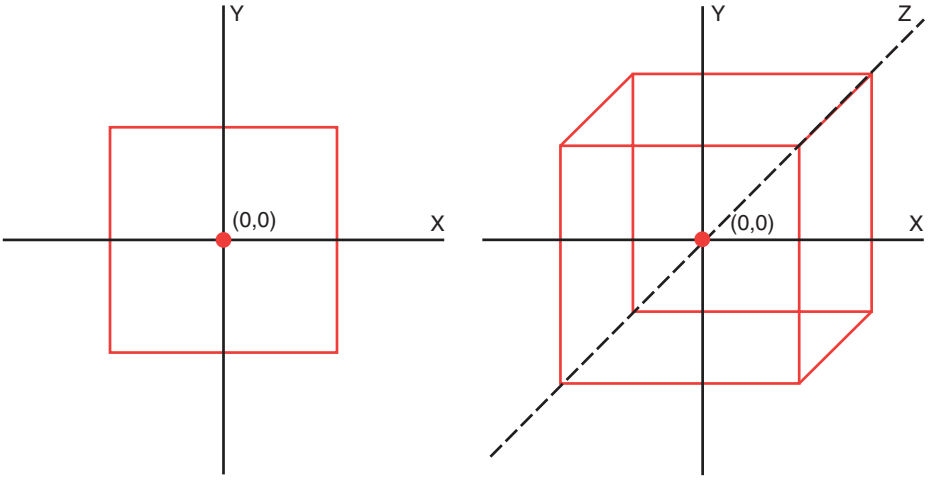


Рис. 2.1. Двумерный квадрат в сравнении с трехмерным кубом

ПРИМЕЧАНИЕ

Узнаем больше о 2D и 3D

Unity — это 3D-движок. Поэтому во всех проектах по умолчанию используются все три измерения. Откроем вам секрет: сегодня уже не существует такого понятия, как истинная 2D-игра. Современные аппаратные средства обрабатывают все как 3D. В 2D-играх ось Z тоже есть, но не используется. Вы можете с удивлением спросить — а зачем мы вообще обсуждаем двумерные системы в этой книге? Правда заключается в том, что даже в 3D-проектах много двумерных элементов. Текстуры, элементы экрана и маппинг работают в двумерной системе. Unity имеет мощный набор инструментов для работы с 2D-играми, и двумерные системы пока еще не канули в Лету.

Использование систем координат

Математический эквивалент системы измерений — система координат. В системе координат используется набор прямых, называемых осями, и позиций,

называемых точками. Эти оси непосредственно соответствуют измерениям, которые они заменяют. Например, в двумерной системе координат есть оси *x* и *y*, которые соответствуют горизонтальному и вертикальному направлению. Если объект движется по горизонтали, мы говорим, что он движется «вдоль оси *x*». Аналогичным образом, в трехмерной системе координат используются оси *x*, *y* и *z* для обозначения горизонтального, вертикального направлений и глубины.

ПРИМЕЧАНИЕ

Общие правила записи координат

Когда речь идет о положении объекта, мы, как правило, перечисляем его координаты. Но говорить, что объект находится в позиции 2 на оси *x* и в позиции 4 на оси *y* — немного громоздко. К счастью, существует сокращенный вариант. В двумерной системе можно записать координаты в виде (*x*, *y*), а в трехмерной (*x*, *y*, *z*). Таким образом, для объекта, который находится в позиции 2 на оси *x* и в позиции 4 на оси *y*, мы пишем (2, 4). Если этот объект также находится в позиции 10 на оси *z*, можно было бы написать (2, 4, 10).

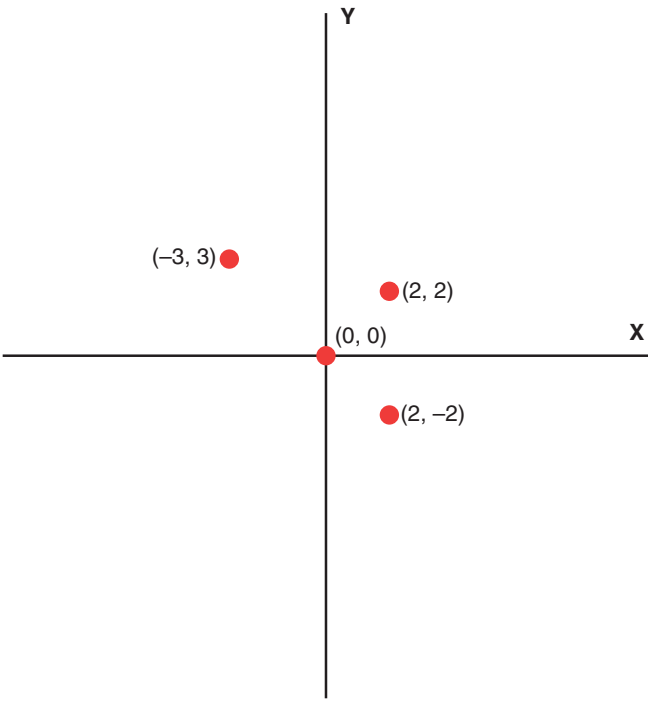


Рис. 2.2. Расположение точек по отношению к началу координат

В каждой системе координат есть точка, в которой все оси пересекаются. Эта точка называется началом координат, и ее координаты всегда (0, 0) в двумерной

системе и (0, 0, 0) — в трехмерной. Это очень важная точка, так как именно от нее отсчитываются все остальные. Координаты любой другой точки — это всего лишь расстояние от начала координат до данной точки вдоль каждой оси. Чем дальше точка от начала координат, тем больше ее координата. Например, если точка перемещается вправо, ее координата *x* становится больше. Если точка перемещается влево, значение координаты *x* становится все меньше, пока точка не пересечет начало координат. После этого значение координаты *x* начинает расти в отрицательную область. Рассмотрим рис. 2.2. В представленной двумерной системе координат определены три точки. Точка (2, 2) на 2 *единицы* удалена от начала координат вдоль осей *x* и *y*. Точка (–3, 3) на 3 единицы левее начала координат и на 3 единицы выше начала координат. Точка (2, −2) смещена на 2 единицы вправо от начала координат и на 2 единицы ниже начала координат.

Глобальные и локальные координаты

Теперь вы узнали об измерениях игрового мира и о системах координат, которые их определяют. То, с чем мы работали до сих пор, называется *глобальной* системой координат. В любой момент времени в глобальной системе координат существует только одна ось *z*, одна ось *y* и одна ось *z*. Точно так же есть одно начало координат, от которого отсчитывается положение всех объектов. Здесь для нас новым будет тот факт, что существует еще так называемая *локальная* система координат. Она уникальна для каждого объекта и полностью обособлена от локальных систем координат других объектов, то есть она имеет собственную ось и начало координат, которые используются только одним конкретным объектом. На рис. 2.3 показаны глобальные и локальные системы координат и отмечены четыре точки, составляющие квадрат, в координатах каждой из этих систем.

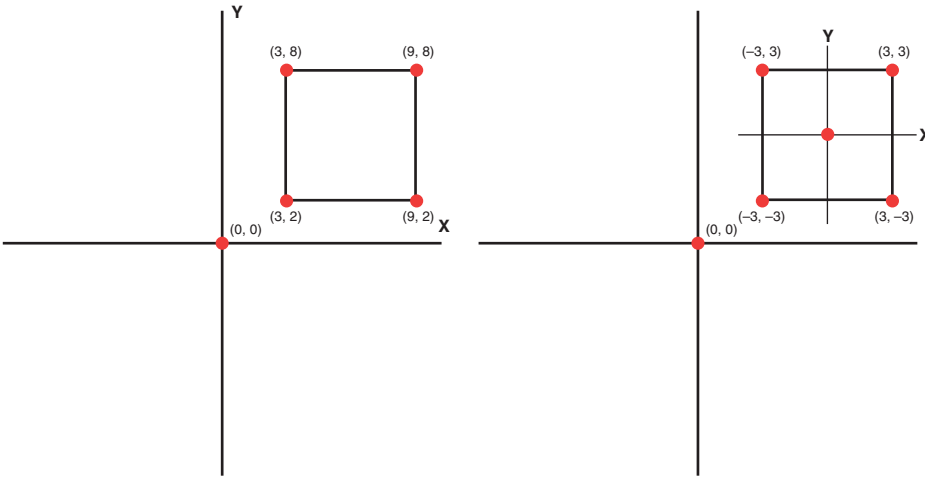


Рис. 2.3. Глобальные координаты (слева) в сравнении с локальными координатами (справа)

Вы можете спросить — зачем нужна локальная система координат, если и так есть глобальная, которая используется для определения положения объектов. Позже в этом часе мы рассмотрим преобразование игровых объектов и наследование признаков у игровых объектов. Оба эти процесса требуют наличия локальной системы координат.

Игровые объекты

Все формы, модели, освещение, камеры, системы частиц и другие элементы в игре Unity имеют общую черту: они являются игровыми объектами, то есть базовыми элементами любой сцены. Несмотря на простоту, они очень мощные. По сути, игровые объекты — это нечто большее, чем преобразования (о них мы более подробно поговорим далее в этом часе) и контейнеры. Контейнер — это часть игрового объекта, предназначенная для хранения различных компонентов, которые делают объект более динамичным и значимым. Что добавлять в ваши игровые объекты — это ваше дело. Существует много компонентов, которые могут внести огромное разнообразие. Читая эту книгу, вы научитесь использовать их.

ПРИМЕЧАНИЕ

Встроенные объекты

Не каждый игровой объект будет изначально пустым. Unity имеет несколько встроенных игровых объектов, которые сразу готовы к использованию. Вы можете найти их, открыв раскрывающийся список **GameObject** в верхней части редактора Unity. Бóльшую часть времени мы будем учиться работать со встроенными и пользовательскими игровыми объектами.

ПРАКТИКУМ

Создание игровых объектов

Давайте уделим некоторое время работе с игровыми объектами. Выполните следующие шаги, чтобы создать несколько основных объектов и изучить их различные компоненты.

Создайте новый проект или новую сцену в существующем проекте.

Добавьте пустой игровой объект, открыв раскрывающийся список **GameObject** и выбрав команду **Create Empty** (вы также можете создать пустой игровой объект, нажав сочетание клавиш **Ctrl+Shift+N** в Windows или **⌘+Shift+N** в macOS).

Посмотрите на панель **Inspector** и обратите внимание, что игровой объект, который вы только что создали, не имеет никаких компонентов, кроме **Transform** (а он есть у каждого объекта). Нажмите кнопку **Add Component** на панели **Inspector**, чтобы увидеть все компоненты, которые вы можете добавить к объекту. Пока не выбирайте ничего.

Добавьте в ваш проект куб. Для этого в раскрывающемся списке **GameObject** установите указатель мыши на пункт **3D Object** и выберите вариант **Cube** из списка.

Обратите внимание на компоненты, которые есть у куба, а у пустого игрового объекта отсутствуют. Компоненты **mesh** делают куб видимым, а компонент **collider** дает ему способность физически взаимодействовать с другими объектами.

Добавьте в свой проект точечный источник света, открыв раскрывающийся список **Create** на панели **Hierarchy** и выбрав пункт **Light** ⇒ **Point Light** из списка.

Обратите внимание, что у точечного источника света только один общий с кубом компонент — **Transform**. В остальном объект полностью ориентирован на то, чтобы испускать свет. Свет, излучаемый точечным источником, дополняет направленный свет, который присутствует на сцене по умолчанию.

Преобразования

Ранее в этом часе вы узнали и изучили различные системы координат и поэкспериментировали с некоторыми игровыми объектами. Теперь соединим одно и другое. При работе с 3D-объектами вы часто слышите термин *преобразование* (англ. — *transform*). Все объекты в 3D-пространстве имеют положение, угол поворота и масштаб. Если объединить эти три параметра, вы получите преобразование объекта. Когда нам понадобится изменить их, мы будем говорить «преобразовать». В Unity это реализовано с помощью компонента **Transform**.

Напомню, что компонент **Transform** — единственный, который есть у каждого игрового объекта, даже у пустого. Благодаря этому компоненту вы можете увидеть текущее состояние объекта и его преобразования. Сейчас это может показаться вам странным, но все довольно просто. Вы разберетесь с этим в кратчайшие сроки. Поскольку преобразование включает в себя положение, угол поворота и масштаб, значит, существуют три способа изменить его: перемещение, вращение и масштабирование. Эти действия можно выполнить с использованием панели **Inspector** или с помощью соответствующих инструментов. На рис. 2.4 и 2.5 показано, какие компоненты и инструменты панели **Inspector** соответствуют различным преобразованиям.

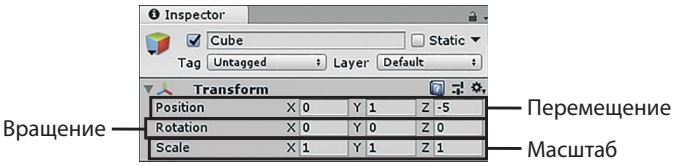


Рис. 2.4. Параметры компонента **Transform** на панели **Inspector**

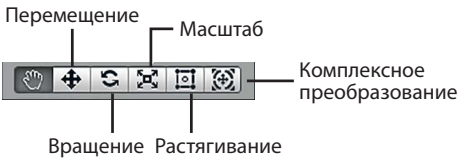


Рис. 2.5. Инструменты преобразования

ПРИМЕЧАНИЕ

Некоторые подробности

Термин Rect (сокращение от Rectangle — прямоугольник) часто встречается у игровых объектов в 2D. Если 2D-объекты, например спрайты, имеют такие же преобразования, как 3D-объекты, вы можете использовать более простой и удобный инструмент **Rect**, чтобы управлять положением, поворотом и масштабированием одновременно. (Расположение инструмента **Rect** приведено на рис. 2.5.) Кроме того, объекты пользовательского интерфейса (UI) в Unity имеют еще один тип преобразования, который называется Rect-преобразованием. В часе 14 мы поговорим о нем более подробно. Сейчас нам достаточно понимать, что это 2D-эквивалент обычного преобразования, который применяется исключительно для редактирования пользовательского интерфейса.

Перемещение

Процесс изменения координат объекта в трехмерной системе называется перемещением, и это самое простое преобразование, которое можно применить к объекту. Когда мы перемещаем объект, он сдвигается вдоль осей. На рис. 2.6 показано перемещение квадрата вдоль оси *x*.

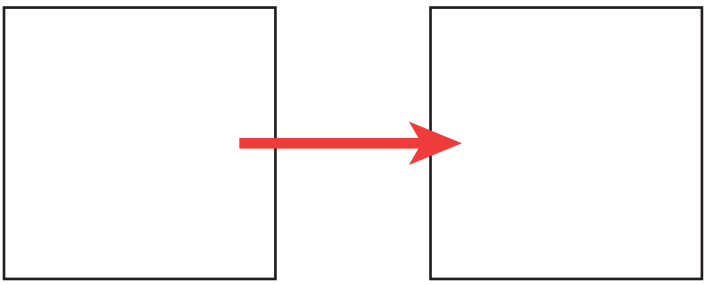


Рис. 2.6. Пример перемещения

Когда вы используете инструмент **Translate** (клавиша **W**) на любом объекте, на панели **Scene** возникают кое-какие изменения. Если конкретно, то появляются три стрелки, направленные от центра объекта по трем осям. Это гизмо перемещения, позволяющий передвигать объекты по сцене. При нажатии и удержании

любой из этих стрелок они окрашиваются в желтый цвет. Затем, если вы подвинете мышь, объект начнет перемещаться вдоль выбранной оси. На рис. 2.7 показан пример гизмо перемещения. Обратите внимание, что гизмо появляются только на панели **Scene**.

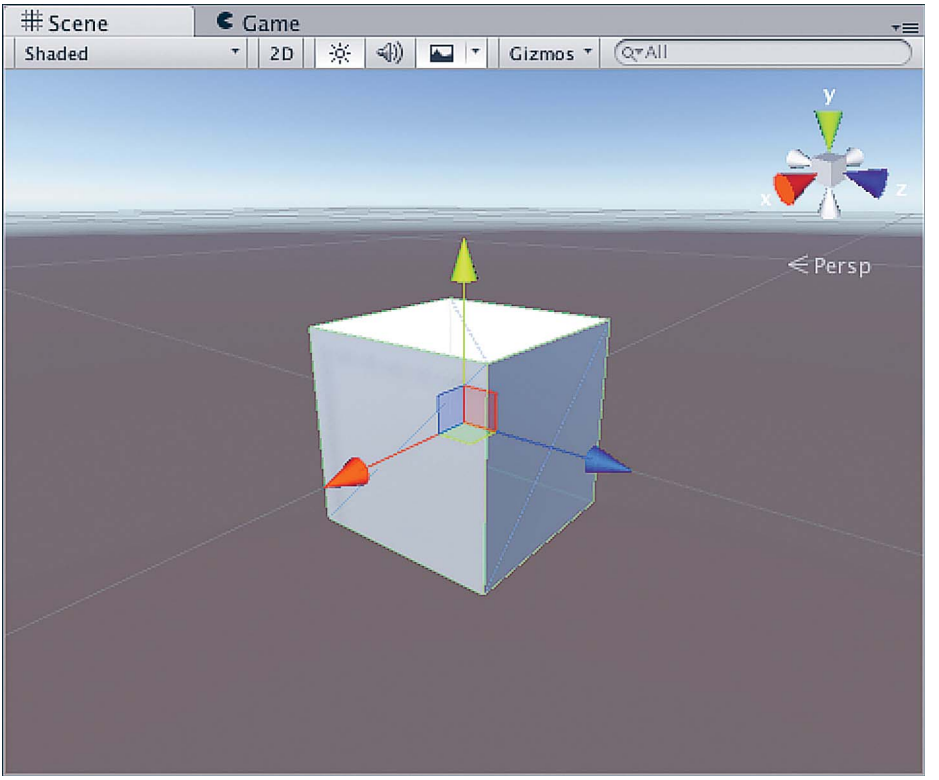


Рис. 2.7. Гизмо перемещения

СОВЕТ

Компонент Transform и инструменты преобразования

В Unity предусмотрено два способа управлять преобразованием объектов. Важно знать, когда использовать каждый из них. При изменении преобразования объекта на панели **Scene** с помощью инструмента преобразования вы также изменяете значения параметров, перечисленных на панели **Inspector**. Часто бывает проще вносить изменения в преобразования объекта с помощью панели **Inspector**: так вы непосредственно указываете требуемые значения. В то же время инструменты преобразования более полезны для внесения оперативных и небольших изменений. Если вы научитесь применять оба способа, ваша работа станет значительно легче.

Вращение

Когда мы вращаем объект, он не перемещается в пространстве. Вместо этого изменяется положение объекта по отношению к пространству. Проще говоря, вращение позволяет переопределить направления осей *x*, *y* и *z* для конкретного объекта. На рис. 2.8 показан пример вращения квадрата вокруг оси *z*.

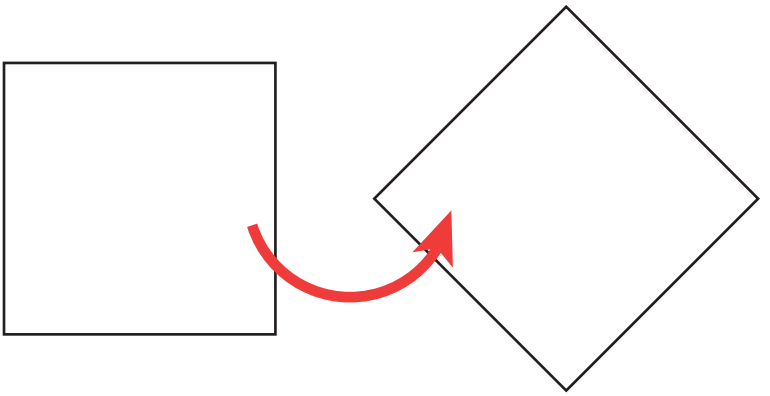


Рис. 2.8. Вращение вокруг оси *z*

СОВЕТ

Определение оси вращения

Если вы не знаете, вокруг которой из осей нужно повернуть объект, чтобы получить желаемый эффект, вы можете использовать простой мысленный трюк. Выберите ось и представьте, что объект зафиксирован на месте с помощью штифта, установленного параллельно этой оси. Объект может вращаться только вокруг этого самого штифта. Теперь определите, какой именно штифт позволит объекту вращаться так, как вам нужно. Это и будет ось, относительно которой вам нужно повернуть объект.

Так же, как и в случае с инструментом **Translate**, вы можете выбрать инструмент **Rotate** (клавиша **E**), чтобы вокруг объекта появился гизмо вращения. Такие гизмо представляют собой окружности вращения объекта вокруг осей. При нажатии и удержании любой окружности она окрашивается в желтый цвет, и после этого вы сможете вращать объект относительно выбранной оси. На рис. 2.9 показано, как выглядит гизмо вращения.

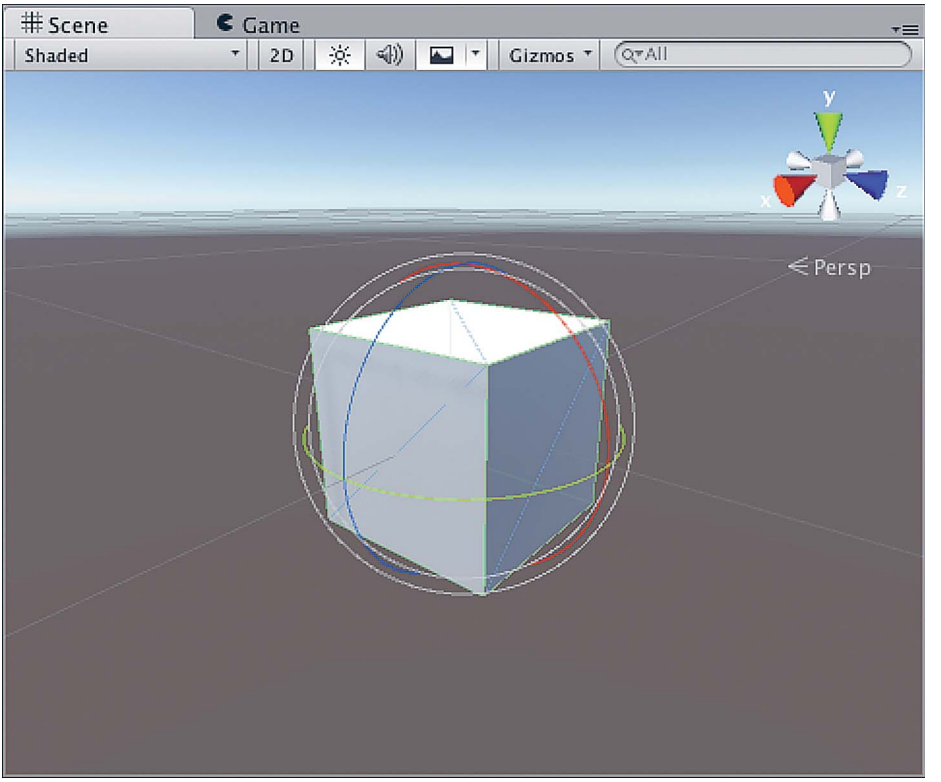


Рис. 2.9. Гизмо инструмента **Rotate**

Масштабирование

Масштабирование — это увеличение или уменьшение объекта в 3D-пространстве. Такое преобразование вполне понятно и просто в использовании. Масштабирование объекта относительно любой оси приводит к изменению размера объекта относительно этой оси. На рис. 2.10 показано уменьшение квадрата относительно осей *x* и *y*. На рис. 2.11 показан гизмо масштабирования, который появляется, когда вы выбираете инструмент **Scaling** (клавиша **R**).

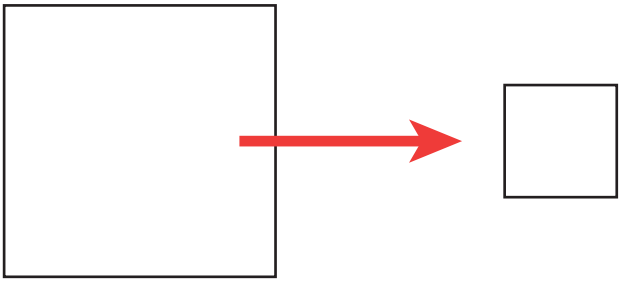


Рис. 2.10. Масштабирование по осям *x* и *y*

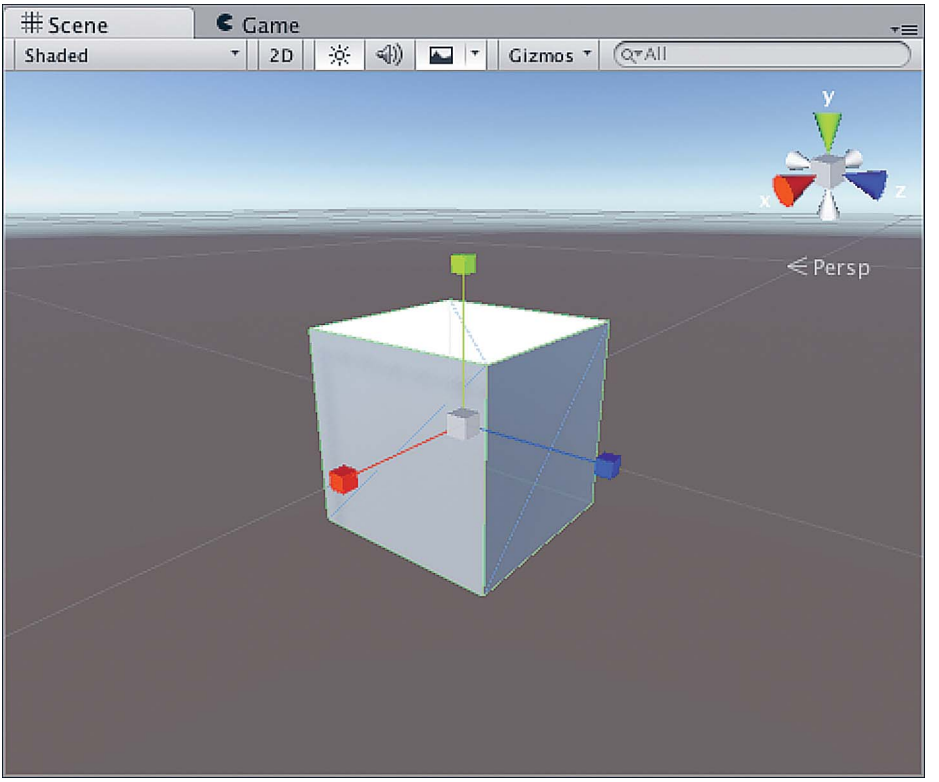


Рис. 2.11. Гизмо масштабирования

Подводные камни преобразований

Как упоминалось ранее в этом часе, в преобразованиях используется локальная система координат. Таким образом, внесенные изменения могут повлиять на будущие преобразования. Например, на рис. 2.12 показано, как одни и те же преобразования, выполненные в обратном порядке, дают в итоге разный результат.

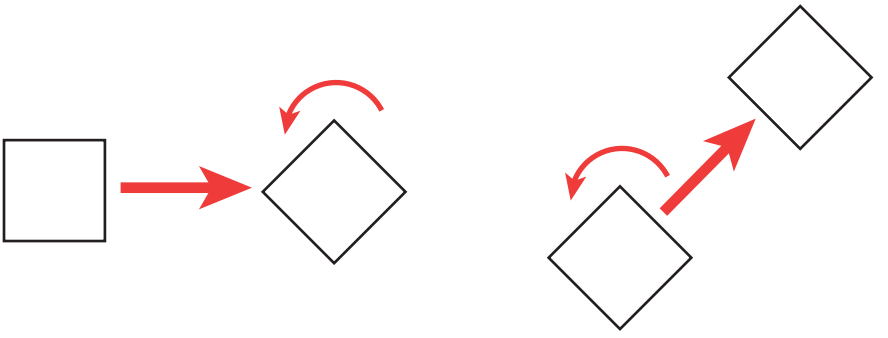


Рис. 2.12. Влияние последовательности выполнения преобразований на результат

Как видно из рисунка, пренебрежение правильным порядком преобразований может привести к неожиданным последствиям. К счастью, результаты преобразований предсказуемые, их можно запланировать.

- Перемещение. Перемещение — это довольно инертное преобразование. То есть любые последующие изменения, как правило, от его результата не страдают.
- Вращение. Вращение изменяет ориентацию локальной системы. Любые перемещения, выполненные после вращения, происходят уже по новым осям. Если вы захотите, например, повернуть объект на 180 градусов вокруг оси z, а затем переместить его в положительном направлении y, то объект будет двигаться вниз, а не вверх.
- Масштабирование. Масштабирование изменяет размер локальной координатной сетки. Это значит, что, масштабируя объект, вы на самом деле увеличиваете его локальную систему координат. В результате объект увеличивается в размере. Это изменение мультипликативно. Например, если объекту задать масштаб 1 (его естественный размер по умолчанию), а затем переместить на 5 единиц вдоль оси x, то объект переместится на 5 единиц вправо. Если же задать объекту масштаб 2, а затем переместить на 5 единиц вдоль оси x, то он визуально переместится на 10 единиц вправо. Это связано с тем, что локальная система координат стала вдвое больше, а $5 \times 2 = 10$. И наоборот, если задать объекту масштаб 0,5, а затем переместить, он визуально переместится на 2,5 единицы ($0{,}5 \times 5 = 2{,}5$).

Когда вы поймете эти правила, вам будет легче вычислить, как именно объект изменится после ваших преобразований. Эффекты зависят от выбора параметра **Local** или **Global**, и к ним придется привыкнуть, поэтому вам стоит поэкспериментировать.

СОВЕТ

Все инструменты в одном флаконе

Последний инструмент сцены на панели инструментов называется **Composite** (клавиша **Y**). Он позволяет одновременно перемещать, вращать и масштабировать 3D-объект.

Расположение гизмо

Настройки расположения гизмо могут показаться новому пользователю непонятными, если он не знает, как они работают. Фактически они определяют положение и выравнивание гизмо преобразований на панели **Scene**. На рис. 2.13 показано, как выравнивание гизмо перемещения изменяется в зависимости от того, выбрали вы локальное или глобальное пространство координат.

Кроме того, на рис. 2.14 показано, как изменяется расположение гизмо, если вы выбрали сразу несколько объектов. Если работает настройка **Pivot**, гизмо располагается по центру первого выбранного объекта. При настройке **Center** гизмо располагается и работает в общем центре всех выбранных объектов.

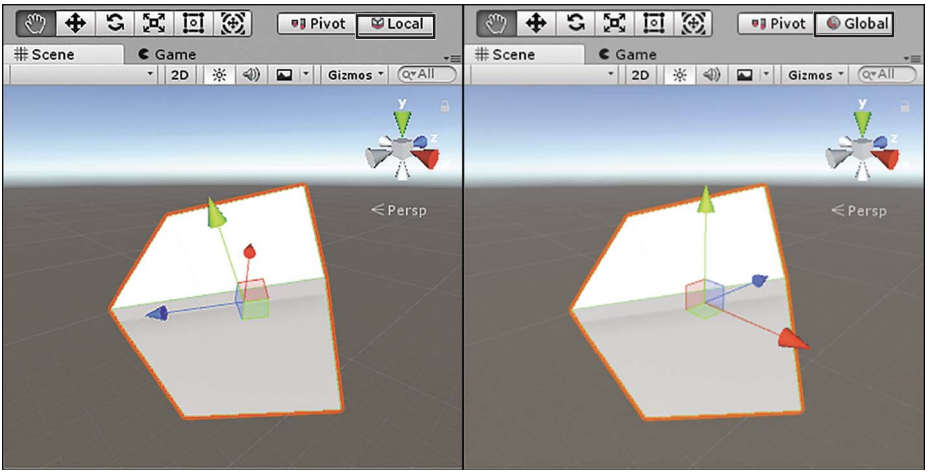


Рис. 2.13. Расположение гизмо при выбранных настройках **Local** и **Global**

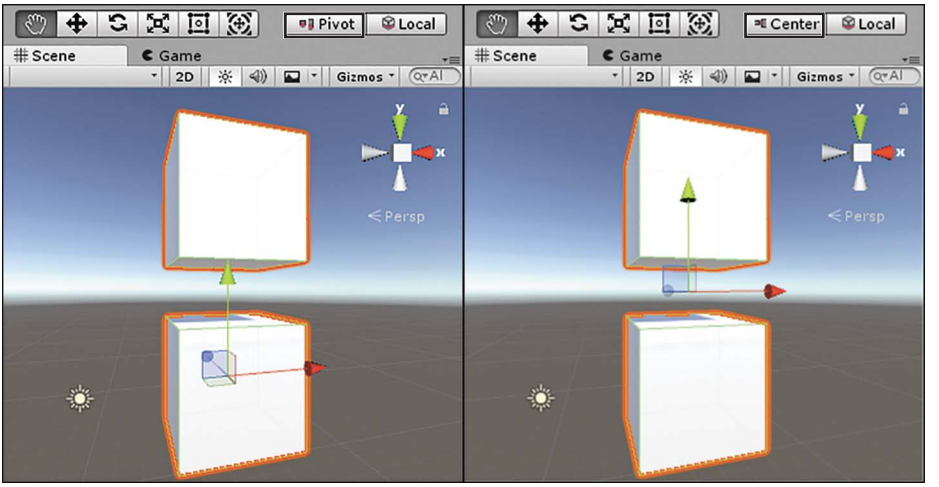


Рис. 2.14. Расположение гизмо при выбранных настройках **Pivot** и **Center**

В целом, вам следует выбирать настройки **Pivot** и **Local**, чтобы избежать путаницы и облегчить свою повседневную работу в Unity.

Преобразования и вложенные объекты

В часе 1 вы узнали, как вложить игровые объекты один в другой на панели **Hierarchy** (путем перетаскивания одного объекта на другой). Напомню, что при этом объект верхнего уровня называется родительским, а другой — дочерним. Преобразования, применяемые к родительскому объекту, работают обычным образом. Объект можно перемещать, масштабировать и вращать. А вот у дочерних объектов есть особенности. У вложенного дочернего объекта преобразования применяются уже относительно родительского объекта, а не относительно глобального пространства. Таким образом, положение дочернего объекта отсчитывается не от начала координат, а от родительского объекта. Если родительский объект вращается, дочерний объект так же перемещается вместе с ним. Но при этом числовые значения углового положения дочернего объекта не изменяются. То же самое касается и масштабирования. При масштабировании родительского объекта дочерний объект тоже изменяется в размерах. Но числовой масштаб дочернего объекта остается прежним. Вас это, наверное, путает. Помните, что преобразования применяются не к объекту, а к системе координат объекта. Поворачивается не объект — поворачивается его система координат. Когда система координат дочернего объекта привязана к локальной системе координат родителя, любые изменения в родительской системе координат непосредственно влияют на дочерний объект (при этом в параметрах дочернего объекта изменений не возникает).

Резюме

В этом часе вы узнали все об игровых объектах в Unity. Сначала мы изучили различия между 2D и 3D. Затем мы поговорили о системах координат и о том, как они математически описывают мир. Затем мы начали работать с игровыми объектами, в том числе с некоторыми из встроенных в Unity объектов. В конце мы изучили три типа преобразований игровых объектов. Вам обязательно нужно поэкспериментировать с преобразованиями, узнать о некоторых возможных опасностях и посмотреть, как преобразования влияют на вложенные объекты.

Вопросы и ответы

Вопрос: **Важно ли понимать, в чем разница между 2D и 3D?**

Ответ: Да. Даже полностью трехмерные игры до сих пор на техническом уровне задействуют некоторые понятия, связанные с 2D.

Вопрос: **Нужно ли мне срочно научиться использовать все встроенные игровые объекты?**

Ответ: Необязательно. Их много, и, если попытаться освоить все сразу, голова пойдет кругом. Не торопитесь и изучайте объекты по мере того, как мы будем обсуждать их здесь.

Вопрос: Как лучше всего познакомиться с преобразованиями?

Ответ: Тут поможет только практика. Продолжайте работать с ними и в конце концов вы привыкнете.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Что означает буква D в сокращениях 2D и 3D?
2. Сколько существует преобразований?
3. Верно или нет: в Unity нет встроенных объектов, и вам придется создавать свои.
4. Предположим, у вас есть объект в позиции с координатами (0, 0, 0). Каким станет положение при перемещении его на 1 единицу по оси *x* и дальнейшем повороте на 90 градусов вокруг оси *z*? Что произойдет, если мы сначала повернем его, а только потом переместим?

Ответы

1. Dimenson — измерение.
2. Три.
3. Нет. В Unity есть множество встроенных объектов.
4. В первом случае объект окажется в позиции (1, 0, 0). А если мы сначала повернем, а потом переместим его, он окажется в положении (0, 1, 0).

Если у вас остались вопросы, вернитесь к разделу «Подводные камни преобразований».

Упражнение

Поэкспериментируйте с преобразованиями объектов, находящихся в отношениях родитель/ребенок. Это поможет вам лучше понять, как именно системы координат влияют на положение объектов.

1. Создайте новую сцену или проект.
2. Добавьте на сцену куб и поместите его в позицию с координатами (0, 2, −5). Самое время вспомнить о сокращенной записи координат: добавленный вами куб должен иметь положение вдоль оси *x*, равное 0, вдоль оси *y* — равное 2, и вдоль оси *z* — равное −5. Вы можете легко установить эти значения в поле компонента **Transform** на панели **Inspector**.
3. Добавьте на сцену сферу. Обратите внимание на значения *x*, *y* и *z* этой сферы.
4. Сделайте сферу вложенным объектом, переместив ее в куб на панели **Hierarchy**. Обратите внимание, как изменились значения. Положение сферы теперь отсчитывается от куба.
5. Поместите сферу в позицию с координатами (0, 1, 0). Обратите внимание, что она не поднимается вверх от начала координат, а вместо этого располагается над кубом.
6. Поэкспериментируйте с различными преобразованиями. Не забудьте попробовать их и на кубе, и на сфере и посмотрите, как они по-разному работают для родительского и дочернего объекта.

3-Й ЧАС

Модели, материалы и текстуры

Что вы узнаете в этом часе

- Основную информацию о моделях
- Как импортировать пользовательские и готовые модели
- Как работать с материалами и шейдерами

В этом часе вы узнаете все о моделях и о том, как они используются в Unity. Мы начнем с рассмотрения фундаментальных принципов мешей и 3D-объектов. Затем вы узнаете, как импортировать свои собственные модели или использовать модели из магазина Asset Store. В конце часа мы изучим работу с материалами и шейдерами в Unity.

Общая информация о моделях

Видеоигры без графических компонентов вряд ли назывались бы видеоиграми. В 2D-играх графика построена из плоских изображений, называемых спрайтами. Здесь достаточно лишь перемещать спрайты по осям *x* и *y* и последовательно прокручивать их, и зрителю будет казаться, что он видит истинное движение и анимацию. В 3D-играх все чуть сложнее. При наличии третьей оси координат объекты должны иметь объем, чтобы обмануть глаз зрителя. В играх используется большое количество объектов, поэтому они должны, в числе прочего, быстро обрабатываться. Это возможно благодаря использованию мешей. Меш в самом простом его виде представляет собой набор взаимосвязанных треугольников. Они выстраиваются один за другим в полосы, образуя как простые, так и весьма сложные объекты. Эти полосы создают трехмерное определение модели и очень быстро обрабатываются. Впрочем, это не ваша забота: Unity обрабатывает все за вас. Позже в данном часе мы рассмотрим, как треугольники составляют различные формы на панели **Scene** в Unity.

ПРИМЕЧАНИЕ

Почему именно треугольники?

Вы можете спросить: почему 3D-объекты целиком составляются из треугольников? Ответ прост: компьютеры обрабатывают графические данные как набор точек, называемых вершинами. Чем меньше у объекта вершин, тем быстрее он отрисовывается. Треугольники имеют два свойства, дающие им явное преимущество. Во-первых, если у вас есть треугольник, вам достаточно добавить всего одну вершину, чтобы сделать еще один. Таким образом, чтобы сделать один треугольник, вам понадобятся три вершины, для двух — всего четыре, а для трех нужно пять вершин. Благодаря этому треугольники очень эффективны. Второе свойство, говорящее в их пользу, заключается в том, что из полос треугольников можно смоделировать любой 3D-объект. Ни одна другая форма не дает такого высокого уровня гибкости и эффективности.

ПРИМЕЧАНИЕ

Разбираемся с терминами: «модель» или «меш»?

Термины *модель* и *меш* похожи, и их зачастую можно использовать как синонимы. Однако между ними существует разница. Меш содержит все точки и линии, которые определяют форму 3D-объекта. Когда вы обращаетесь к форме 3D-объекта или форме модели, вы на самом деле взаимодействуете с мешем. Меш также можно назвать *географией модели*, или *гео*. Модель, следовательно, является объектом, в который входит меш.

Меш в модели нужен для определения ее размеров, но, помимо этого, модель включает в себя анимацию, текстуры, материалы, шейдеры и другие меши. Общее правило состоит в следующем: если рассматриваемый элемент не содержит ничего, кроме информации о вершинах, то это меш. В противном случае — это модель.

ПРИМЕЧАНИЕ

А что насчет 2D?

В этом часе будет приведено много информации о визуализации 3D-объектов. Это, безусловно, полезно, но как быть, если вы хотите делать только 2D-игры? Надо ли вам тогда тратить время на этот урок (и все остальные, где не говорится о 2D)? Ответ — конечно, да! Как уже упоминалось в часе 2, истинных 2D-игр больше не существует. Спрайты — это всего лишь текстуры, наложенные на плоские 3D-объекты. Освещение применяется к 2D-объектам. Даже камеры, используемые для 2D- и 3D-игр, одинаковые. Все понятия, которые мы изучим здесь, можно применять непосредственно к вашим 2D-играм, так как на самом деле они 3D-игры. Когда вы побольше поработаете с Unity, вы поймете, что грань между 2D и 3D крайне размыта!

Встроенные 3D-объекты

В Unity есть несколько встроенных основных мешей (или примитивов), с которыми вы можете работать. Это, как правило, простые формы, используемые для побочных задач, или объединяемые для создания более сложных объектов. На рис. 3.1 показаны доступные встроенные меши. (В предыдущих часах мы уже работали с кубом и сферой.)

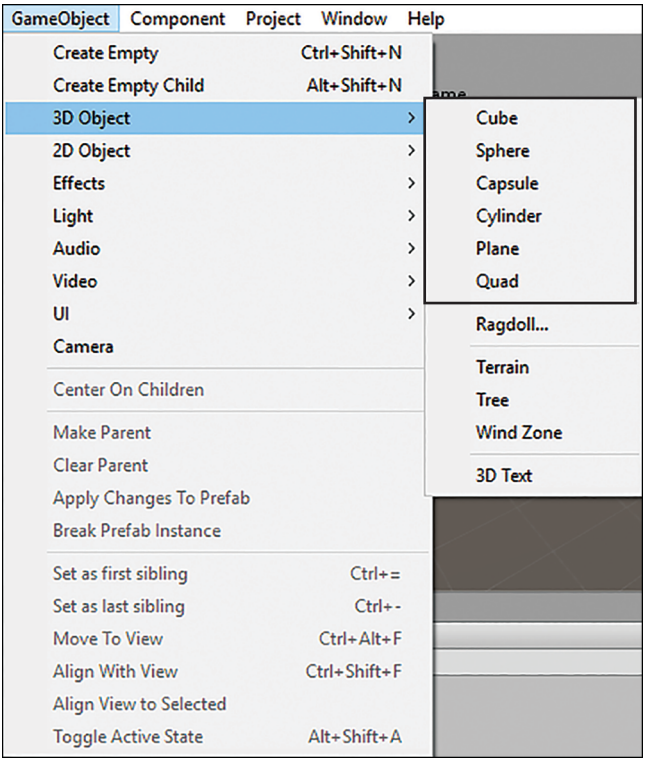


Рис. 3.1. Встроенные меши в Unity

СОВЕТ

Моделирование с использованием простых мешей

Как быть, если в вашей игре нужен сложный объект, но вы не можете найти подходящую модель? С помощью создания вложенных объектов в Unity вы можете легко делать простые модели, используя встроенные меши. Разместите меши рядом друг с другом так, чтобы они выглядели примерно так, как вам нужно. Затем привяжите все объекты к одному центральному. Таким образом, при перемещении родителя все дети будут перемещаться вместе с ним. Это, возможно, не самый красивый способ создавать игровые модели, но все же он крайне полезен и удобен для прототипирования!

Импортирование моделей

Встроенные модели — это хорошо, но в большинстве случаев вашим играм будут нужны более сложные художественные средства. К счастью, в Unity довольно легко импортировать в проект собственные 3D-модели. Для этого поместите файл, содержащий 3D-модель, в папку *Assets*, и модель будет добавлена в проект. Оттуда вы сможете переместить ее на сцену или на панель **Hierarchy** и построить игровой объект вокруг нее. Unity поддерживает расширения моделей *.fbx*, *.dae*, *.3ds*, *.dxf* и *.obj*. Поэтому вы можете работать в любой программе для 3D-моделирования.

ПРАКТИКУМ

Импорт собственных 3D-моделей

Выполните следующие действия, чтобы импортировать пользовательский 3D-объект в проект Unity.

1. Создайте новый проект Unity или новую сцену.
2. На панели **Project** создайте новую папку с названием **Models** в папке **Assets**. Для этого щелкните правой кнопкой мыши по папке **Assets** и выберите команду **Create** ⇒ **Folder**.
3. В прилагаемых к книге файлах найдите файл *Torus.fbx* в папке *Час 3*.
4. Используя файловый менеджер вашей операционной системы и редактор Unity совместно, перетащите файл *Torus.fbx* в папку *Models*, созданную на шаге 2. В Unity откройте папку **Models**, где вы увидите вновь появившийся файл **Torus**. Если вы сделали все правильно, панель **Project** будет выглядеть так, как показано на рис. 3.2. (Примечание: если вы используете более раннюю версию Unity или изменяли настройки редактора, у вас по умолчанию может быть создана папка **Materials**. Если это так, беспокоиться не о чем. Более подробно о материалах мы поговорим позже в этом часе.)

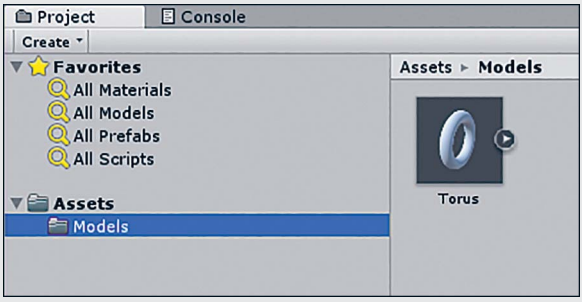


Рис. 3.2. Вид панели **Project** после добавления модели **Torus**

5. Перетащите ассет **Torus** из папки **Models** на панель **Scene**. Обратите внимание, что у добавленного на сцену игрового объекта **Torus** есть фильтр меша и рендер меша. Они позволяют модели появиться на экране.
6. В настоящее время тор на нашей сцене очень мал, поэтому выберите ассет **Torus** на панели **Project** и взгляните на панель **Inspector**. Измените значение **scale** с 1 на **100** и нажмите кнопку **Apply** в нижней правой части панели **Inspector**.

ВНИМАНИЕ!

Дефолтное масштабирование меша

Большинство параметров, касающихся мешей, на панели **Inspector** относится к более продвинутым, и нам пока рано говорить о них. Сейчас нас интересует параметр **Scale**. В программах для работы с 3D-моделями используются общие единицы, но в каждой программе может быть свой масштаб. По умолчанию Unity обрабатывает общую единицу как один метр; другое программное обеспечение, например Blender, может обрабатывать ее как сантиметр. Обычно, когда вы импортируете модель, Unity постарается угадать масштаб в зависимости от типа файла. Иногда, однако (как в данном случае), это не работает, но вы можете настроить масштаб с помощью масштабного коэффициента. Изменяя значение масштабного коэффициента от 1 до 100, вы тем самым инструктируете Unity импортировать модель в 100 раз больше, преобразуя сантиметры модели в метры Unity.

Модели и Asset Store

Совершенно не обязательно быть экспертом в моделировании, чтобы делать игры в Unity. Магазин Asset Store — это простой и эффективный способ найти готовые модели и импортировать их в свои проекты. Модели в Asset Store могут быть бесплатные или платные, могут поставляться по одной или в коллекции аналогичных моделей. Некоторые модели имеют собственные текстуры, в других содержатся только данные мешей.

ПРАКТИКУМ

Загрузка моделей из Asset Store

Давайте посмотрим, как найти и загрузить модели из Asset Store в Unity. Выполните следующие действия, чтобы загрузить модель под названием **Robot Kyle** и импортировать ее на вашу сцену.

1. Создайте новую сцену (выбрав команду меню **File** ⇒ **New Scene**). На панели **Project** введите текст **Robot Kyle t: Model** в строку поиска (рис. 3.3).

Рис. 3.3. Поиск ассета модели

2. В разделе **Search filter** нажмите кнопку **Asset Store** (рис. 3.3). Если кнопка не видна, вам, возможно, потребуется изменить размер окна редактора или панели **Project**. У вас обязательно должно быть подключение к Интернету.
3. Найдите модель **Robot Kyle**, размещенную компанией Unity Technologies. Вы можете увидеть название опубликовавшей ассет компании, щелкнув по нему мышью и посмотрев на панель **Inspector** (рис. 3.4).

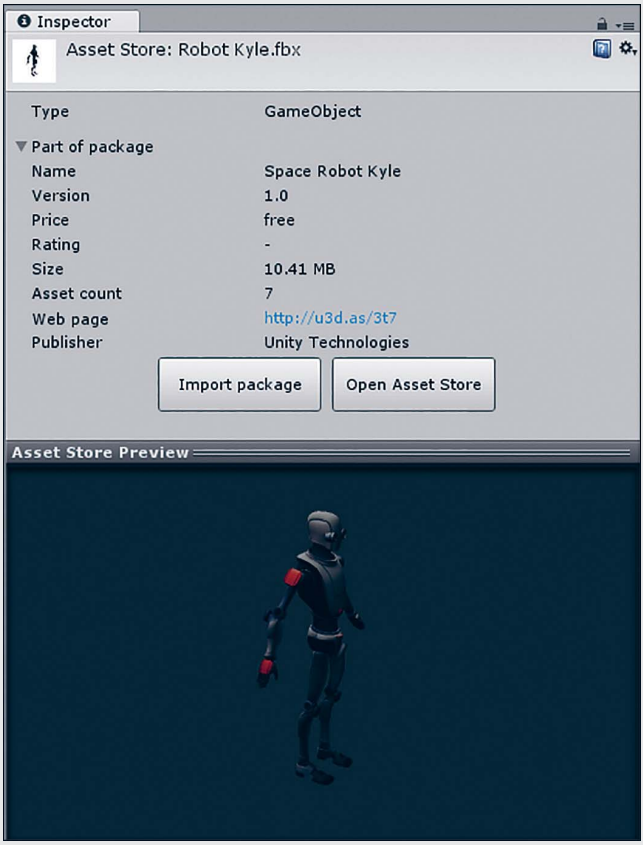


Рис. 3.4. Панель **Inspector** с информацией о данном ассете

4. На панели **Inspector** нажмите кнопку **Import Package**. После этого вам может быть предложено ввести данные вашей учетной записи Unity. В случае, если у вас возникнут проблемы на данном этапе, в файлах примеров к этому часу вы найдете нужный. Дважды щелкните по нему, чтобы импортировать.
5. Когда откроется диалоговое окно **Importing Package**, оставьте все флажки установленными и нажмите кнопку **Import**.
6. Найдите модель робота в папке *Assets\Robot Kyle\Model* и перетащите ее на панель **Scene**. Обратите внимание, что модель на сцене будет выглядеть довольно маленькой, и вам, возможно, потребуется изменить масштаб, чтобы рассмотреть ее.

Текстуры, шейдеры и материалы

Применение графических ассетов к 3D-моделям может представлять сложность для новичков, которые совсем не знакомы с этим процессом. В Unity используется простая и понятная последовательность операций, которая дает вам массу возможностей выбрать, как будет выглядеть конечный результат. Графические ассеты подразделяются на текстуры, шейдеры и материалы. Мы рассмотрим каждый тип отдельно в соответствующих разделах, а на рис. 3.5 показано, как они сочетаются друг с другом. Обратите внимание, что текстуры не применяется непосредственно к моделям. Вместо этого текстуры и шейдеры применяются к материалам. А вот материалы, в свою очередь, применяются к моделям. Это позволяет быстро и аккуратно изменять внешний вид модели без особых трудностей.

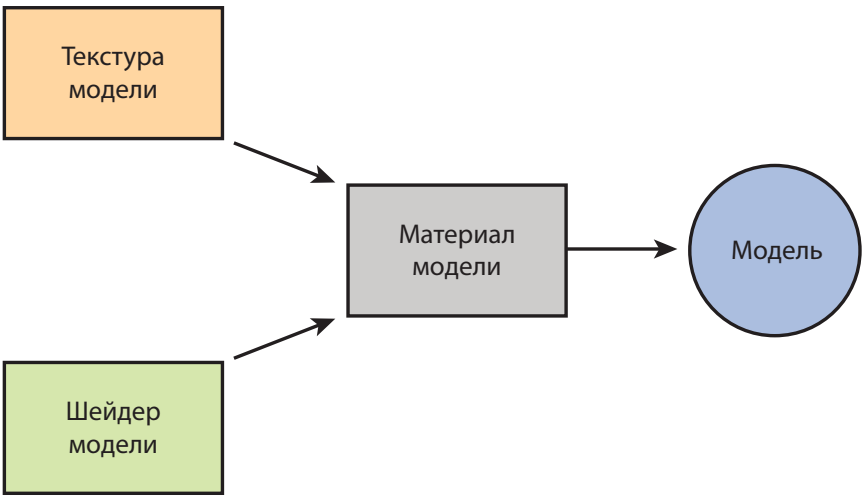


Рис. 3.5. Компоновка модели из графических ассетов

Текстуры

Текстуры — это плоские изображения, которые накладываются на 3D-объекты. Они придают моделям красочность и необычность, выступая против пустоты и скучности. Поначалу может показаться странным, что 2D-изображение можно наложить на 3D-модель. Но, как только вы поймете, что к чему, все окажется довольно просто. Представьте на секунду банку с консервами. Если вы оторвете этикетку от банки, вы увидите, что это всего лишь плоский кусок бумаги. Этикетка играет роль текстуры: обернутая вокруг трехмерной банки, она обеспечивает ей более приятный внешний вид.

Добавить текстуры в проект Unity так же легко, как и любой другой ассет. Сначала создайте папку для хранения ваших текстур (имя *Textures* отлично подойдет!). Затем перетащите в эту папку все текстуры, которые нужны в вашем проекте. Вот и все!

ПРИМЕЧАНИЕ

Это всего лишь развертка!

Довольно легко представить, как текстура оборачивается вокруг банки, но как быть с более сложными объектами? При создании сложной модели обычно генерируется нечто под названием «*развертка*». Развертка — это что-то вроде карты, которая показывает, как именно плоская текстура оборачивается вокруг модели. Если вы заглянете в папку *Textures* в папке *Robot Kyle*, о которой мы говорили ранее, вы найдете там текстуру *Robot_Color*. Она выглядит странно, но это не что иное, как развернутая текстура для модели. Генерация разверток, создание моделей и текстур — это отдельная форма искусства, о которой мы не будем говорить здесь подробно. Примерного представления о том, как это работает, должно хватить вам для изучения книги.

ВНИМАНИЕ!

Странные текстуры

Позже в этом часе мы научимся накладывать текстуры на модели. Вы можете заметить, что текстуры слегка деформируются или поворачиваются не в том направлении. Знайте, что это не ошибка. Подобное происходит, когда вы берете базовую прямоугольную 2D-текстуру и накладываете ее на модель. Модель не имеет представления о том, как «надеть» эту текстуру правильно, поэтому она накладывает ее «как получится». Чтобы избежать этой проблемы, используйте текстуры, специально предназначенные (то есть правильно развернутые) для модели, которую вы используете.

Шейдеры

Текстура модели определяет, что нарисовано на ее поверхности, а шейдер — то, *как* это рисуется. Давайте представим процесс таким образом: материал — интерфейс между вами и шейдером. Материал говорит вам, что нужно шейдеру, чтобы

отрисовать объект, и вы предоставляете ему эти элементы, чтобы все выглядело так, как вы хотите. Сейчас это, возможно, звучит бессмысленно, но позже, когда вы будете создавать материалы, вы начнете понимать, как они работают. Мы более подробно поговорим о шейдерах позже в этом часе, потому что у вас не получится использовать шейдеры без материала. На самом деле большая часть сведений, которые нужно знать о материалах, — это скорее информация о шейдерах материалов.

СОВЕТ

Мысленный эксперимент

Если вы не совсем хорошо понимаете, как работает шейдер, представьте следующее: вот у вас есть кусок дерева. Физическая форма дерева — это его меш; цвет, текстура и видимые элементы — это текстура. Теперь возьмите этот кусок дерева и намочите его. Меш не изменился. Кусок по-прежнему сделан из того же вещества (дерево). Но в то же время он выглядит по-другому — он стал немного темнее и более блестящим. В этом примере мы рассмотрели два «шейдера»: сухая древесина и сырая древесина. В «шейдере» сырой древесины добавлено что-то такое, что изменило внешний вид модели, хотя и сама модель, и текстура остались прежними.

Материалы

Как упоминалось ранее, материалы — это, в общем-то, контейнеры для шейдеров и текстур, которые могут быть применены к моделям. Большая часть настроек материалов зависит от того, какой выбран шейдер, однако у всех шейдеров есть некоторые общие функциональные возможности.

Чтобы изготовить новый материал, начните с создания папки **Materials**. Затем щелкните по ней правой кнопкой мыши и выберите команду меню **Create** ⇒ **Material**. Присвойте вашему материалу описательное имя — и дело сделано!

На рис. 3.6 представлены два материала с различными параметрами шейдеров. Обратите внимание, что в обоих случаях используется один и тот же шейдер **Standard**. У обоих материалов базовое альбедо имеет белый цвет (подробнее об альбедо мы поговорим позже), а вот настройки параметра **Smoothness** — различные. Материал с названием **Flat** имеет низкое значение параметра **Smoothness**, поэтому его освещение выглядит равномерно, так что свет отражается во множестве направлений. Материал с названием **Shiny** имеет более высокое значение параметра **Smoothness**, что создает более сфокусированное отражение света. У обоих материалов есть область предварительного просмотра, где вы можете увидеть, как этот материал будет выглядеть непосредственно на модели.

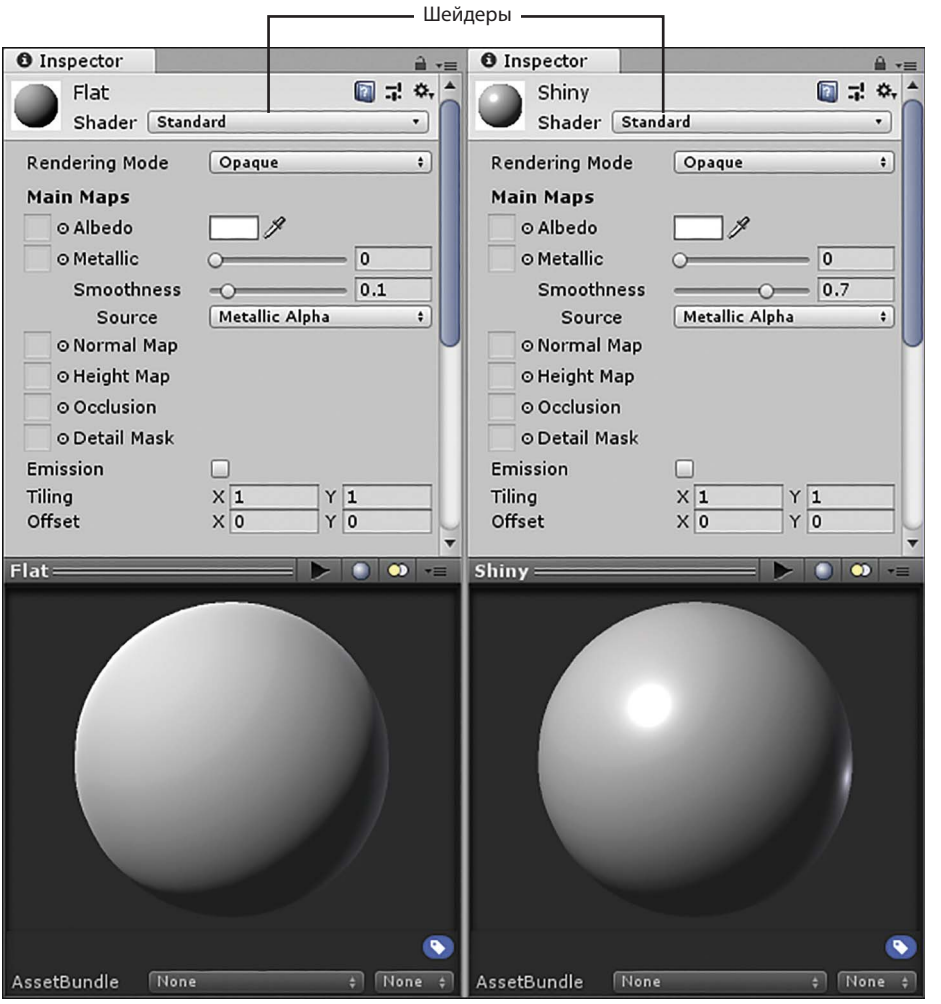


Рис. 3.6. Два материала с различными настройками шейдера

И снова о шейдерах

Теперь, когда мы познакомились с текстурами, моделями и шейдерами, настало время узнать, как все это работает вместе. В Unity есть очень мощный встроенный шейдер **Standard**, который будет использоваться в этой книге.

В таблице 3.1 описаны общие параметры шейдеров. Помимо перечисленных, существует много других параметров шейдера **Standard**, но мы в этой книге сосредоточимся на работе с теми, которые приведены в таблице.

ТАБЛ. 3.1. Общие параметры шейдера

Параметр	Описание
Albedo	Параметр **Albedo** определяет базовый цвет объекта. Благодаря встроенной в Unity системе PBR (Physically Based Rendering — физически корректный рендеринг) этот цвет взаимодействует со светом так же, как и реальный объект. Например, желтый альбедо будет выглядеть желтым при белом освещении, но зеленым — при синем освещении. Здесь было бы полезно использовать текстуру, которая содержит цветовую информацию о вашей модели
Metallic	Этот параметр определяет буквально то, что следует из названия: насколько металлически выглядит материал. Параметр может принимать на вход текстуру, используя ее как «карту» металлических свойств разных частей модели. Для получения реалистичной картинки установите этот параметр равным 0 или 1
Smoothness	Параметр **Smoothness** основной в PBR, поскольку он включает в себя различные микродефекты, мелкие детали, метки и возраст, который определяет степень гладкости (или шероховатости) поверхности. В результате полученная модель выглядит более или менее блестящей. Этот параметр использует ту же карту текстуры, как и **Metallic**. Для получения реалистичной картинки установите этот параметр равным 0 или 1
Normal map	Параметр **Normal Map** — это карта нормалей, которая будет применена к модели. Карта нормалей может использоваться для создания рельефа или бугорков. Этот параметр полезен при расчете освещения, чтобы более детально проработать модель
Tiling	Свойство **Tiling** определяет, как часто текстура повторяется на модели. Она может повторяться по осям *x* и *y*. (Напомню, что текстуры плоские, поэтому ось *z* в данном случае отсутствует)
Offset	Свойство **Offset** определяет, с какого положения *x* и *y* текстуры начинает применяться шейдер

Может показаться, что такой объем информации будет трудновато переварить, но как только вы чуть лучше познакомитесь с основами текстур, шейдеров и материалов, вы увидите, что значение параметра **Smoothness** вашего материала станет приближаться к 1.

В редакторе Unity есть несколько других шейдеров, которые мы в этой книге не рассматриваем. Шейдер **Standard** весьма гибок, и его должно быть достаточно для удовлетворения большинства ваших основных потребностей.

ПРАКТИКУМ

Применение к моделям текстур, шейдеров и материалов

Выполните предложенное задание, чтобы обобщить свои знания о текстурах, шейдерах и материалах и создать пристойную кирпичную стену.

1. Создайте новый проект или новую сцену. Обратите внимание, что при создании нового проекта редактор закрывается и повторно открывается.
2. Создайте папки **Textures** и **Materials**.
3. Найдите файл *Brick_Texture.png* среди файлов примеров и перетащите его в папку **Textures**, созданную вами на шаге 2.
4. Добавьте на сцену куб. Разместите его в позиции с координатами (0, 1, –5). Задайте значения масштабирования (5, 2, 1). На рис. 3.7 показаны свойства куба.

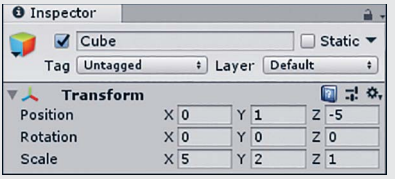


Рис. 3.7. Свойства куба

5. Создайте новый материал (для этого щелкните правой кнопкой мыши по папке **Materials** и выберите пункт **Create** ⇒ **Material**) и присвойте ему имя **BrickWall**.
6. Оставьте выбранным шейдер **Standard**, а в поле **Main Maps** нажмите на круговой селектор круга (маленький кружок) слева от слова **Albedo**. Выберите пункт **Brick_Texture** из списка.
7. Нажмите и перетащите созданный материал кирпичной стены из панели **Project** на куб на панели **Scene**.
8. Обратите внимание, что текстура слишком сильно растягивается по стене. Выберите материал, а затем присвойте параметру **Tiling** оси *x* значение **3**. Убедитесь, что вы делаете это в разделе **Main Maps** (Основные карты), а не **Secondary Maps** (Дополнительные карты). Теперь стена выглядит намного лучше. Итак, на вашей сцене появилась текстурированная кирпичная стена. На рис. 3.8 показан конечный результат.

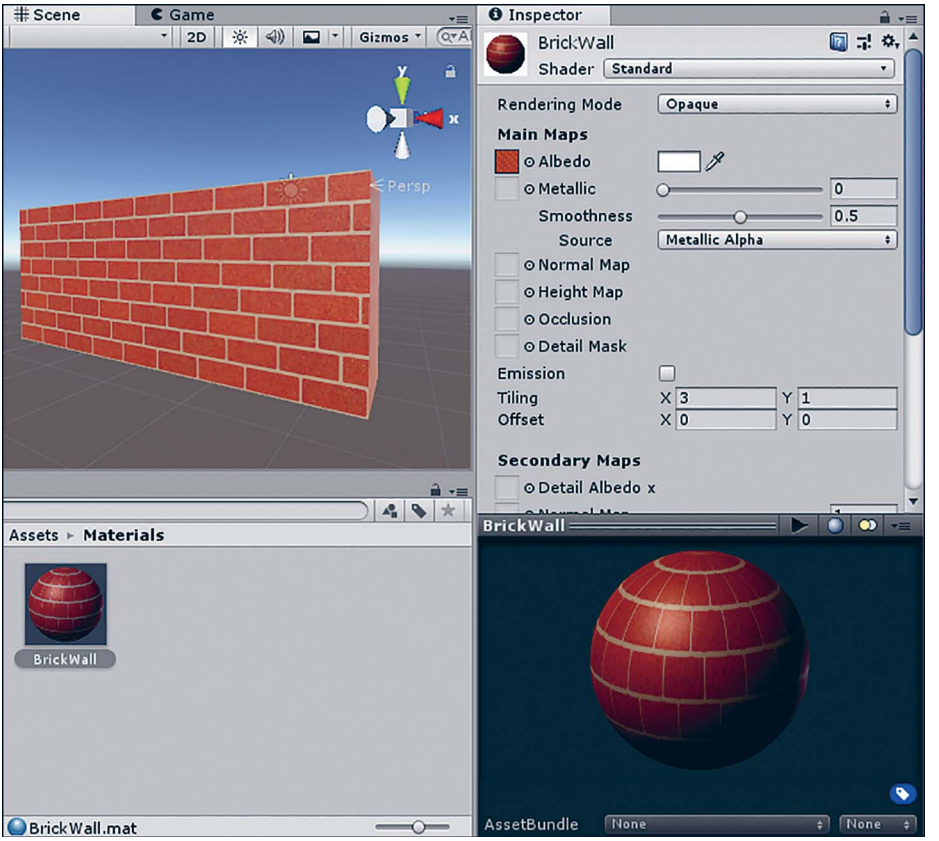


Рис. 3.8. Вот что должно получиться после выполнения практикума

Резюме

В этом часе вы узнали все о моделях в Unity. Сначала вы получили представление о том, как модели формируются из набора вершин, называемого мешем. Затем вы узнали, как использовать встроенные модели, импортировать собственные, а также загружать модели из магазина Asset Store. Затем мы изучили процесс работы с моделями в редакторе Unity. Мы поэкспериментировали с текстурами, шейдерами и материалами. Завершился этот час созданием текстурированной кирпичной стены.

Вопросы и ответы

Вопрос: **Могу ли я создавать игры, если не умею рисовать?**

Ответ: Разумеется. Благодаря наличию бесплатных ресурсов в Интернете и магазине Unity Asset Store вы можете найти различные художественные ассеты и использовать их в ваших играх.

Вопрос: **Нужно ли мне знать, как использовать все встроенные шейдеры?**

Ответ: Необязательно. Многие шейдеры крайне специфичны и применяются редко. Начните с шейдера **Standard**, о котором мы говорили в этом уроке, после чего вы всегда можете узнать больше о других, если того потребует ваш игровой проект.

Вопрос: **В магазине Unity Asset Store платные художественные ассеты — значит ли это, что я могу продавать свои художественные ассеты?**

Ответ: Да, именно так. Магазин Asset Store не ограничивается имеющимися в нем художественными ассетами. Если вы создадите ассеты высокого качества, вы, разумеется, сможете продавать их в магазине.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Верно или нет: благодаря своей простоте, именно квадраты используются для построения мешей в моделях.
2. Какие форматы файлов 3D-моделей поддерживает редактор Unity?
3. Верно или нет: из магазина Unity Asset Store можно загрузить только платные модели.
4. Объясните взаимосвязь между текстурами, шейдерами и материалами.

Ответы

1. Нет. Меши строятся из треугольников.
2. Это форматы.*fbx*,.d*ae*, *.3ds*,.d*xf* и.*obj*.
3. Нет. В магазине Asset Store есть много бесплатных моделей.

4. Материалы — это контейнеры для текстур и шейдеров. Шейдеры определяют свойства, которые задаются с помощью материала, и то, как материал отображается.

Упражнение

В этом упражнении мы предлагаем вам поэкспериментировать с тем, как шейдеры могут повлиять на внешний вид модели. Для всех моделей мы будем использовать один и тот же меш и текстуру, а различаться будут только шейдеры. Проект, созданный в этом упражнении, называется **Hour 3_Exercise** и находится в файлах примеров в папке *Час 3*.

1. Создайте новую сцену или проект.
2. Добавьте в ваш проект папки **Materials** и **Textures**. Найдите в файлах примеров к часу 3 файлы *Brick_Normal.png* и *Brick_Texture.png* и перетащите их в папку **Textures**.
3. На панели **Project** выберите текстуру **Brick_Texture**. На панели **Inspector** присвойте параметру **Aniso Level** значение **3**, чтобы повысить качество отрисовки текстуры на изгибах. Нажмите кнопку **Apply**.
4. На панели **Project** выберите пункт **Brick_Normal**. На панели **Inspector** измените тип текстуры на **Normal Map**. Нажмите кнопку **Apply**.
5. На панели **Hierarchy** выделите объект **Directional Light** и задайте его положение в позиции с координатами (0, 10, −10) и угол вращения (30, −180, 0).
6. Добавьте в ваш проект четыре сферы. Задайте каждой из них масштаб (2, 2, 2). Расставьте сферы в точках с координатами (1, 2, −5), (−1, 0, −5), (1, 0, −5) и (−1, 2, −5) соответственно.
7. Создайте четыре новых материала в папке **Materials**. Назовите их **DiffuseBrick**, **SpecularBrick**, **BumpedBrick** и **BumpedSpecularBrick**. Задайте свойства ваших материалов так, как показано на рис. 3.9.

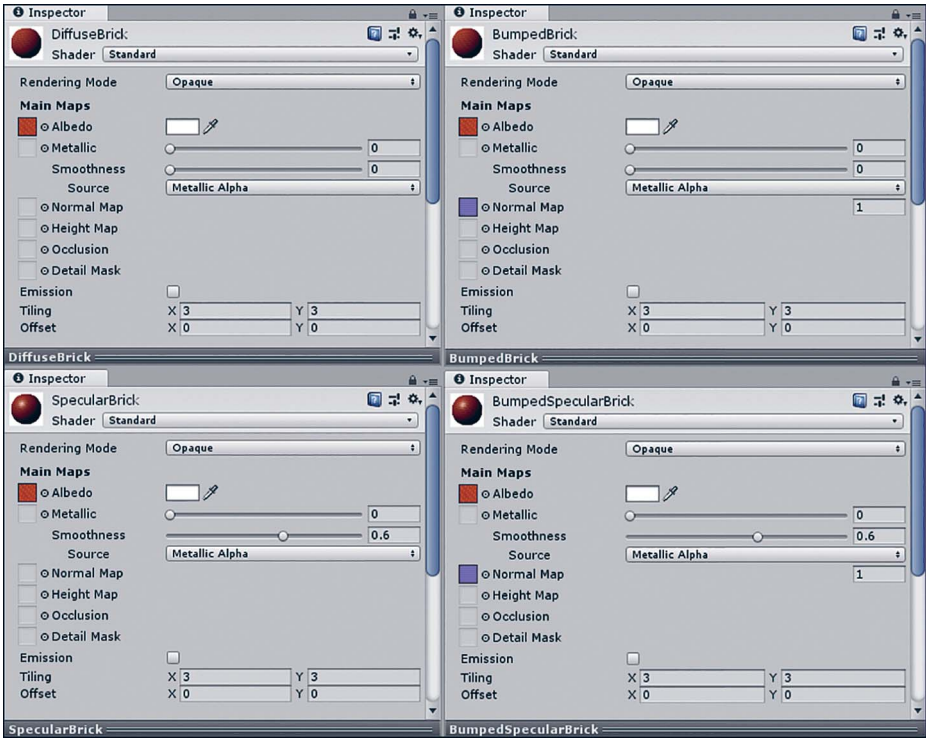


Рис. 3.9. Свойства материалов

8. Перетащите каждый из материалов на одну из четырех сфер. Обратите внимание, как свет и кривизна сфер взаимодействуют с различными шейдерами. Помните, что вы можете перемещаться по сцене и рассмотреть сферы со всех сторон.

4-Й ЧАС

Ландшафт и элементы окружающей среды

Что вы узнаете в этом часе

- Основы ландшафта
- Как создавать ландшафт
- Как украсить ландшафт текстурами
- Как добавить на ландшафт деревья и траву
- Как передвигаться по ландшафту с помощью контроллера персонажа

В этом часе мы изучим, как генерируется ландшафт. Вы узнаете, что такое ландшафт, как создать его и как придать ему нужную форму. Вы также получите практический опыт работы с рисованием текстур и тонкой настройкой. Кроме того, вы научитесь делать обширные, просторные и реалистично выглядящие фрагменты ландшафта для игр и использовать контроллер для перемещения и исследования.

Генерация ландшафта

Все уровни в 3D-играх представляют собой мир в той или иной форме. Эти миры могут быть весьма абстрактными, а могут и реалистичными. Часто в играх с большими уровнями «на открытом воздухе» есть так называемый ландшафт. Этот термин применяется к любой части игры, которая имитирует ландшафт реального мира. Высокие горы, обширные равнины и глубокие болота — вот примеры игровых ландшафтов.

В Unity ландшафт — это плоский меш, который может принимать различные формы. Проще всего представить ландшафт как песок в песочнице. Вы можете выкапывать в нем ямы, а можете насыпать холмы. Единственное, чего ландшафт сделать не позволяет, — «нахлеста» земли. Это означает, что вы не можете создать, например, пещеру или выступающую вперед скалу, такие элементы моделируются отдельно. Кроме того, как любой другой объект в Unity, ландшафт имеет

параметры положения, поворота и масштаба (несмотря на то, что они обычно не изменяются во время игры).

Добавление ландшафта в проект

Создание плоского ландшафта на сцене — довольно простая задача, для решения которой требуется знать некоторые основные параметры. Чтобы добавить ландшафт на сцену, выберите команду **GameObject ⇒ 3D Object ⇒ Terrain**. На панели **Project** появится новый ассет с названием **New Terrain**, а на сцене — объект под названием **Terrain**. Перемещаясь по сцене, вы можете также заметить, что созданный участок ландшафта довольно велик. Более того, он гораздо крупнее, чем требуется в данный момент. Поэтому нам необходимо изменить некоторые свойства ландшафта.

Чтобы наш ландшафт стал более управляемым, нам нужно изменить *разрешение* ландшафта. При этом вы можете изменить длину, ширину и максимальную высоту участка ландшафта.

Почему мы используем именно термин «разрешение», станет более очевидно позже в этом уроке, когда вы узнаете о картах высот. Чтобы изменить разрешение участка ландшафта, выполните следующие действия.

1. Выберите ландшафт на панели **Hierarchy**. Найдите и нажмите кнопку **Terrain Settings** на панели **Inspector** (рис. 4.1).
2. Найдите раздел **Resolution** (Разрешение) и обратите внимание, что в данный момент параметры **Terrain Width** (Ширина участка ландшафта) и **Terrain Length** (Длина участка ландшафта) имеют значение 500. Задайте обоим параметрам значение **50**.

Прочие параметры в разделе **Resolution** определяют отрисовку текстур и производительность ландшафта. Трогать их пока не нужно. После того как вы измените ширину и длину, вы увидите, что участок станет гораздо меньше и более управляемым. Теперь пришло время придать ему нужную форму.

ПРИМЕЧАНИЕ

Размер участка ландшафта

В данный момент вы работаете с фрагментом размером 50 единиц в длину и в ширину. Мы задали такой размер исключительно для удобства, пока вы учитесь использовать различные инструменты. В реальной игре ландшафт, вероятно, будет гораздо больше, в зависимости от ваших потребностей. Стоит также отметить, что, если у вас уже есть карта высот (мы поговорим об этом в следующем разделе), соотношение длины и ширины ландшафта должно соответствовать соотношению у карты высот.

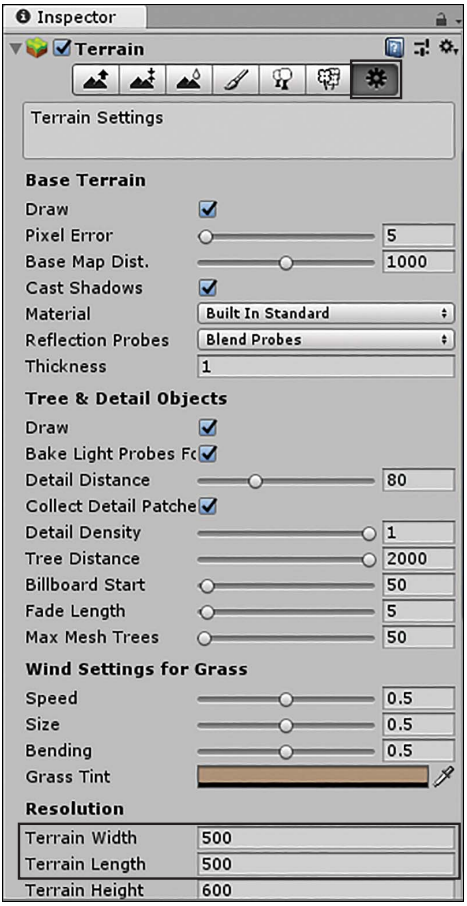


Рис. 4.1. Настройки разрешения

Составление карты высот

Традиционно для 8-битных изображений доступно 256 оттенков серого в диапазоне значений от 0 (черный) до 255 (белый). Эти показатели используются для создания так называемой карты высот. Карта высот — это черно-белое изображение, которое содержит информацию о высотах точек и похоже на топографическую карту. Более темные участки можно рассматривать как низкие точки, а более светлые — как высокие. На рис. 4.2 приведен пример карты высот. Выглядит она, может быть, не впечатляюще, но изображение вроде этого может здорово изменить вашу сцену.

Применить карту высот к вашему пока что плоскому ландшафту легко. Достаточно просто создать плоский ландшафт и импортировать в него карту высот, как описано в практикуме ниже.

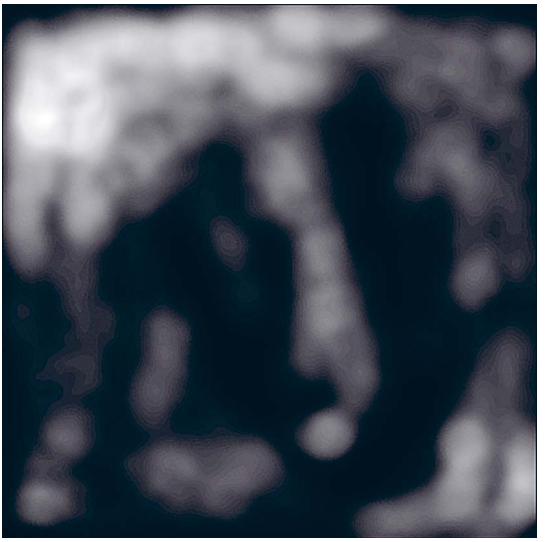


Рис. 4.2. Простая карта высот

ПРАКТИКУМ

Применение карты высот к ландшафту

Выполните предложенные ниже действия, чтобы импортировать и применить карту высот.

1. Создайте новый проект или сцену в редакторе Unity. Найдите файл *terrain.raw* в файлах примеров для часа 4 и поместите в папку, где вы позже сможете легко найти его.
2. Создайте новый ландшафт, следуя руководству, изложенному ранее в этом часе. Ширину и длину фрагмента ландшафта обязательно сделайте равными **50**.
3. Выберите ландшафт на панели **Hierarchy**, нажмите кнопку **Terrain Settings** на панели **Inspector** (см. рис. 4.1). В разделе **Heightmap** (Карта высот) нажмите кнопку **Import Raw.**
4. В диалоговом окне **Import Raw Heightmap** найдите файл *terrain.raw* из шага 1 и нажмите кнопку **Open**.
5. В диалоговом окне **Import Heightmap** установите параметры, как показано на рис. 4.3. (Примечание: свойство **Byte Order** (Порядок байтов) не имеет отношения к операционной системе, которая установлена на вашем компьютере. Это тип ОС, в которой создана карта высот.)
6. Нажмите кнопку **Import**. Ваш ландшафт выглядит странно, не так ли? Проблема заключается в том, что, когда вы уменьшили длину и ширину ландшафта до 50 для большей управляемости, вы оставили высоту, равную 600. Это, очевидно, слишком высокое значение для текущей задачи.

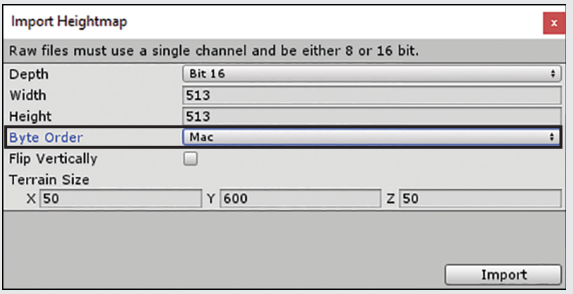


Рис. 4.3. Диалоговое окно **Import Heightmap**

7. Измените разрешение ландшафта, вернувшись к группе элементов управления **Resolution** в разделе **Terrain Settings** панели **Inspector**. Сделайте значение высоты равным **60**. Теперь вы получите гораздо более приятный результат, как показано на рис. 4.4.

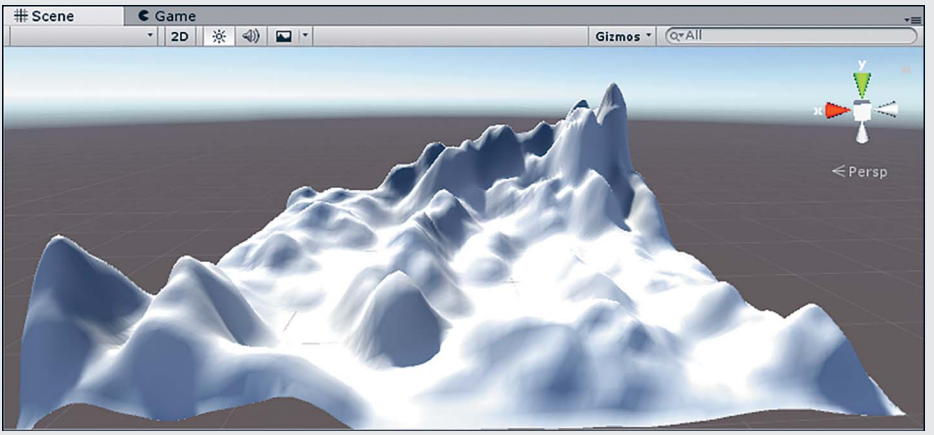


Рис. 4.4. Ландшафт после импорта карты высот

СОВЕТ

Расчет высоты

Карта высот может показаться вам случайным набором цветов, но на самом деле с ней все довольно просто. Высота строится как процент от 255 и максимальной высоты ландшафта. Максимальная высота ландшафта по умолчанию составляет 600, но это значение легко изменить. Если применить формулу *(Оттенок серого) / 255 × (Максимальная высота)*, вы можете легко вычислить высоту любой точки ландшафта.

Например, черный цвет имеет значение 0, значит, любая точка черного цвета будет иметь высоту (*0/255 × 600 = 0*) 0 единиц. Белый цвет имеет значение 255, значит, все белые точки будут иметь высоту 600 единиц (*255/255 × 600 = 600*).

Если у вас есть участок серого цвета со значением 125, то ландшафт на нем будет иметь высоту около 294 единиц (*125/255 × 600 = 294*).

ПРИМЕЧАНИЕ

Форматы карты высот

В редактор Unity импортируются карты высот, представляющие собой изображения в оттенках серого в формате.*raw*. Есть много способов их создать. Вы можете использовать простой редактор изображений или даже саму программу Unity. Если вы создаете карту высот с помощью графического редактора, старайтесь делать карту такой же длины и ширины, как и ландшафт, иначе возникнут некоторые искажения. Если вы работаете с помощью инструментов среды Unity и хотите создать карту высот для полученного ландшафта, вы можете сделать это, перейдя в подраздел **Heightmap** в разделе **Terrain Settings** на панели **Inspector** и нажав кнопку **Export Raw**.

Как правило, для больших ландшафтов и в случаях, где важна производительность, следует экспортировать карту высот и конвертировать ландшафт в меш в другой программе. Имея 3D-меш, вы можете также добавить пещеры, свесы и иные подобные элементы перед импортом сетки для использования в Unity. Однако здесь важно отметить, что, если вы импортируете меш для использования в качестве ландшафта, вы не сможете применить инструменты Unity для нанесения текстуры и рисования на нем. (Тем не менее вы можете найти сторонние ассеты в магазине Asset Store, в которых эта функция сохранена).

Инструменты для создания ландшафта в Unity

В Unity существует множество инструментов для создания ландшафта вручную. Они расположены на панели **Inspector** в компоненте **Terrain** и работают по одному и тому же принципу: вы используете кисть с заданным размером и непрозрачностью для «рисования» ландшафта. В сущности, программа считает, что вы рисуете карту высот, а редактор уже превращает ее в 3D-ландшафт. Эффекты рисования кумулятивны, то есть чем больше вы рисуете в некоторой области, тем сильнее становится эффект в ней. На рис. 4.5 показаны инструменты, которые можно использовать для создания практически любого ландшафта, какой вы можете себе представить.

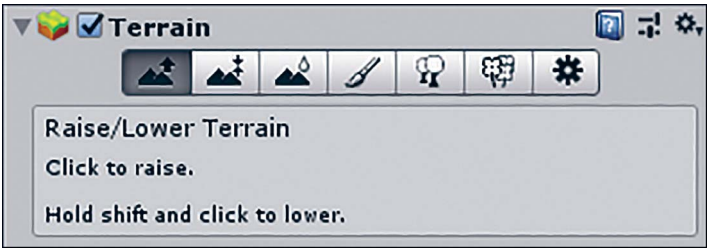


Рис. 4.5. Инструменты рисования ландшафта

Первый инструмент, который вы научитесь использовать, — это **Raise/Lower**. Он позволяет поднимать или опускать ландшафт в том месте, где вы рисуете. Для создания ландшафта этим инструментом выполните следующие действия.

1. Выберите кисть. (Кисть определяет размер и форму эффекта.)
2. Выберите размер и непрозрачность кисти. (Непрозрачность определяет, насколько сильный эффект кисть будет создавать.)
3. Нажав и удерживая левую кнопку мыши на ландшафте на панели **Scene**, перетаскивайте мышь, чтобы поднять рельеф. Выполняя то же самое с удерживаемой клавишей **Shift**, вы можете опустить рельеф.

На рис. 4.6 показаны удачные настройки для создания рельефа для нашего ландшафта размером 50 × 50 с высотой 60.

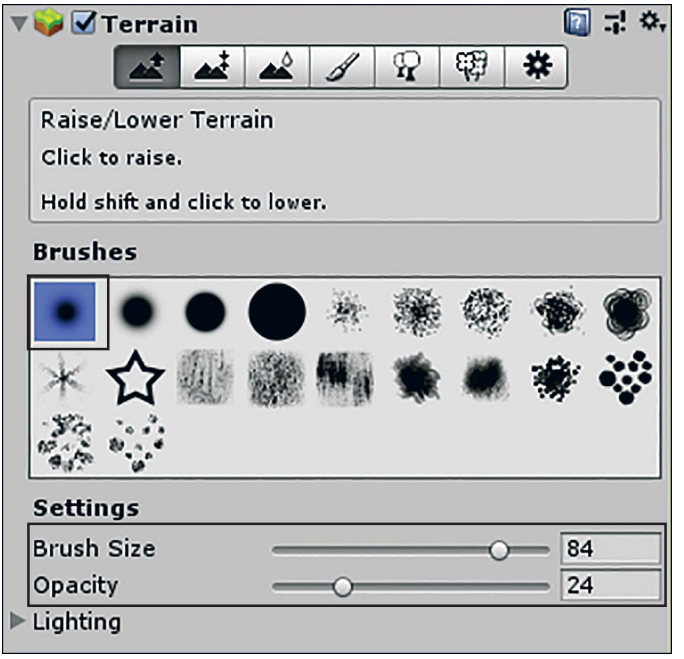


Рис. 4.6. Удачные настройки для создания рельефа

Следующий инструмент называется **Paint Height**. Он работает почти точно так же, как и **Raise/Lower**, и еще устанавливает ландшафт на заданную высоту. Если указанная высота выше текущей, ландшафт будет подниматься, а если ниже — опускаться. Это полезно для создания, например, горного плато и других плоских структур в ландшафте. Самое время опробовать этот инструмент!

СОВЕТ

Выравнивание ландшафта

Если вы в какой-то момент хотите вернуть ландшафт в исходное плоское состояние, вы можете сделать это, выбрав инструмент **Height Paint** и нажав кнопку **Flatten** (Выровнять). Одно из преимуществ этого метода заключается в том, что вы можете выровнять ландшафт до высот, отличных от 0. Если максимальная высота равна 60 единиц и вы выполните выравнивание на высоте 30, у вас будет возможность поднять или опустить ландшафт на 30 единиц. Это позволяет легко создавать долины в плоском ландшафте.

Последний инструмент, который мы испытаем, это **Smooth Height**. Он не вносит заметных изменений в ландшафт, но убирает резкие неровности и зазубрины, которые появляются во время редактирования. Это что-то вроде шлифовальной машины. Вы будете использовать его как последний штрих для устранения дефектов, когда основной ландшафт уже создан.

ПРАКТИКУМ

Создание ландшафта

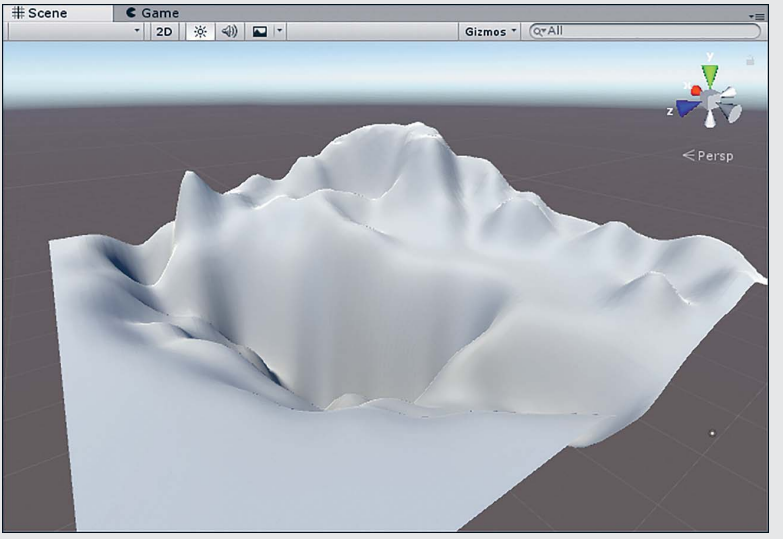


Рис. 4.7. Пример созданного ландшафта

Теперь, когда вы узнали об инструментах для создания ландшафта, пришло время попрактиковаться в их использовании. В этом упражнении мы сделаем заданный участок ландшафта.

1. Создайте новый проект или сцену и добавьте на сцену ландшафт. Установите размер ландшафта 50 × 50 единиц и высоту 60.
2. Выровняйте ландшафт на высоте 20. Для этого выберите инструмент **Paint Height**, задайте высоту **20** единиц и нажмите кнопку **Flatten**. (Примечание: если вам показалось, что ландшафт исчез, — это не так. Он лишь переместился на отметку 20 единиц).
3. Используя различные инструменты, попробуйте создать ландшафт, похожий на тот, что представлен на рис. 4.7. (Примечание: освещение на рисунке было изменено, чтобы вы могли лучше его рассмотреть.)
4. Продолжайте экспериментировать с инструментами.

СОВЕТ

Практика, практика, практика

Разработка крутых и реалистичных уровней — это само по себе искусство. Придется хорошо подумать, чтобы правильно разместить в игре холмы, долины, горы и озера. Элементы должны быть не только визуально реалистичны, но и расположены таким образом, чтобы на полученном уровне было возможно играть.

Навык создания уровней не возникает внезапно. Следует долго практиковать, оттачивать и совершенствовать свои навыки, чтобы создавать интересные и запоминающиеся уровни.

Текстуры ландшафта

Теперь вы знаете, как создать физические измерения в 3D-мире. Несмотря на то что у вашего ландшафта множество особенностей, он по-прежнему гладкий, блестящий (из-за используемого по умолчанию материала) и в нем трудно ориентироваться. Пора добавить вашему уровню изюминку. В этом разделе вы узнаете, как задать ландшафту текстуру, чтобы придать ему привлекательный внешний вид. Как и создание рельефа, текстурирование ландшафта во многом похоже на рисование. Вы выбираете кисть и текстуру, а затем раскрашиваете свой мир.

Импорт ассетов ландшафта

Прежде чем украсить ваш мир, вам понадобятся собственно текстуры, с которыми можно работать. В Unity есть некоторые готовые ассеты, которые вы можете использовать, но для этого вы должны их импортировать. Чтобы их загрузить, выберите команду меню **Assets** ⇒ **Import Package** ⇒ **Environment**. Появится диалоговое

окно **Import Unity Package** (рис. 4.8). Здесь вы можете выбрать, какие именно ассеты хотите импортировать. Иногда бывает полезно снять флажки с ненужных элементов, чтобы сократить размер проекта. Но сейчас давайте оставим все настройки по умолчанию и нажмем кнопку **Import**. На панели **Project** должна появиться новая папка **Standard Assets** ниже каталога **Assets**. В ней расположены все ассеты ландшафта, которые мы будем использовать на протяжении этого часа.

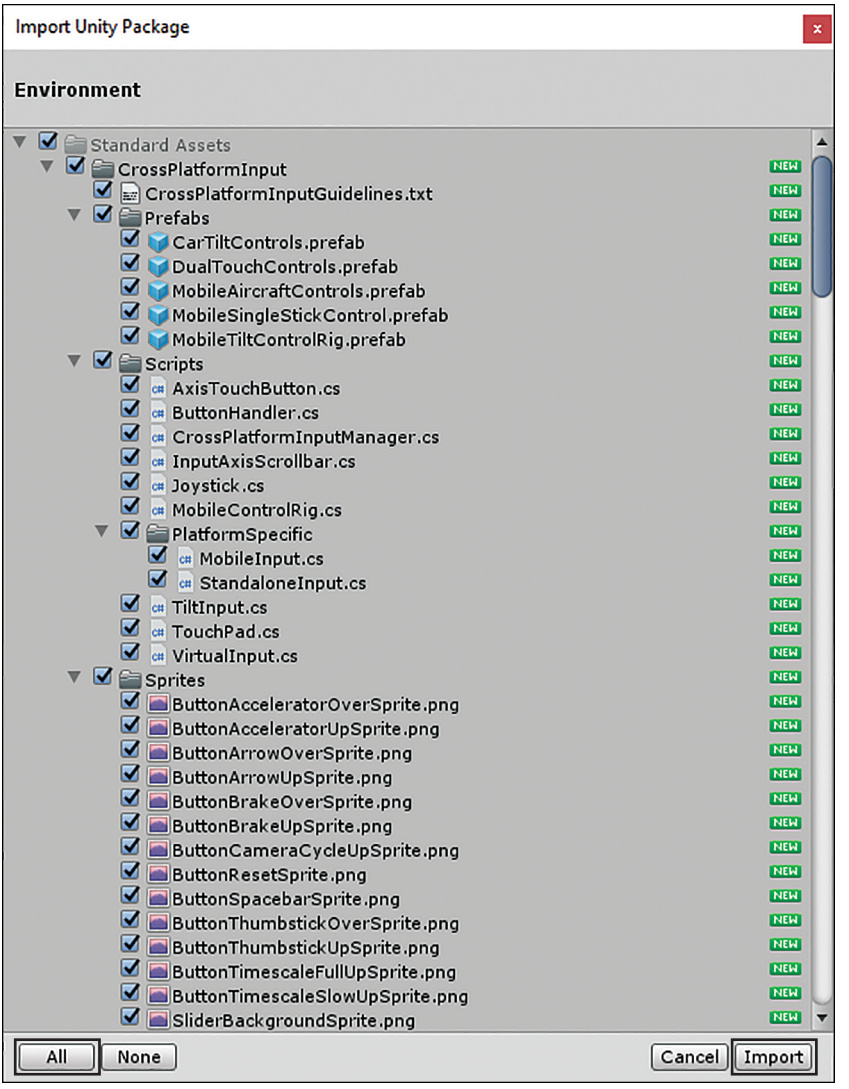


Рис. 4.8. Диалоговое окно **Import Unity Package**

ПРИМЕЧАНИЕ

Потерянные пакеты

Если в меню **Assets** ⇒ **Import Package** нет пакета **Environment**, это означает, что вы не выбрали опцию **Standard Assets** при установке среды Unity. Поскольку вам нужны эти ассеты (и многие другие, которые будут использоваться в этой книге), вы можете запустить программу установки еще раз и отметить установку стандартных ассетов.

Текстурирование ландшафта

Чтобы начать рисовать ландшафт, сначала нужно загрузить текстуру. На рис. 4.9 показан инструмент **Paint Texture** на панели **Inspector**, который появляется после выбора ландшафта на панели **Hierarchy**. Обратите внимание на три числовых свойства: **Brush size**, **Opacity** и **Target strength**. С первыми двумя вы уже знакомы, а **Target strength** — это максимальный уровень непрозрачности, которого можно достигнуть, если рисовать на одной и той же области. Значение этого параметра представляет собой процент, где 1 соответствует 100%. Используйте его в качестве меры контроля, чтобы избежать чрезмерной окраски текстур.

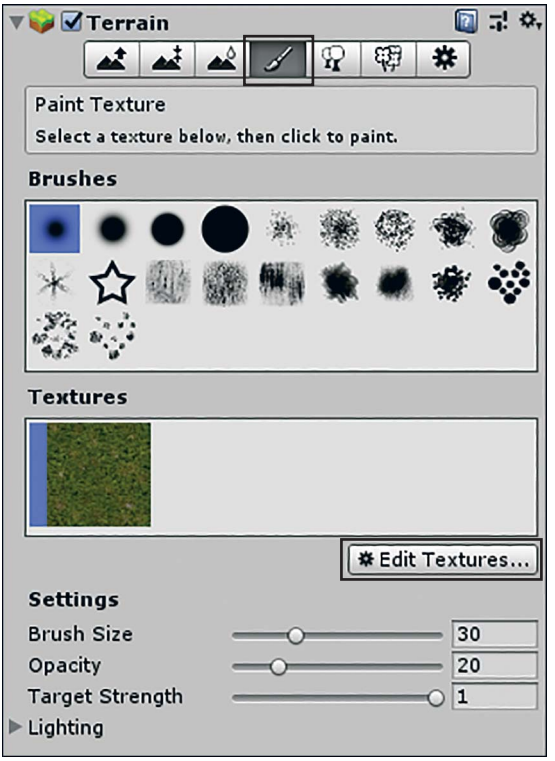


Рис. 4.9. Инструмент **Paint Texture** и его параметры

Чтобы загрузить текстуру, выполните следующие действия.

1. Выберите команду **Edit Textures** ⇒ **Add Texture** на панели **Inspector** (не путайте с основным меню редактора Unity).
2. В диалоговом окне **Edit Terrain Texture** нажмите кнопку **Select** в области текстуры (рис. 4.10) и выберите текстуру **GrassHillAlbedo**.

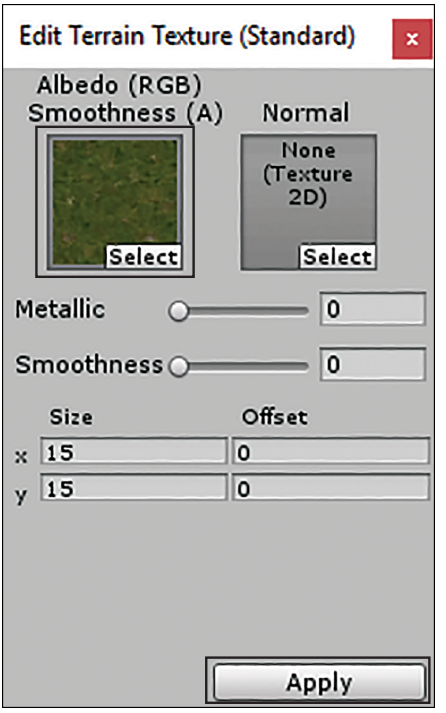


Рис. 4.10. Диалоговое окно **Edit Terrain Texture**

3. Нажмите кнопку **Add**. Нет необходимости добавлять карту нормалей, но вы можете это сделать, если у вас ребристая текстура.

После этого весь ваш ландшафт покроется пятнистой травой. Он уже выглядит лучше, чем белый, который был раньше, но до реалистичного вида все еще далеко. Теперь мы перейдем к настоящему рисованию и улучшим наш ландшафт.

ПРАКТИКУМ

Рисуем текстурами на ландшафте

Выполните предложенные ниже действия, чтобы нанести на ваш ландшафт новую текстуру и тем самым придать ему более реалистичный двухцветный эффект.

1. Следуя алгоритму, приведенному в начале этого часа, добавьте новую текстуру. На этот раз загрузите текстуру **GrassRockyAlbedo**. Загрузив текстуру, не забудьте выбрать ее. (Примечание: под изображением выбранной текстуры появляется синяя полоса.)
2. Установите следующие параметры кисти: параметр **Brush Size** — **30**, **Opacity** — **20**, и **Target Strength** — **0,6**.
3. Легкими движениями (нажав и удерживая кнопку мыши) нанесите текстуру на крутые склоны и трещины вашего ландшафта. Такой прием позволяет создать впечатление, что трава не растет на крутых склонах и между возвышенностями.
4. Поэкспериментируйте с нанесением текстур. Попробуйте использовать текстуру **CliffAlbedoSpecular** на крутых склонах горы, а текстурой **SandAlbedo** можно нарисовать тропу.

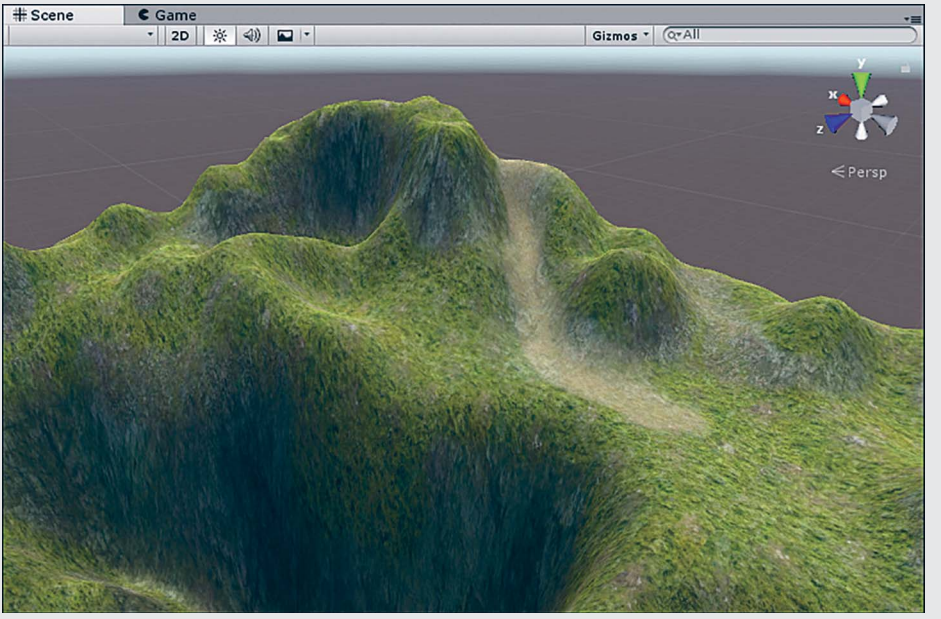


Рис. 4.11. Пример горы с двумя текстурами и песчаной тропой

Вы можете загрузить сколько угодно текстур, на свое усмотрение, чтобы добиться реалистичного внешнего вида. Обязательно попрактикуйтесь в текстурировании, чтобы определить, какие сочетания текстур выглядят лучше всего.

ПРИМЕЧАНИЕ

Создание текстур ландшафта

Игровые миры часто уникальны, и для их создания требуются пользовательские текстуры, которые подходили бы обстановке и атмосфере игры. Вам будет полезно знать некоторые общие принципы создания собственных текстур для ландшафта. Во-первых, всегда старайтесь сделать шаблон повторяемым. Это означает, что тайлы текстур должны получаться «бесшовными». Чем больше текстура, тем меньше бросается в глаза ее цикличность. Второе: старайтесь делать текстуру квадратной. И, наконец, старайтесь выбирать в качестве размера текстуры одну из степеней числа 2 (32, 64, 128, 512 и так далее). Последние две рекомендации влияют на сжатие текстур и их эффективность. Немного попрактиковавшись, вы научитесь в кратчайшие сроки создавать потрясающие текстуры ландшафта.

СОВЕТ

Политика скрытности

Текстурируя ландшафт, не забывайте работать как можно тоньше. В природе один элемент плавно перетекает в другой без резких переходов, и вы должны следовать именно этому принципу. Если, отдалив камеру от участка ландшафта, вы можете точно сказать, где находится граница между текстурами, — ваша работа недостаточно тонкая. Часто лучше наносить текстуры маленькими и тонкими мазками, а не одним широким движением.

ВНИМАНИЕ!

Что за ошибки TerrainData?

В зависимости от параметров проекта и версии редактора Unity, которую вы используете, программа может выдавать вам ошибки при запуске сцены. Например, иногда возникает ошибка, в которой говорится что-то вроде «TerrainData is missing splat texture … make sure it is marked for read/write in the importer» (Отсутствует текстура в данных ландшафта… Убедитесь, что вы поставили отметку чтения/записи в импортере). Это значит, что у вас возникли проблемы с доступом во время задействования текстур. К счастью, исправить ошибку очень просто. Все, что вам нужно сделать, это нажать на вызвавшую ошибку текстуру на панели **Project,** и на панели **Inspector** появятся настройки импорта. На панели **Inspector** откройте список **Advanced** и установите флажок **Read/Write Enabled**. Проблема решена!

Генерация деревьев и травы

Мир, в котором только плоские текстуры, весьма скучен. Почти на каждом природном ландшафте имеется какая-нибудь растительность. В этом разделе вы узнаете, как добавлять и настраивать деревья и траву, чтобы придать вашему ландшафту живой вид и ощущение реалистичности.

Рисуем деревья

Добавление деревьев на ландшафт работает примерно так же, как создание рельефа и текстурирование. Весь процесс очень похож на рисование. От вас требуется загрузить модель дерева, установить свойства для деревьев, а затем закрасить область, где вы хотите их видеть. Основываясь на заданных вами опциях, Unity сгенерирует деревья и изменит их внешний вид так, чтобы все выглядело более естественно и органично.

Для генерации деревьев на ландшафте используется инструмент **Paint Trees**. Выбрав на сцене нужный ландшафт, выберите инструмент **Paint Trees** на панели **Inspector** в области компонента **Terrain (Script)**. На рис. 4.12 показан инструмент **Paint Trees** и его дефолтные настройки.

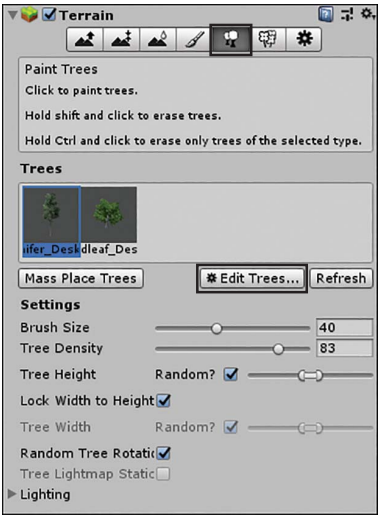


Рис. 4.12. Инструмент **Paint Trees**

В таблице 4.1 приведено описание свойств инструмента **Paint Trees**.

ТАБЛ. 4.1. Описание свойств инструмента Paint Trees

Свойство	Описание
Brush Size	Размер области, в которую добавляются деревья при нанесении
Tree Density	Плотность генерации деревьев
Tree Height/ Width, **Rotation** и т. д.	Определяет, как деревья отличаются друг от друга. Настройка этих свойств позволяет вам создать впечатление множества различных деревьев вместо дублирования одного и того же

ПРАКТИКУМ

Размещение деревьев на ландшафте

Давайте рассмотрим порядок действий при размещении деревьев на ландшафте с помощью инструмента **Paint Trees**. Это упражнение предполагает, что вы уже создали новую сцену и добавили ландшафт. Задайте ландшафту параметры длины и ширины, равные **100**. Результат будет выглядеть лучше, если на ландшафте уже создан рельеф и проведено текстурирование. Выполните следующие действия.

1. Выберите пункт **Edit Trees** ⇒ **Add Tree**, после чего откроется диалоговое окно **Add Tree** (рис. 4.13).
2. Далее нажмите на значок круга справа от текстового поля **Tree Prefab** в диалоговом окне **Add Tree**, после чего откроется окно **Tree Selector**.

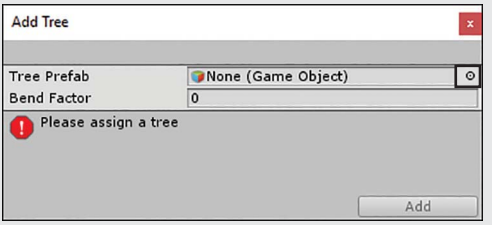


Рис. 4.13. Диалоговое окно **Add Tree**

3. Выберите дерево **Conifer_Desktop** и нажмите кнопку **Add**.
4. Установите размер кисти **2** и плотность размещения **10**. Оставьте параметры **Tree Width** и **Tree Height** (ширина и высота деревьев) заданными как **Random** (Случайно), но уменьшите среднее значение размера с помощью ползунка.
5. Нанесите деревья на ландшафт, зажимая левую кнопку мыши над областями, где вам нужны деревья. Удерживая нажатой клавишу **Shift** и левую кнопку мыши, вы можете удалить ненужные деревья. Если рисовать не получается, выберите пункт **Terrain Settings** ⇒ **Tree & Detail Objects** и убедитесь, что флажок **Draw** установлен.
6. Продолжайте экспериментировать с различными размерами кисти, плотностью нанесения и деревьями.

Рисуем траву

Теперь, когда вы научились рисовать деревья, вы узнаете, как нанести траву и прочую мелкую растительность на ваш ландшафт. Трава и другие мелкие растения в редакторе Unity называются *деталями*. Таким образом, для рисования травы нам понадобится инструмент под названием **Paint Details**. В отличие от деревьев, которые являются 3D-моделями, детали — это *билборды* (см. примечание «Билборды»). Детали наносятся на местность с помощью кисти рисующими движениями, так же, как мы делали ранее в этом часе. На рис. 4.14 показан инструмент **Paint Details** и некоторые из его свойств.

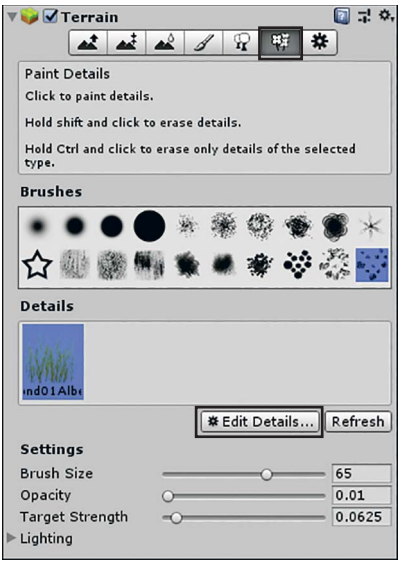


Рис. 4.14. Инструмент **Paint Details**

ПРИМЕЧАНИЕ

Билборды

Билборды — это особый тип визуальных составляющих в трехмерном мире, которые дают эффект 3D-модели, хотя сами таковыми не являются. 3D-модели существуют во всех трех измерениях. Это значит, что, двигаясь вокруг, вы можете увидеть разные стороны модели. Но билборды — это плоские изображения, которые всегда повернуты к камере. При попытке обойти билборд он повернется туда, где вы находитесь. Обычно свойствами билбордов обладает трава, частицы, эффекты экрана.

Выращивание травы на ландшафте — довольно простой процесс. Сначала нужно добавить текстуру травы.

1. Нажмите кнопку **Edit Details** на панели **Inspector** и выберите пункт **Add Grass Texture**.
2. В диалоговом окне **Edit Grass Texture** щелкните мышью по кружку рядом с текстовым полем **Texture** (рис. 4.15). Выберите текстуру **GrassFrond01AlbedoAlpha**. Вы можете воспользоваться поиском по слову *grass*, чтобы найти ее.
3. Установите любые свойства текстуры, какие хотите. Обратите особое внимание на цветовые свойства, так как они задают диапазон естественных цветов для травы.
4. Когда вы закончите изменение настроек, нажмите кнопку **Apply**.

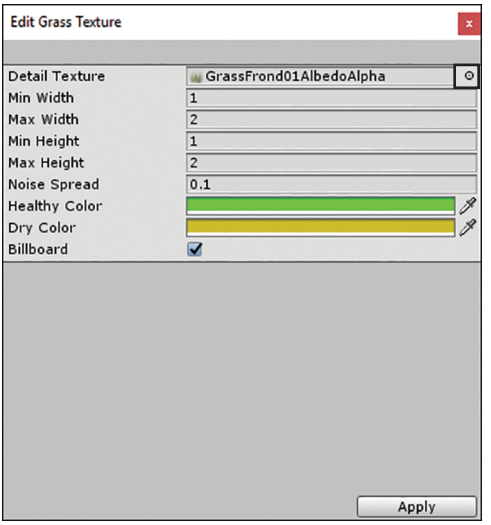


Рис. 4.15. Диалоговое окно **Edit Grass Texture**

Когда трава будет загружена, вам нужно выбрать кисть и задать ее свойства, и вы будете готовы рисовать траву.

СОВЕТ

Реалистичная трава

Вы можете заметить, что, когда начинаете рисовать траву, она выглядит нереалистично. Вам нужно обратить внимание на несколько моментов при добавлении травы на ландшафт. Во-первых, на цвета, которые вы установили для текстуры травы. Выбирайте их потемнее и в тон земле. Следующее, что вам нужно сделать, это выбрать кисть неправильной формы, чтобы устранить резкие края. (На рисунке показана удачная кисть, которую можно использовать.) Наконец, прозрачность и целевая интенсивность должны оставаться низкими. Можно, например, начать с непрозрачности, равной 0,01, и целевой интенсивности, равной 0,0625. Если вам нужно больше травы, вы можете сильнее закрашивать нужный участок. Вы также можете вернуться в диалоговое окно **Edit Details** и изменить свойства текстуры травы.

ВНИМАНИЕ!

Растительность и эффективность

Чем больше деревьев и травы на вашей сцене, тем больше ресурсов компьютера требуется на обработку. Если для вас остро стоит вопрос производительности, сохраняйте количество растительности минимальным. Кое-какие свойства, которые мы рассмотрим позже в этом часе, помогут вам справиться. А пока возьмите за правило следующее: старайтесь добавлять деревья и траву только на те участки, где они действительно необходимы.

СОВЕТ

Исчезновение травы

Как и в случае с деревьями, отображение травы зависит от расстояния до наблюдателя. По мере отдаления от зрителя деревья начинают отображаться с пониженным качеством, а трава вообще перестает отображаться. В результате вокруг зрителя образуется круговая область, за пределами которой трава не видна. Вы можете изменить это расстояние, используя свойства, описанные далее в этом часе.

Настройки ландшафта

Последний из инструментов для ландшафта, представленный на панели **Inspector** — это **Terrain Settings**. С его помощью можно настраивать то, как в целом выглядит и функционирует сам ландшафт, текстуры, деревья и детали. На рис. 4.16 показан инструмент **Terrain Settings**.

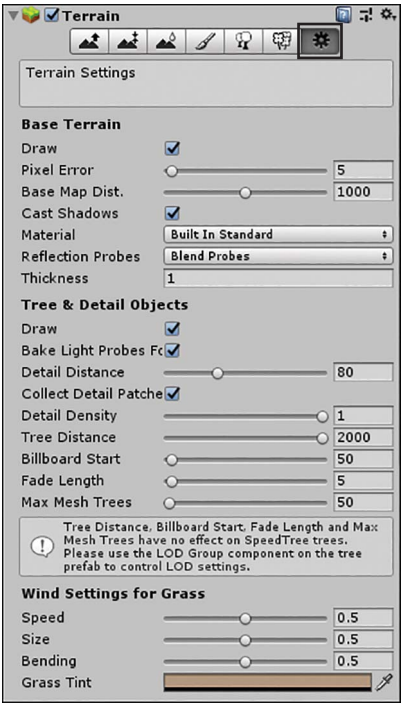


Рис. 4.16. Инструмент **Terrain Settings**

Первая группа параметров касается всего ландшафта в целом. В таблице 4.2 описаны некоторые из них.

ТАБЛ. 4.2. Параметры группы Base Terrain

Параметр	Описание
Draw	Включает или выключает отображение ландшафта
Pixel Error	Определяет допустимое количество ошибок при отображении геометрии ландшафта. Чем выше это значение, тем ниже детальность ландшафта
Base Map Dist	Определяет максимальное расстояние, с которого отображаются текстуры высокого разрешения. Когда зритель находится дальше заданного расстояния, качество текстур снижается
Cast Shadows	Определяет, отбрасывается ли тень от рельефа
Material	Параметр для выбора пользовательского материала для рендеринга местности. Материал должен содержать шейдер, который подходит для отрисовки ландшафта
Reflection Probes	Определяет, как в данном ландшафте функционируют зонды отражения. Опция работает только тогда, когда применяется встроенный стандартный материал или пользовательский материал, который поддерживает рендеринг с отражениями. Это дополнительный параметр, в данной книге он не рассматривается
Thickness	Определяет, насколько зона столкновения ландшафта распространяется в отрицательном направлении по оси *y*. Столкновение объектов с ландшафтом у поверхности рассчитывается на глубину, равную толщине. Это помогает предотвратить попадание быстро движущихся объектов внутрь ландшафта без использования дорогих средств обнаружения непрерывного столкновения

Кроме того, некоторые параметры непосредственно влияют на то, как на местности отображаются деревья и детали (например, трава). В таблице 4.3 описаны эти параметры.

ТАБЛ. 4.3. Параметры группы Tree and Detail Objects

Параметр	Описание
Draw	Определяет отображение деревьев и деталей
Bake Light Probes For	Делает освещение в режиме реального времени более реалистичным и эффективным. Этот параметр требует высокой производительности

Параметр	Описание
Details Distance	Задает расстояние от камеры, на котором детали перестают отображаться
Collect Detail Patches	Предварительно подгружает детали ландшафта, чтобы предотвратить ошибки при перемещении по местности, но затрачивает на это дополнительную память
Tree Distance	Задает расстояние от камеры, на котором деревья перестают отображаться
Billboard Start	Определяет расстояние от камеры, на котором 3D-модели деревьев начинают превращаться в билборды более низкого качества
Fade Length	Определяет расстояние, на котором деревья превращаются из билбордов в 3D-модели более высокого качества. Чем выше это значение, тем плавнее превращение
Max Mesh Trees	Определяет общее число деревьев, которые могут одновременно отображаться как 3D-меши, а не билборды

Следующая группа настроек касается ветра. Поскольку вам еще не довелось побегать внутри вашего мира (хотя позже мы этим займемся), вы можете удивиться, как работает ветер. В целом редактор Unity имитирует легкий ветерок над ландшафтом, он сгибает и раскачивает траву, оживляя сцену. В таблице 4.4 описаны параметры ветра. (Настройки группы **Resolution** были рассмотрены ранее в этом часе в разделе «Добавление ландшафта в проект».)

ТАБЛ. 4.4. Параметры ветра

Параметр	Описание
Speed	Определяет скорость и, следовательно, силу воздействия ветра
Size	Определяет размер участка травы, единовременно подверженного влиянию ветра
Bending	Определяет интенсивность раскачивания травы из-за ветра
Grass Tint	Определяет общую окраску всей травы на уровне. (Это на самом деле настройка не для ветра, но связана с ним.)

Контроллеры персонажа

Итак, ландшафт готов. Мы слепили его, раскрасили текстурами и нанесли деревья и траву. Сейчас настало время погулять по нашему уровню и посмотреть, как на нем играется. В редакторе Unity предусмотрено два основных типа контроллеров для персонажа, которые позволяют оказаться внутри сцены без особых усилий с вашей стороны. Идея такая: вы добавляете контроллер на сцену, а затем перемещаетесь по ней, используя схему управления, общую для большинства игр от первого лица.

Добавление контроллера персонажа

Чтобы добавить контроллер персонажа на сцену, вам в первую очередь необходимо импортировать ассет, выбрав пункт меню **Assets** ⇒ **Import Package** ⇒ **Character**. В диалоговом окне **Import Package** оставьте все флажки установленными и нажмите кнопку **Import**. На панели **Project** появится новая папка под названием **Characters** под папкой **Standard Assets**. Поскольку у вас нет 3D-модели, которую вы могли бы взять в качестве игрока, вам придется использовать контроллер от первого лица. Найдите ассет **FPSController** в папке **Character Controllers** (рис. 4.17) и перетащите его на ландшафт на панели **Scene**.

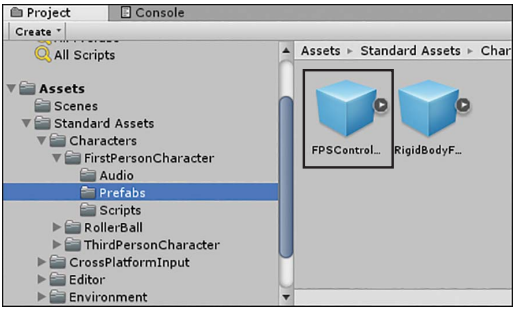


Рис. 4.17. Контроллер персонажа **FPSController**

Сейчас, когда вы добавили на вашу сцену контроллер персонажа, вы можете передвигаться по созданному уровню. Обратите внимание, что теперь вы смотрите из той точки, куда был помещен контроллер. Вы можете использовать клавиши **W**, **A**, **S** и **D** для перемещения, мышь, чтобы оглядеться, и клавишу **Пробел**, чтобы прыгать. Поэкспериментируйте с элементами управления, если вам нужно привыкнуть к ним, и наслаждайтесь игрой в вашем собственном мире!

СОВЕТ

Сообщение «2 Audio Listeners»

Когда вы добавили контроллер персонажа на сцену, вы, возможно, заметили сообщение в нижней части редактора: «There are 2 audio listeners in the scene (На сцене присутствует 2 компонента Audio Listener)». Это связано с тем, что у объекта **Main Camera** (камера, которая находится на сцене по умолчанию) есть компонент **Audio listener**, и точно такой же есть у контроллера персонажа, который вы добавили. Поскольку камеры — это точка обзора игрока, слушать аудио может только одна из них. Вы можете решить проблему, удалив компонент **Audio listener** из объекта **Main Camera**. Более того, вы можете вообще удалить объект **Main Camera**, если хотите, так как у объекта **FPSController** есть своя камера.

СОВЕТ

Падение в бездну

Если вы обнаружили, что камера падает в мир всякий раз, когда вы запускаете сцену, то, возможно, ваш контроллер персонажа частично застрял в земле. Попробуйте переместить его немного выше. После запуска сцены камера должна слегка упасть, пока не достигнет земли и не остановится.

Резюме

В этом часе вы узнали все о ландшафтах в редакторе Unity. Сначала вы узнали, что такое ландшафты и как добавлять их на сцену. Затем вы научились создавать рельеф с помощью карты высот и встроенных инструментов редактора Unity. Вы узнали, как сделать ваши ландшафты более привлекательными, нанося реалистичные текстуры. И, наконец, вы научились добавлять на ландшафт деревья и траву, а также прогулялись по уровню с помощью контроллера персонажа.

Вопросы и ответы

Вопрос: **Обязательно ли в моей игре должен быть ландшафт?**

Ответ: Вовсе нет. Многие игры строятся внутри моделируемых комнат, в абстрактных пространствах или с мешами для внешних областей.

Вопрос: **Мой ландшафт выглядит так себе. Это нормально?**

Ответ: Потребуется некоторое время, чтобы научиться хорошо работать инструментами создания рельефа. Когда вы попрактикуетесь, ваш уровень будет выглядеть гораздо лучше. Истинное качество рождается лишь путем кропотливой работы над уровнем.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Верно или нет: можно ли сделать пещеры с помощью ландшафта в Unity?
2. Как называется черно-белое изображение, содержащее информацию о высотах ландшафта?
3. Верно или нет: создание рельефа в редакторе Unity чем-то похоже на рисование.
4. Как я могу получить доступные текстуры ландшафта в редакторе Unity?

Ответы

1. Нет. Ландшафт в редакторе Unity не может перекрываться.
2. Карта высот.
3. Верно.
4. Импорт ассетов ландшафта осуществляется с помощью команды **Assets** ⇒ **Import Package** ⇒ **Environment**.

Упражнение

Попрактикуйтесь в создании и текстурировании ландшафта. Создайте ландшафт, который содержит следующие элементы:

- пляж;
- горный хребет;
- плоскую равнину.

Чтобы уместить все это, вам может потребоваться пляж побольше. После создания рельефа нанесите на ландшафт текстуры так, как описано ниже. Вы найдете все текстуры, перечисленные здесь, в пакете **Terrain Assets**.

- Для пляжа можно использовать текстуру SandAlbedo, плавно переходящую в GrassRockyAlbedo.
- Равнины и все плоские участки должны быть текстурированы GrassHillAlbedo.

- Текстура GrassHillAlbedo должна плавно переходить в GlassRockyAlbedo по мере того как рельеф становится круче.
- Текстура GlassRockyAlbedo должна переходить в Cliff в самых высоких точках.

Закончите ваш ландшафт, добавив деревья и траву. Подойдите к упражнению творчески и сделайте все так, как хотите. Создайте мир, которым вы будете гордиться.

5-Й ЧАС
Источники света и камеры

Что вы узнаете в этом часе

- Как работать с освещением в Unity
- Ключевые компоненты камер
- Как работать с несколькими камерами на сцене
- Как работать со слоями

В этом часе вы научитесь пользоваться освещением и камерами в редакторе Unity. Мы начнем с изучения основных особенностей освещения. Затем мы исследуем различные типы источников света и их применение в различных задачах. Разобравшись с освещением, мы начнем работать с камерами. Вы узнаете, как добавлять новые камеры, размещать их, а также создавать интересные эффекты с их помощью. В конце этого часа мы разберемся, как работать со слоями в редакторе Unity.

Источники света

В любых видах визуальных средств информации именно освещение во многом определяет, как воспринимается сцена. Яркий, слегка желтоватый свет придает сцене теплый и солнечный вид. Возьмите ту же сцену и подсветите ее слабым голубым светом, и она станет выглядеть жутковато и тревожно. Цвет освещения также будет сочетаться с цветом скайбокса, что придаст ему более реалистичный вид.

В большинстве сцен, в которых требуется реализм или драматический эффект, используется, по крайней мере, один источник света (а часто и больше). В предыдущих часах мы уже немного поработали с освещением, чтобы подчеркнуть нужные нам элементы на сцене. В этом разделе мы познакомимся с освещением поближе.

ПРИМЕЧАНИЕ

Схожие свойства

У разных источников света есть много одинаковых свойств. Если у источника есть свойство, которое уже определено в другом источнике света, то повторно задаваться оно не будет. Помните, что, если два источника света разных типов имеют свойства с одинаковыми названиями, эти свойства будут делать одно и то же.

ПРИМЕЧАНИЕ

Что такое «источник света»?

В движке Unity источник света сам по себе не является объектом — это компонент. Это означает, что при добавлении источника света на сцену вы на самом деле лишь добавляете игровой объект с компонентом **Light**, который может стать источником света любого типа.

Запекание и отображение в реальном времени

Перед тем как приступить непосредственно к работе с источниками света, вам нужно понять два основных способа использования света: в режиме baking (запекание) и в режиме реального времени. Здесь важно не забывать, что весь свет в играх — это, в той или иной мере, набор вычислений. Свет обсчитывается компьютером в три этапа.

1. Рассчитываются цвет, направление и диапазон имитируемых световых лучей от источника света.
2. Когда лучи падают на поверхность, они освещают ее, тем самым изменяя цвет поверхности.
3. Вычисляется угол падения луча на поверхность, и свет отражается. Шаги 1 и 2 повторяются снова и снова (в зависимости от настройки света). С каждым отражением свойства световых лучей изменяются поверхностями, от которых они отражаются (как и в реальной жизни).

Этот процесс повторяется для каждого источника света в каждом кадре и создает глобальное освещение (в котором освещенность и цвет объектов зависит от других объектов вокруг них). Процесс немного упрощается благодаря функции Precomputed Realtime GI (Предварительный расчет глобального освещения в реальном времени), которая включена по умолчанию и не требует от вас дополнительных действий. При этом расчет освещения, описанный выше, выполняется до запуска сцены, и в результате только часть расчетов происходит во время игры. Вы, возможно, уже видели, как это работает. Если вы когда-либо открывали новую сцену в Unity и замечали, что элементы сцены в первые мгновения отображаются темными, значит, вы видели процесс предварительного вычисления.

Baking (*Запекание*), с другой стороны, это полный предварительный расчет света и тени для текстур и объектов во время их создания. Вы можете сделать это с помощью Unity или с помощью графического редактора. Например, если вы создадите текстуру стены с темным пятном, напоминающим человеческую тень, а затем поставите модель человека рядом со стеной, создастся впечатление, что именно эта модель отбрасывает тень на стену. Но правда в том, что тень «запечена» в текстуре. Запекание может повысить быстродействие игры, так как движку не придется рассчитывать свет и тень для каждого кадра. Для ваших текущих потребностей запекание не требуется, так как игры, описанные в этой книге, не настолько сложные, чтобы требовалось ускорение быстродействия.

Точечные источники освещения

Первым мы рассмотрим точечный источник света. Расценивайте его как лампу накаливания. Весь свет излучается из одной центральной точки во всех направлениях. Точечный источник — наиболее распространенный тип света для освещения внутренних областей. Чтобы добавить его на сцену, выберите команду **GameObject** ⇒ **Light** ⇒ **Point Light**. Размещенным на сцене игровым объектом **Point Light** можно манипулировать так же, как и любым другим. В таблице 5.1 описаны его свойства.

ТАБЛ. 5.1. Свойства точечного источника света

Свойство	Описание
Type	Определяет тип света, который испускает компонент. Поскольку это точечный источник света, его тип — **Point**. Изменение этого свойства меняет тип света
Range	Определяет, насколько далеко распространяется свет. Степень освещения снижается равномерно от источника света на заданное расстояние
Color	Определяет цвет света. Это свойство аддитивно, то есть, если вы освещаете синий объект красным светом, он в итоге приобретет фиолетовый цвет
Mode	Определяет режим света: в режиме реального времени, запекание или комбинация этих режимов
Intensity	Определяет, насколько ярким будет свет. Обратите внимание, что дальность при этом сохраняется такой, как указано в свойстве **Range**
Indirect Multiplier	Определяет, насколько ярким остается свет после того, как он отражается от объектов (движок Unity поддерживает свойство **Global Illumination**, то есть выполняет расчет отраженного света)

Свойство	Описание
Shadow Type	Определяет, как рассчитывается тень на сцене для этого источника. Мягкая тень более реалистичная, но требует больших вычислительных ресурсов
Cookie	Позволяет задать кубическую карту (например, скайбокс), которая определяет шаблон падения света. Cookie более подробно описаны позже в этом часе
Draw Halo	Определяет отображение светового гало (ореола) вокруг источника. Более подробно о гало мы поговорим позже в этом часе
Flare	Позволяет задать ассет **Flare** и имитировать эффект бликов на линзе камеры от яркого света
Render Mode	Определяет важность источника света. Возможные значения: **Auto**, **Important** и **Not Important**. Важный источник рендерится в более высоком качестве, а менее важный — быстрее. На данный момент нам подойдет опция **Auto**
Culling Mask ()	Определяет, какие слои освещаются данным источником. По умолчанию источник освещает все. Более подробно о слоях мы поговорим позже в этом часе

ПРАКТИКУМ

Добавление точечного освещения на сцену

Чтобы создать сцену с динамическим точечным освещением, выполните следующие действия.

1. Создайте новый проект или сцену и отключите направленное освещение, установленное по умолчанию.
2. Добавьте на сцену плоскость (выбрав команду **GameObject** ⇒ **3D Object** ⇒ **Plane**). Задайте положение плоскости (0, 0,5, 0) и поворот (270, 0, 0). Плоскость должна находиться в поле зрения камеры, но видна только с одной стороны при просмотре в режиме **Scene**.
3. Добавьте два куба на сцену, расположив их в позициях с координатами (–1,5, 1, –5) и (1,5, 1, –5).
4. Добавьте на сцену источник точечного освещения (для этого выберите команду **GameObject** ⇒ **Light** ⇒ **Point Light**). Расположите его в позиции с координатами (0, 1, –7). Обратите внимание, как источник освещает внутренние стороны кубов и задний план (рис. 5.1).
5. Установите тип **Hard Shadows** (Жесткие тени) для тени от точечного источника, а затем попробуйте подвигать его. Изучите характеристики источника света. Поэкспериментируйте с цветом, диапазоном и интенсивностью.

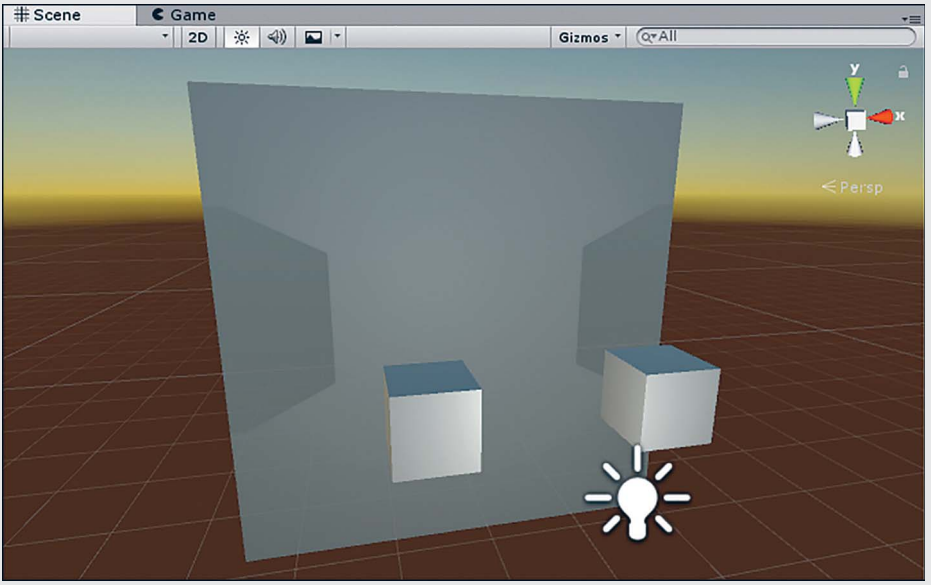


Рис. 5.1. Результат выполнения упражнения из практикума

Прожекторы

Прожекторы работают примерно так же, как фары автомобиля или фонарики. Свет прожектора начинается в некоторой центральной точке и распространяется конусом. Иными словами, прожектор освещает то, что находится перед ним, оставляя все остальное темным. Главная разница заключается в том, что точечный источник светит во всех направлениях, а прожектор можно направить.

Чтобы добавить прожектор на сцену, выберите команду **GameObject** ⇒ **Create Other**⇒ **Spotlight**. А если у вас на сцене уже есть источник, вы можете изменить его тип на **Spot**, и он превратится в прожектор.

Прожекторы имеют только одно свойство, которое мы не рассматривали: **Spot Angle** (Угол освещения). Свойство **Spot Angle** определяет радиус конуса света, испускаемого прожектором.

ПРАКТИКУМ

Добавление прожектора на сцену

Теперь у вас есть возможность поработать с прожекторами в редакторе Unity. Для краткости мы используем проект с точечным источником, созданный в предыдущем практикуме. Если вы пропустили предыдущее упражнение, то выполните сначала его, а затем следующие действия.

1. Скопируйте сцену с точечным источником света из предыдущего проекта (выбрав пункт меню **Edit** ⇒ **Duplicate**) и назовите новую сцену **Spotlight**.
2. Щелкните правой кнопкой мыши по объекту **Point Light** на панели **Hierarchy** и выберите команду меню **Rename**. Задайте объекту имя **Spotlight**. На панели **Inspector** задайте свойству **Type** значение **Spot**. Поместите световой объект в позицию с координатами (0, 1, –13).
3. Поэкспериментируйте со свойствами прожектора. Обратите внимание, как изменение дистанции, интенсивности и угла освещения влияет на результат.

Направленный свет

Последний тип источника света, который мы рассмотрим в этом часе, — направленный свет. Он похож на прожектор в том смысле, что его можно направить. Но, в отличие от прожектора, он освещает всю сцену. Проще всего считать направленный свет чем-то вроде Солнца. Мы уже использовали направленный свет, такой, как Солнце, в часе 4. Освещение от источника направленного света распространяется равномерно по параллельным линиям по всей сцене.

На вновь созданной сцене направленный свет присутствует по умолчанию. Чтобы добавить новый направленный свет на сцену, выберите команду **GameObject** ⇒ **Light** ⇒ **Diretional Light**. Кроме того, если у вас на сцене уже есть источник света, вы можете изменить его тип на **Directional**, и он превратится в источник направленного света.

Такие источники имеют дополнительное свойство, которое мы не описывали ранее: **Cookie Size** (Размер затенения). Мы поговорим о нем позже в этом часе, а сейчас лишь отметим, что это свойство определяет размер шаблона затенения и то, сколько раз он повторяется на сцене.

ПРАКТИКУМ

Добавление источника направленного света на сцену

Теперь добавим источник направленного света на сцену в редакторе Unity. Это упражнение тоже будет основано на проекте, созданном в предыдущем практикуме. Если вы не выполнили это упражнение, сделайте это, а затем выполните следующие действия.

1. Скопируйте сцену **Spotlight** из предыдущего проекта (выбрав команду меню **Edit** ⇒ **Duplicate**) и назовите новую сцену **Directional Light**.
2. Щелкните правой кнопкой мыши по объекту **Spotlight** на панели **Hierarchy** и выберите команду **Rename**. Переименуйте объект, назвав его **Directional Light**. На панели **Inspector** присвойте свойству **Type** значение **Directional**.
3. Измените угол поворота источника света на (75, 0, 0). Обратите внимание, как меняется небо, когда вы поворачиваете источник. Это связано с тем, что на сцене используется процедурный скайбокс. Более подробно о скайбоксах мы поговорим в часе 6.
4. Обратите внимание, как свет падает на объекты на сцене. Теперь измените положение источника света на (50, 50, 50). Освещение объектов не меняется. Поскольку направленный свет — это параллельные линии по всей сцене, положение источника не имеет значения. Освещение может изменяться только за счет вращения.
5. Поэкспериментируйте со свойствами направленного света. У него нет свойства **Range** (так как для него дистанция бесконечна), но обратите внимание на то, как цвет и интенсивность влияют на внешний вид сцены.

ПРИМЕЧАНИЕ

Световые области и излучающие материалы

Есть еще два типа источников света, которые в данной книге не рассматриваются: световые области и излучающие материалы.

Световая область — это функция, которая используется для процедуры *Lightmap Baking*. Это более сложные темы по сравнению с тем, что вам нужно для простых игровых проектов, поэтому мы не рассматриваем их. Если вы хотите узнать больше, вы можете найти необходимую информацию в документации о Unity в Интернете.

Излучающий материал — это нанесенный на объект материал, который фактически излучает свет. Такой тип источника может быть очень полезным для моделирования экрана телевизора, индикаторов и т. п.

Создание источников света из объектов

Поскольку источники света в редакторе Unity являются компонентами, любой объект на сцене может стать источником света. Чтобы добавить в объект источник света, сначала выделите объект. Затем на панели **Inspector** нажмите кнопку **Add Component**. Появится новый список. Выберите подпункт **Rendering**, а затем пункт **Light**. Теперь ваш объект имеет компонент «источник света». Можно добавить источник света на объект, выделив объект и выбрав команду **Component** ⇒ **Rendering** ⇒ **Light**.

Отметим пару нюансов о добавлении света на объекты. Во-первых, объект не будет блокировать свет. Это означает, что источник внутри куба станет излучать свет,

несмотря на твердый материал куба вокруг. Во-вторых, добавление света на объект не заставляет его светиться. Сам объект не будет выглядеть так, будто он испускает свет, но свет исходить, тем не менее, станет.

Гало

Гало — это светящиеся круги, которые появляются вокруг источника света в условиях тумана или облачности (рис. 5.2). Они возникают потому, что свет отражается от мелких частиц, рассеянных вокруг источника. В редакторе Unity вы можете легко добавить источнику света гало. Каждый источник имеет флажок **Draw Halo**. Если флажок установлен, вокруг источника отображается гало. Если вы не видите гало, вы, возможно, находитесь слишком близко к источнику, поэтому попробуйте немного отодвинуть камеру.

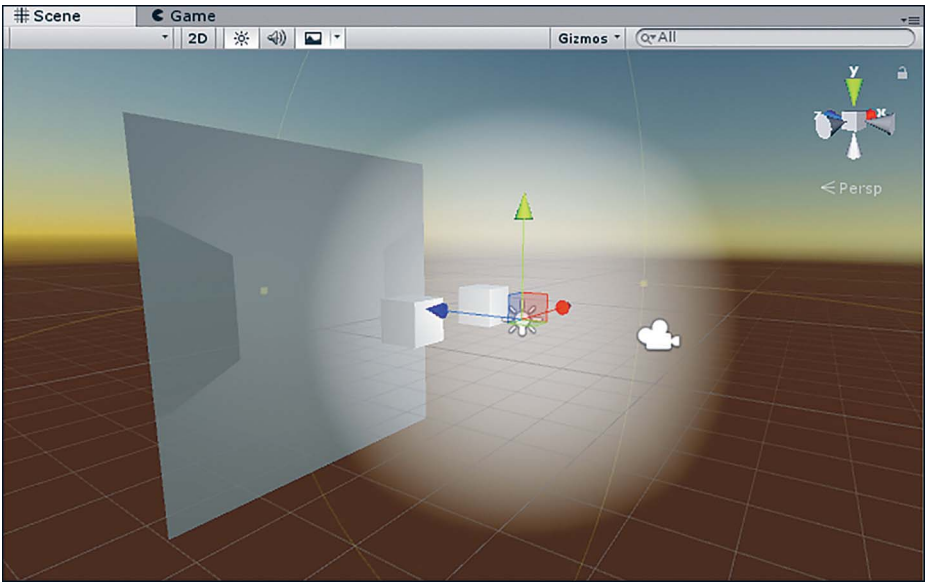


Рис. 5.2. Гало вокруг источника света

Размер гало определяется свойством **Range** источника света. Чем больше этот параметр, тем больше гало. В редакторе Unity также есть несколько свойств, которые применяются ко всем гало на сцене. Вы можете изучить эти свойства, выбрав команду меню **Window** ⇒ **Rendering** ⇒ **Lightning Settings**. Раскройте группу элементов управления **Other Settings**, и настройки появятся на панели **Inspector** (рис. 5.3).

Свойство **Halo Strength** определяет, насколько большим будет гало по отношению к дистанции освещения. Например, если источник света имеет параметр **Range**,

равный 10, и **Halo Strength**, равный 1, то гало будет простираться на все 10 единиц. Если параметру **Halo Strength** присвоено значение 0,5, то гало распространяется только на 5 единиц (10 × 0,5 = 5). Свойство **Halo Texture** позволяет указать другую форму для гало с помощью новой текстуры. Если вы не хотите создавать форму гало, вы можете оставить это поле пустым, и будет использовано круговое гало по умолчанию.

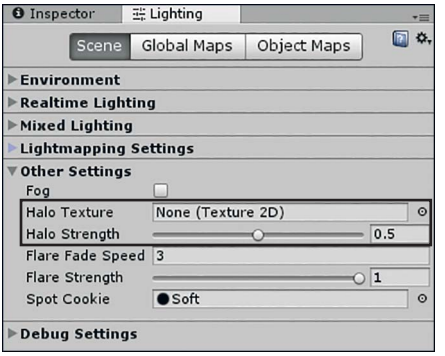


Рис. 5.3. Параметры освещения сцены

Cookie

Если вы когда-нибудь пробовали светить фонариком на стену, поместив руку между лучом и стеной, вы, вероятно, замечали, что ваша рука блокирует часть света, оставляя на стене тень в форме руки. Вы можете имитировать этот эффект в Unity с помощью затенителей. Cookie — это специальные текстуры, которые вы можете добавить к источнику света, чтобы определить форму его излучения. Для точечного источника, прожекторов и направленного света используются разные затенители. В прожекторах и направленных источниках применяются черно-белые плоские текстуры затенителей. Прожекторы не повторяют затенители, а направленные источники — повторяют. В точечных источниках света также используются черно-белые текстуры, но источник помещается в кубмап. Кубмап — это шесть текстур, из которых формируется куб (наподобие скайбокса).

Добавить к источнику света cookie довольно легко. Вам нужно выбрать текстуру для свойства **Cookie** источника света. Чтобы cookie работал, нужно правильно настроить текстуру заранее: выберите ее в редакторе Unity, а затем измените ее свойство на панели **Inspector**. На рис. 5.4 показаны настройки для превращения текстуры в cookie.

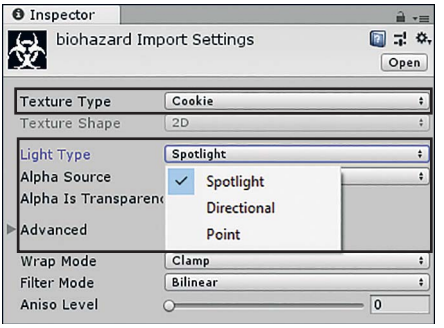


Рис. 5.4. Свойства текстуры-затенителя для точечного источника, прожектора и направленного света

ПРАКТИКУМ

Добавление cookie в прожектор

Для этого упражнения нам понадобится изображение *biohazard.png*, которое можно найти в файлах примеров для часа 5. Выполните следующие шаги, чтобы поместить cookie в центр внимания и увидеть весь процесс от начала до конца.

1. Создайте новый проект или сцену. Удалите из сцены направленный свет.
2. Добавьте на сцену плоскость, расположите ее в позиции с координатами (0, 1, 0) и задайте вращение (270, 0, 0).
3. Добавьте к объекту **Main Camera** прожектор, выделив объект **Main Camera**, а затем выбрав команду **Component** ⇒ **Rendering** ⇒ **Light** и изменив тип источника на **Spot**. Установите параметр дистанции равным **18**, угол освещения **40**, а интенсивность **3**.
4. Перетащите текстуру *biohazard.png* из файлов примеров к книге на панель **Project**. Выберите текстуру, а затем на панели **Inspector** измените ее тип на **Cookie**, установите тип источника света **Spotlight**, а параметру **Alpha Source** присвойте значение **From Grayscale**. В результате ваш cookie будет блокировать свет там, где текстура черная.
5. Выделите объект **Main Camera**, а затем перетащите текстуру **biohazard** в поле свойства **Cookie** компонента **Light**. Вы увидите символ биологической опасности, который спроецируется на плоскость (рис. 5.5).
6. Поэкспериментируйте с различными значениями дистанции и интенсивности. Попробуйте повернуть плоскость и посмотрите, как деформируется и искажается символ.

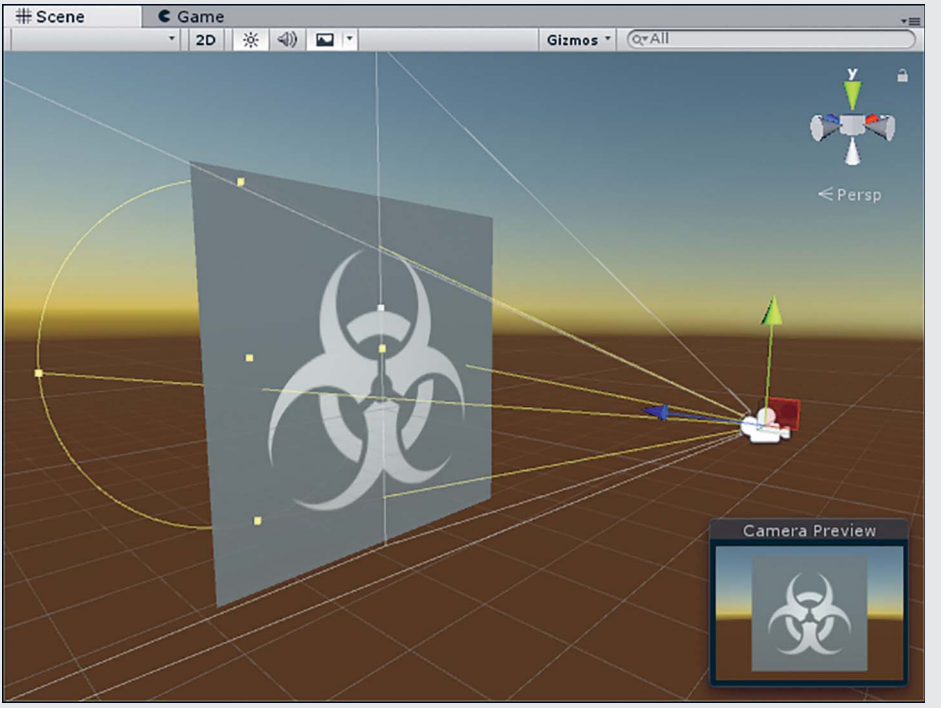


Рис. 5.5. Прожектор с затенителем

Камеры

Камера — это точка обзора (или угол обзора) мира. Она показывает игроку перспективу и контролирует, как для него отображаются объекты. В каждой игре в Unity есть, по крайней мере, одна камера. Камера генерируется всякий раз, когда вы создаете новую сцену. Она всегда указана на панели **Hierarchy** под названием **Main Camera**. В этом разделе вы узнаете все о камерах и о том, как использовать их для достижения интересных эффектов.

Как устроена камера

У всех камер имеется одинаковый набор свойств, которые определяют их поведение. Они описаны в таблице 5.2.

ТАБЛ. 5.2. Свойства камеры

Свойство	Описание
Clear Flags	Определяет, что показывает камера в тех областях, где нет ни одного игрового объекта. По умолчанию выбрана опция **Skybox**. Если скайбокса нет, камера показывает заливку сплошным цветом. Опция **Depth Only** применяется только тогда, когда работают несколько камер. Опция **Don't Clear** создает полосы обесцвечивания и должна использоваться только тогда, когда вы пишете пользовательский шейдер
Background	Задает цвет фона, если нет скайбокса
Culling Mask	Определяет, какие слои видимы для камеры. По умолчанию камера видит все. Вы можете убрать определенные слои (мы рассмотрим работу со слоями позже в этом часе), и они не будут видны через эту камеру
Projection	Определяет, как камера видит мир. Доступно две опции: **Perspective** и **Orthographic**. Перспективные камеры отображают мир в 3D, при этом более близкие объекты видятся крупнее, а удаленные — мельче. Этот параметр используется, если вашей игре нужна глубина. Опция **Orthographic** позволяет игнорировать глубину и отображает все плоским
Field of View	Определяет ширину области, которую видит камера
Clipping Planes	Определяет дистанцию, на которой объекты видимы для камеры. Объекты, которые находятся ближе ближайшей плоскости или дальше дальней плоскости, не будут видны
View Port Rect	Определяет, какая часть экрана проецируется в камеру. (Название свойства — это сокращение от View Port Rectangle — Прямоугольник обзора.) По умолчанию параметры *x* и *y* равны 0, и это значит, что камера в начале расположена в нижней левой части экрана. Если ширина и высота равны 1, то камера охватывает 100% экрана по вертикали и по горизонтали. Позже мы поговорим об этом более подробно
Depth	Задает приоритет для нескольких камер. Камеры с меньшим приоритетом отрисовываются первыми, а камеры с высоким приоритетом отрисовываются позже и могут скрыть их
Rendering Path	Определяет рендеринг для данной камеры. Оставьте эту настройку как **Use Player Settings**
Target Texture	Позволяет указать текстуру, которую камера будет отображать вместо экрана

Occlusion Culling	Отключает рендеринг объектов, когда они в настоящее время не видны камерой, потому что закрыты другими объектами
Allow HDR	Определяет, ограничены ли внутренние расчеты освещения редактора Unity основной цветовой гаммой. (HDR означает High Dynamic Range (Высокий динамический диапазон)). Это свойство позволяет использовать продвинутые визуальные эффекты
Allow MSAA	Задействует базовый, но от этого не менее эффективный тип сглаживания под названием MultiSample antialiasing. Сглаживание — это способ удаления краев пикселей во время рендеринга графики
Allow Dynamic Resolution	Позволяет динамически регулировать разрешение для консольных игр

У камер много свойств, но большую часть из них вы можете один раз установить и забыть о них. Также у камер есть несколько дополнительных компонентов. Слой бликов позволяет камере видеть блики от источников света, а компонент **Audio Listener** — захватывать звук. Если вы добавляете на сцену больше камер, вы должны удалить из них эти компоненты. На сцене может быть только один **Audio Listener**.

Использование нескольких камер

Многие эффекты в современных играх не были бы возможны без использования нескольких камер. К счастью, в редакторе Unity вы можете разместить на сцене столько камер, сколько захотите. Чтобы добавить на сцену новую камеру, выберите команду **GameObject** ⇒ **Camera**. Кроме того, вы можете добавить компонент камеры к игровому объекту, который уже есть на сцене. Чтобы сделать это, выделите объект, нажмите кнопку **Add Component** на панели **Inspector** и выберите команду **Rendering** ⇒ **Camera**. Помните, что добавление компонента камеры к существующему объекту не позволяет автоматически добавить слой бликов или компонент **Audio Listener**.

ВНИМАНИЕ!

Использование нескольких компонентов Audio Listener

Как уже упоминалось ранее, на сцене может быть только один **Audio Listener** (Микрофон). В более старых версиях Unity использование двух или более микрофонов вызовет ошибку, и сцену не получится запустить. Но сейчас, если вы используете несколько микрофонов, вы лишь увидите предупреждающее сообщение, но звук не будет восприниматься правильно. Эта тема подробно рассматривается в часе 21.

ПРАКТИКУМ

Работа с несколькими камерами

Лучший способ понять принцип взаимодействия нескольких камер — попрактиковаться в работе с ними. В этом упражнении мы сфокусируемся на базовых манипуляциях с камерой.

1. Создайте новый проект или сцену и добавьте на нее два куба. Поместите кубы в позициях с координатами (–2, 1, –5) и (2, 1, –5).
2. Переместите объект **Main Camera** в позицию с координатами (–3, 1, –8) и задайте ей поворот (0, 45, 0).
3. Добавьте на сцену новую камеру (команда **GameObject** ⇒ **Camera**) и поместите ее в позицию с координатами (3, 1, –8). Задайте ей вращение (0, 315, 0). Обязательно отключите микрофон у этой камеры, сбросив флажок рядом с соответствующим компонентом.
4. Запустите сцену. Обратите внимание, что отображается только вторая камера. Это происходит потому, что вторая камера имеет более высокое значение параметра **Depth**, чем **Main Camera**, которая отрисовывается на экране первой, а вторая камера — поверх нее. Измените глубину **Main Camera** до **1** и запустите сцену снова. Теперь работает только **Main Camera**.

Разделение экрана и «картинка-в-картинке»

Как мы видели ранее в этом часе, наличие нескольких камер на сцене — не очень хорошая идея, если одна из них выводится поверх другой. В этом разделе вы научитесь использовать свойство **Normalized Viewport Rect** для разделения экрана и создания эффекта «картинка-в-картинке».

Normalized Viewport Rect обрабатывает экран как простой прямоугольник. Левый нижний угол прямоугольника имеет координаты (0, 0), а верхний правый угол (1, 1). Но это не означает, что экран должен быть идеально квадратным, так как координаты представляют собой проценты от номинального размера. Таким образом, координата 1 означает 100%, а координата 0,5–50%. Когда вы поймете это, размещать камеры на экране станет проще простого. По умолчанию камеры выводят изображение из точки (0, 0) с шириной и высотой, равной 1 (или 100%). Поэтому изображение занимает весь экран. Но если вы измените эти параметры, вы получите другой эффект.

ПРАКТИКУМ

Создание системы камер для разделения экрана

В этом упражнении мы рассмотрим создание системы камер для разделения экрана. Такая система часто применяется в играх для двух участников, когда они пользуются одним и тем же экраном, разделенным надвое. Это упражнение основывается на предыдущем практикуме по использованию нескольких камер. Выполните следующие действия.

1. Откройте проект, созданный в предыдущем практикуме.
2. Убедитесь, что камера **Main Camera** имеет глубину –1, а свойства **X** и **Y** параметра **Viewport Rect** камеры **Main Camera** равны 0. Установите свойства **W** и **H** равными **1** и **0,5**, соответственно (то есть 100% от ширины и 50% высоты).
3. Убедитесь, что вторая камера тоже имеет глубину –1. Установите свойства **X** и **Y** параметра **Viewport Rect** равными **(0, 0,5)**. В результате камера начнет выводить изображение с середины экрана. Установите свойства **W** и **H** равными **1** и **0,5** соответственно.
4. Запустите сцену и обратите внимание, что теперь на экран выводятся обе камеры одновременно (рис. 5.6). Вы можете разделять экран столько раз, сколько захотите.

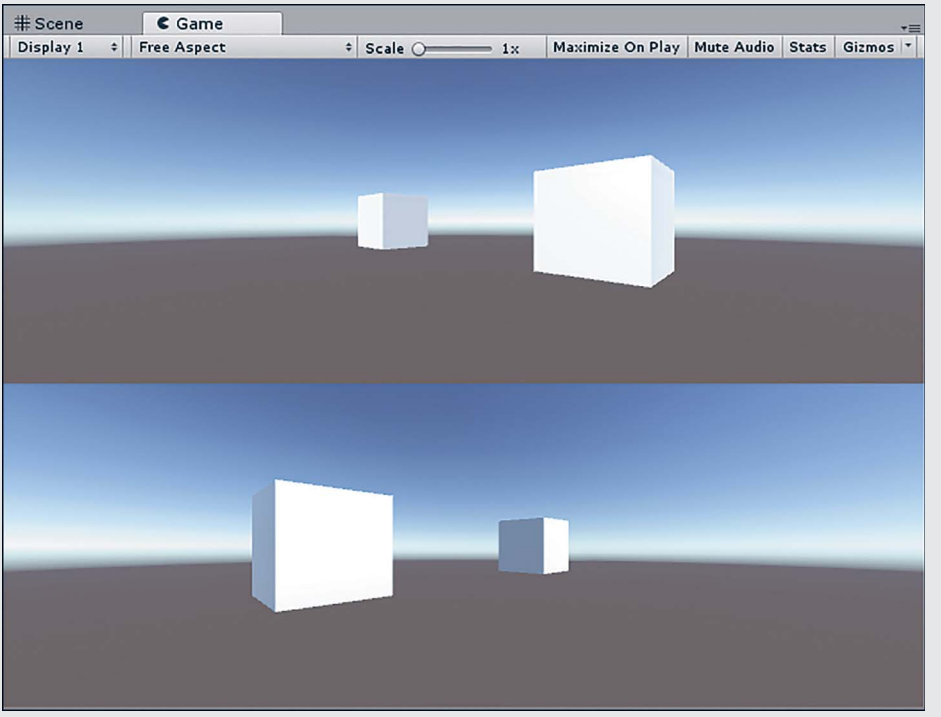


Рис. 5.6. Эффект разделения экрана

ПРАКТИКУМ

Создание эффекта «картинка-в-картинке»

Эффект «картинка-в-картинке» обычно используется для создания таких элементов, как миникарты. В этом случае одна камера выводит изображение поверх другой в определенной области. Данное упражнение основывается на том, что мы сделали ранее в практикуме по использованию нескольких камер.

1. Откройте проект, созданный в практикуме «Работа с несколькими камерами».
2. Убедитесь в том, что камера **Main Camera** имеет глубину, равную –1. Установите свойства **X** и **Y** параметра **Viewport Rect** камеры **Main Camera** равными **0**, а свойства **W** и **H** равными **1**.
3. Установите глубину второй камеры, равную **0**. Установите свойства **X** и **Y** параметра **Viewport Rect** равными **(0,75, 0,75)**, а свойства **W** и **H** — равными **0,2**.
4. Запустите сцену. Обратите внимание, что вторая камера выводит изображение в верхнем правом углу экрана (рис. 5.7). Поэкспериментируйте с настройками порта обзора, чтобы камера выводилась в разных углах.

Слои

Часто бывает трудно хорошо организовать работу с большим количеством объектов в проекте и на сцене. Иногда хочется, чтобы предметы были доступны для просмотра только определенным камерам или освещались конкретными источниками света. Иногда нужно, чтобы столкновения происходили только между определенными типами объектов. Движок Unity позволяет организовать это с помощью *слоев*. Слои — это группы схожих по свойствам объектов, которые обрабатываются движком соответствующим образом. По умолчанию в редакторе есть 8 встроенных слоев и 24 пользовательских слоя.

ВНИМАНИЕ!

Не увлекайтесь слоями!

Добавление слоев может стать отличным решением для формирования сложного поведения объектов, не требующим больших затрат сил. Но будьте осторожны: не делайте лишние слои для элементов, если этого не требуется. Слишком часто люди произвольно создают их при добавлении объектов на сцену, думая, что «возможно, этот слой мне еще пригодится».

Такой подход может привести к организационному кошмару, когда вы пытаетесь вспомнить, за что отвечает каждый слой и что он делает. Проще говоря, добавляйте слои только тогда, когда они действительно нужны, не используйте их без очевидной причины.

Работа со слоями

Все игровые объекты изначально располагаются на слое по умолчанию. Проще говоря, отдельный объект не имеет конкретного слоя и свален в кучу со всем остальным. Вы можете легко переместить объект на другой слой на панели **Inspector**. Для этого выделите объект, а затем выберите слой в раскрывающемся списке на панели **Inspector** и выберите новый слой, к которому будет отнесен объект (рис. 5.8). По умолчанию вам доступны пять слоев: **Default**, **TransparentFX**, **Ignore Raycast**, **Water** и **UI**. Сейчас вы можете смело игнорировать большинство из них, на данный момент они нам не нужны.

Имеющиеся встроенные слои не слишком полезны для вас, и вы легко можете добавлять новые. Добавление слоев выполняется в окне **Tags & Layers Manager**, которое можно открыть тремя различными способами.

- Выделив объект, в раскрывающемся списке **Layer** выберите пункт **Add Layer** (рис. 5.8).

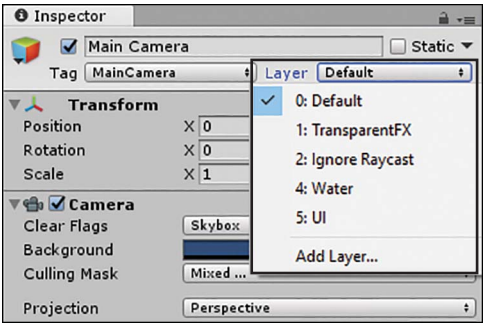


Рис. 5.8. Выбор слоя в раскрывающемся списке

- В меню в верхней части редактора выберите команду **Edit** ⇒ **Project Settings** ⇒ **Tags and Layers**.
- В раскрывающемся списке **Layers** на панели **Scene** выберите пункт **Edit Layers** (рис. 5.9).

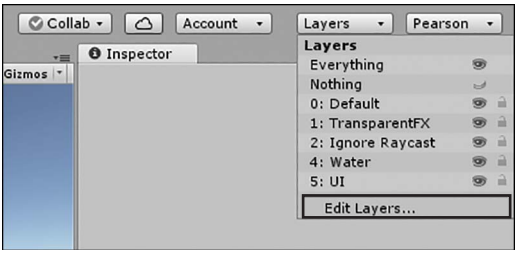


Рис. 5.9. Выбор слоев на панели **Scene**

В окне **Tags & Layers Manager** щелкните мышью на поле справа от одного из пользовательских слоев, чтобы задать ему имя. На рис. 5.10 показан этот процесс с добавлением двух новых слоев. (Это только пример для рисунка, и у вас их не будет, если вы не добавите их самостоятельно.)

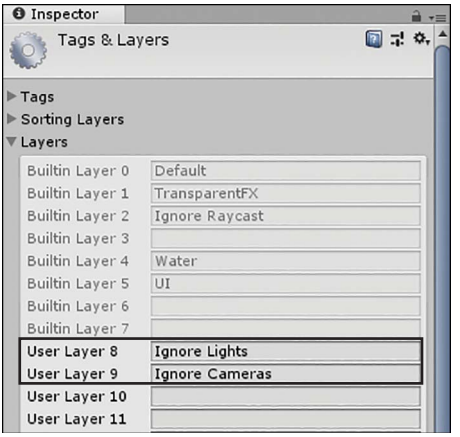


Рис. 5.10. Добавление новых слоев в окне **Tags & Layers Manager**

Использование слоев

У слоев есть много применений. Полезность слоев ограничивается только вашей фантазией. В этом разделе мы рассмотрим четыре самых типовых применения.

Первое заключается в том, чтобы скрыть слои со сцены. При нажатии на переключатель **Layers** на панели **Scene** (см. рис. 5.9) вы можете выбрать, какие слои будут отображаться на панели **Scene**, а какие — нет. По умолчанию на сцене отображаются все слои.

СОВЕТ

Невидимые элементы сцены

Частая ошибка — случайное отключение отображение слоя на сцене. Если вы не знакомы с возможностью сделать слои невидимыми, это совершенно собьет вас с толку. Важно отметить, что всякий раз, когда предметы не отображаются на панели **Scene**, хотя должны, стоит проверить селектор **Layers**, чтобы убедиться, что все слои видимы.

Второй вариант — это использовать слои для отключения освещения определенных объектов. Это полезно, если вы создаете собственный пользовательский интерфейс с применением системы затенения или сложной системы освещения. Чтобы отключить освещение слоя, выберите источник света, а затем на панели

Inspector выберите свойство **Culling Mask** и отключите все слои, которые требуется (рис. 5.11).

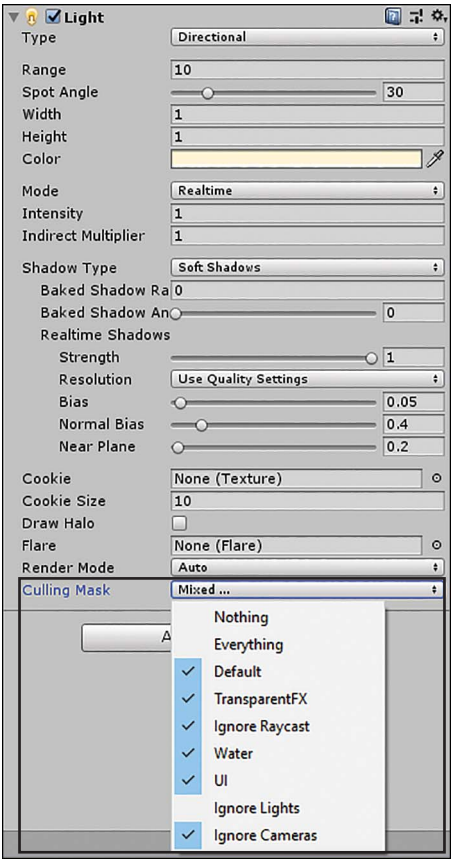


Рис. 5.11. Свойство **Culling Mask**

Третья ситуация, для которой вы можете использовать слои, — инструктировать Unity, какие физические объекты могут взаимодействовать друг с другом. Вы можете задать эти параметры, выбрав команду меню **Edit** ⇒ **Project Settings** ⇒ **Physics** и перейдя к разделу **Layer Collision Matrix** (рис. 5.12).

Последнее, что нужно знать о слоях, — вы можете использовать их, чтобы определить, что видит или не видит конкретная камера. Это полезно, если вы хотите создать пользовательский визуальный эффект с помощью нескольких камер для одного зрителя. Как описано выше для освещения, чтобы отключить отображение слоев, откройте свойство **Culling Mask** на компоненте камеры и сбросьте все флажки объектов, отображение которых вы хотите отключить.

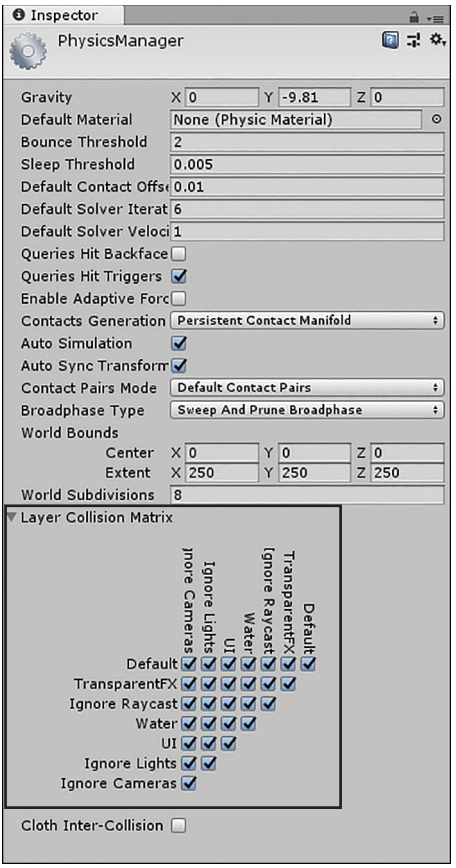


Рис. 5.12. Матрица столкновения слоев (раздел **Layer Collision Matrix**)

ПРАКТИКУМ

Игнорирование источников света и камер

Выполните приведенные ниже действия, чтобы попрактиковаться со слоями при работе с источниками света и камерами.

1. Создайте новый проект или сцену. Добавьте на сцену два куба и поместите их в позиции с координатами (2, 1, –5) и (2, 1, –5).
2. Перейдите к окну **Tags & Layers Manager** с помощью любого из трех методов, перечисленных выше, и добавьте два новых слоя, назвав их **Ignore Lights** и **Ignore Cameras** (см. рис. 5.10).
3. Выберите один из кубов и добавьте его на слой **Ignore Lights**. Затем выберите другой куб и добавьте на слой **Ignore Cameras**.

4. Выделите объект **Directional Light** на сцене, а затем в его свойстве **Culling Mask** отключите слой **Ignore Lights**. Обратите внимание, что теперь подсвечивается только один из кубов. Второй куб проигнорирован, так как находится на соответствующем слое.
5. Выделите объект **Main Camera** и отключите слой **Ignore Cameras** в его свойстве **Culling Mask**. Запустите сцену и обратите внимание, что в поле зрения камеры появляется только один неосвещенный куб. Другой куб проигнорирован камерой.

Резюме

В этом часе мы изучили свет и камеры. Мы поработали с различными типами источников света. Мы также научились добавлять затенители и гало на источники света. Кроме того, вы узнали все об основах работы с камерами и добавлении нескольких камер для создания разделения экрана и эффекта «картинка-в-картинке». В завершение часа мы изучили работу со слоями в редакторе Unity.

Вопросы и ответы

Вопрос: **Я заметил, что в этом уроке мы не рассматривали лайтмаппинг. Нужно ли изучить его?**

Ответ: Лайтмаппинг — полезный метод для оптимизации освещения сцены. Он чуть более продвинутый, и вам не обязательно уметь использовать его, чтобы придать вашим сценам отличный вид.

Вопрос: **Как понять, какая камера мне нужна: перспективная или ортографическая?**

Ответ: Как уже упоминалось в этом часе, общее правило таково: перспективная камера используется для 3D-игр или эффектов, а ортографическая — для 2D-игр и эффектов.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Если нужно осветить всю сцену одним источником света, какой тип источника использовать?

2. Сколько камер вы можете добавить на сцену?
3. Сколько пользовательских слоев можно создать?
4. Какое свойство определяет, какие слои не освещаются или не отображаются на камерах?

Ответы

1. Направленный свет — единственный тип света, который равномерно падает на всю сцену.
2. Вы можете добавить сколько угодно камер.
3. 24.
4. Свойство **Culling Mask**.

Упражнение

В этом упражнении мы поработаем с несколькими камерами и источниками света. У вас появится возможность поэкспериментировать с настройками, как вам хочется, чтобы раскрыть свой творческий потенциал.

1. Создайте новую сцену или проект. Удалите направленный свет. Добавьте на сцену сферу и поместите ее в позицию с координатами (0, 0, 0).
2. Добавьте на сцену четыре точечных источника света. Поместите их в позиции с координатами (−4, 0, 0), (4, 0, 0), (0, 0, −4), и (0, 0, 4). Задайте каждому из них свой цвет. Настройте параметры **Range** и **Intensity**, чтобы создать на сфере визуальный эффект, который вам нравится.
3. Удалите со сцены объект **Main Camera** (щелкнув правой кнопкой мыши по нему и выбрав команду **Delete**). Добавьте на сцену четыре камеры. Отключите микрофон на трех из них. Поместите их в позиции с координатами (2, 0, 0), (−2, 0, 0), (0, 0, 2), и (0, 0, −2). Поверните все камеры так, чтобы они были направлены на сферу.
4. Измените точки обзора всех камер так, чтобы добиться эффекта разделения экрана со всеми четырьмя камерами. Каждая камера должна выводить изображение в своем углу экрана, занимая одну четверть размера экрана (рис. 5.13). Этот шаг вам нужно выполнить самостоятельно. Если вы не знаете, как решить задачу, найдите это упражнение в файлах примеров к часу 5.

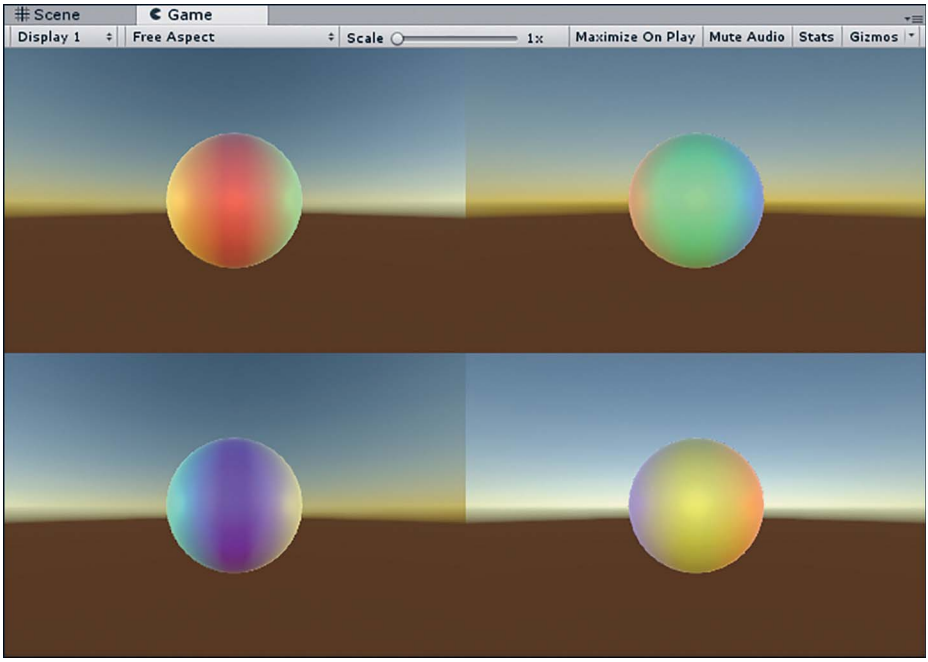


Рис. 5.13. Выполненное упражнение

6-Й ЧАС

Игра первая: «Великолепный гонщик»

Что вы узнаете в этом часе

- Как создать простую игру
- Как применить свои знания о ландшафтах, чтобы создать уникальный мир для игры
- Как добавить в игру объекты, чтобы придать ей интерактивность
- Как тестировать и настраивать законченную игру

В этом часе мы используем все приобретенные знания, чтобы создать свою первую игру на движке Unity. Сначала мы познакомимся с основными элементами дизайна игры. Затем мы создадим мир, в котором будет происходить игра. После этого мы добавим некоторые интерактивные объекты, чтобы мир превратился игру. Затем мы завершим игру и отшлифуем ее для улучшения работы.

СОВЕТ

Завершенный проект

Обязательно изучите материалы этого часа, чтобы создать полноценный игровой проект. Если у вас возникнут затруднения, найдите готовый вариант игры в файлах примеров для часа 6. Изучите его, если вам нужна помощь или вдохновение!

Проектирование

Проектирование — это этап разработки, когда вы загодя планируете все основные функции и компоненты игры. Считайте это подготовкой шаблона или плана, чтобы сам процесс создания игры шел более гладко, потому что обычно на проработку проекта уходит много времени. Поскольку игра, которую вы сделаете в этом часе, довольно простая, этап проектирования будет коротким. Чтобы создать игру, следует сосредоточиться на трех областях планирования: концепция, правила и требования.

Концепция

Идея этой игры проста: вы начинаете на одном конце некоторой области и быстро бежите на другую сторону. На вашем пути будут холмы, деревья и препятствия. Цель состоит в том, чтобы как можно быстрее достичь финишной зоны. Для первой игры выбрана именно эта концепция, так как она использует все, с чем вы работали в данной книге до сих пор. Кроме того, поскольку вы пока не освоили создание скриптов в Unity, вы еще не можете проектировать очень сложные взаимодействия. Игры, которые мы станем делать в последующих часах, будут более сложными.

Правила

В каждой игре должен быть набор правил. Правила служат двум целям. Во-первых, они определяют, как именно игрок станет действовать. Во-вторых, поскольку программное обеспечение является процессом разрешений (см. примечание «Процесс разрешения» ниже), правила определяют действия, доступные игрокам, чтобы преодолевать трудности. Правила нашей игры «Великолепный гонщик» будут следующими.

- В игре не будет условия победы или поражения, только завершение. Игра заканчивается, когда игрок попадает в финишную зону.
- Игрок всегда появляется в одном и том же месте. Зона финиша тоже находится в одном и том же месте.
- В игре будут водные преграды, и всякий раз, когда игрок попадает в одну из них, он перемещается обратно в точку спауна (появления).
- Цель игры состоит в том, чтобы достичь финиша как можно быстрее. Это правило неявно и не встроено в игру напрямую. Вместо этого в игру встроены сигналы, которые подсказывают игроку, в чем заключается цель. Идея в том, что игроки будут интуитивно стремиться достичь лучшего времени на основе получаемых ими сигналов.

ПРИМЕЧАНИЕ

Процесс разрешений

При создании игры следует всегда помнить, что программное обеспечение — это процесс разрешений. Это означает, что, если вы намеренно не разрешите что-то, оно будет недоступно игроку. Например, если игрок хочет залезть на дерево, но вы не создали для него какой-либо способ сделать это, действие не разрешается. Если вы не дадите игрокам возможность прыгать, они не смогут прыгать. Все действия, которые вы хотите разрешить игроку, должны быть явно встроены. Помните, что нельзя предполагать каких-либо действий — все должно быть спланировано! Кроме того, помните, что игроки могут весьма изобретательно комбинировать действия, например, создавая стопки ящиков, а затем прыгая с верхних ящиков, если вы сделаете это возможным.

ПРИМЕЧАНИЕ

Терминология

В этом часе мы будем использовать некоторые новые термины.

Спаун: спаунинг — это процесс, с помощью которого игрок или сущность попадает в игру.

Точка спауна: точка спауна — это место, где появляется игрок или сущность. Их может быть одна или несколько. Они могут быть стационарными или передвигаться.

Условие: условие — это своего рода триггер. Условие победы — это событие, которое дает игроку выиграть (например, набрать достаточное количество очков). Условие поражения — это событие, которое приводит к тому, что игрок проигрывает (например, потерять все очки здоровья).

Менеджер игры: менеджер игры диктует правила и управляет процессом. Он отвечает за информацию о том, когда игра выиграна или проиграна (или иным образом закончена). Любой объект может быть назначен менеджером игры, если он всегда находится на сцене. Часто менеджером игры назначается пустой объект или камера **Main Camera**.

Тестирование: тестирование — это процесс, когда реальные игроки играют в игру, которая все еще находится в разработке, чтобы посмотреть, понравится ли она людям, и что можно улучшить.

Требования

Важный шаг в процессе проектирования заключается в том, что нужно определить, какие ассеты будут необходимы для игры. Обычно команда разработчиков игры состоит из нескольких человек. Некоторые из них занимаются проектированием, другие работают с программным обеспечением, а третьи — с художественным оформлением. Каждый участник команды должен что-то делать, чтобы приносить пользу на каждом этапе процесса разработки. Если каждый станет ждать чего-то, что нужно ему для начала работы, в процессе будет много перерывов. Поэтому нужно определиться с ассетами заранее, чтобы объекты можно было создавать до того, как они пригодятся. Вот список всех требований, предъявляемых к игре «Великолепный гонщик».

- Прямоугольный участок ландшафта. Область должна быть достаточно большой, чтобы вместить сложные гонки. На ландшафте должны быть препятствия, а также точка спауна и точка финиша (рис. 6.1).
- Текстуры и эффекты окружающей среды для ландшафта. Их можно найти в стандартных ассетах Unity.
- Объект точки спауна, объект точки финиша, а также объект водных преград. Они будут генерироваться в Unity.
- Контроллер персонажа. Его можно найти в стандартных ассетах Unity.

- Графический пользовательский интерфейс (GUI). Он предоставлен в прилагаемых к книге файлах примеров. Обратите внимание, что в этом часе мы для простоты используем графический интерфейс старого стиля, который работает исключительно на основе скрипта. Вам в ваших проектах следует использовать новую систему UI, о которой мы поговорим в часе 14.
- Менеджер игры. Его мы создадим в Unity.

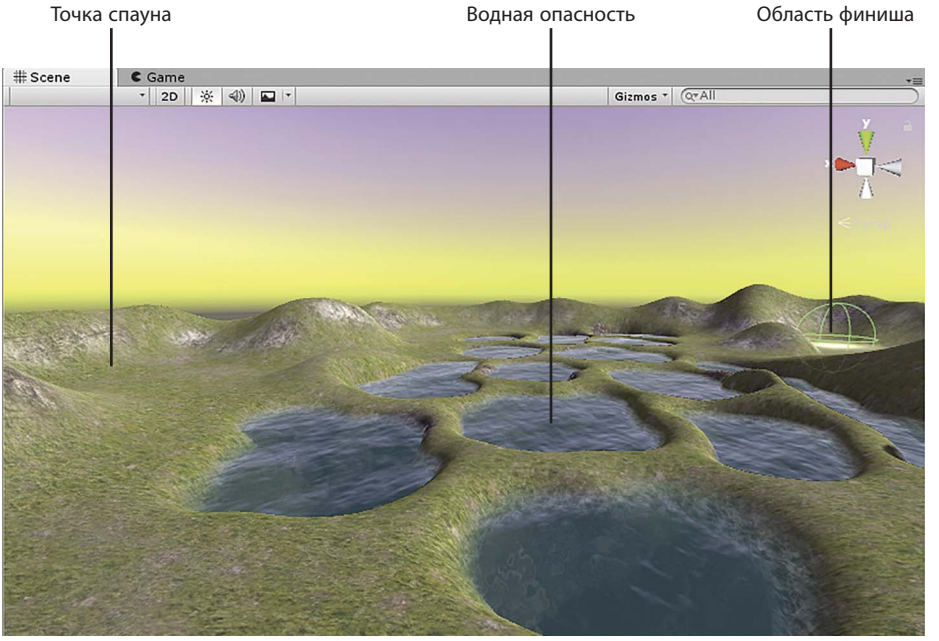


Рис. 6.1. Общая схема местности для игры «Великолепный гонщик»

Создание игрового мира

Теперь, когда у вас есть базовая идея игры, настало время создавать ее. Есть много элементов, с которых можно начать. В этом проекте мы начнем с создания мира. Поскольку это линейная гоночная игра, мир будет в длину больше, чем в ширину (или наоборот, смотря с какой стороны посмотреть). Чтобы быстро создать игру, мы будем использовать много стандартных ассетов Unity.

Создание мира

Существует много способов создать ландшафт для «Великолепного гонщика». У каждого человека, вероятно, есть свое видение данной задачи. Чтобы упростить процесс и гарантировать, что все получат аналогичный результат в течение этого

часа, вам предоставляется готовая карта высот. Чтобы создать ландшафт, выполните следующие действия.

1. Создайте новый проект и присвойте ему имя **Amazing Racer**. Добавьте в проект ландшафт и поместите его в позицию с координатами (0, 0, 0) на панели **Inspector**.
2. Найдите файл *TerrainHeightmap.raw* среди файлов примеров для часа 6. Импортируйте файл *TerrainHeightmap.raw* в качестве карты высот для ландшафта (нажав кнопку **Import Raw** в разделе **HeightMap** ⇒ **Terrain Settings** на панели **Inspector**).
3. Оставьте значения параметров **Depth**, **Width** и **Height** по умолчанию. Выберите значение **Mac** для параметра **Byte Order** и установите значения параметров **Terrain Size** равными **200** в ширину, **100** в длину и **100** в высоту.
4. Создайте папку **Scenes** в папке **Assets** и сохраните текущую сцену под названием **Main**.

Теперь ландшафт имеет рельеф и соответствует игровому миру. Вы можете внести небольшие коррективы на свой вкус.

ВНИМАНИЕ!

Создание собственного ландшафта

Мы подготовили для вас карту высот, чтобы вы смогли быстро пройти весь процесс разработки игры. Но вам ничто не мешает создать собственный мир и сделать эту игру поистине уникальной. Если вы решитесь на это, то имейте в виду, что некоторые значения координат и углов поворота, которые будут указаны здесь, могут не совпадать. Если вы хотите создать свой мир, обратите внимание на правильное размещение объектов и располагайте их в мире соответственно.

Добавление окружения

На данном этапе вы можете начать текстурирование и добавлять эффекты окружения на ваш ландшафт. Вам необходимо импортировать пакет ассетов **Environment** (выбрав команду **Assets** ⇒ **Import Package** ⇒ **Environment**).

Теперь вы можете раскрасить ваш мир так, как вам хочется. Варианты, изложенные в следующих шагах, являются лишь ориентирами. Работайте так, как вам нравится.

1. Поверните направленный свет так, как вам хочется.
2. Раскрасьте ландшафт текстурами. В примере используются следующие текстуры: **GrassHillAlbedo** для плоских поверхностей, **CliffAlbedoSpecular**

для крутых частей, **GrassRockyAlbedo** для областей между ними и **MudRockyAlbedoSpecular** для ям.

3. Добавьте на ландшафт деревья. Их должно быть немного, и расти они должны в основном на плоских поверхностях.

4. Добавьте на сцену воду из ассетов пакета **Environment**. Найдите префаб **Water4Advanced** в папке *Assets\Standard Assets\Environment\Water\Water4\Prefabs* и перетащите его на вашу сцену (вы узнаете больше о префабах в часе 11). Поместите воду в позицию с координатами (100, 29, 100) и задайте ей масштаб (2, 1, 2).

Теперь ландшафт завершен и готов к работе. Обязательно уделите достаточно времени текстурированию, чтобы достичь хорошего сочетания и реалистичного вида.

Туман

В редакторе Unity вы можете добавить на сцену туман, чтобы имитировать множество различных природных явлений, таких как дымка, реалистичный туман или пропадание объектов из видимости на больших расстояниях. Вы также можете использовать туман, чтобы придать вашему миру таинственный и чуждый вид. В игре «Великолепный гонщик» с помощью тумана вы можете скрыть отдаленные части ландшафта и добавить в игру таким образом элемент исследования.

Добавить в игру туман довольно просто.

1. Выберите пункт меню **Window** ⇒ **Lighting** ⇒ **Settings**. Откроется диалоговое окно **Lightning**.

2. Включите туман, установив флажок **Fog** в группе параметров **Other Settings**.

3. Измените цвет на белый и присвойте параметру **Density** (Плотность) значение **0,005**. (Примечание: это произвольные значения, которые могут быть изменены (или удалены) по вашему желанию).

4. Поэкспериментируйте с плотностью и цветами тумана. В таблице 6.1 описаны различные свойства тумана.

ТАБЛ. 6.1. Параметры тумана

Параметр	Описание
Fog Color	Задает цвет тумана
Fog Mode	Определяет, как рассчитывается туман. Всего три режима: **Linear** (линейный), **Exponential** (экспоненциальный) и **Exponential Squarer** (экспоненциальный в квадрате). Для мобильных устройств лучше всего работает режим **Linear**
Density	Определяет, насколько силен эффект тумана. Это свойство используется только в том случае, если задан режим тумана **Exponential** или **Exponential Squared**.
Start и End	Определяет, на каком расстоянии от камеры начинается и заканчивается отображение тумана. Эти свойства работают только в режиме **Linear**

Скайбоксы

Вы можете сделать вашу игру гораздо круче, добавив в нее *скайбокс*. Скайбокс — это «большой „ящик“, надетый поверх всего мира». На самом деле это куб, состоящий из шести плоских сторон, но его направленные внутрь текстуры делают так, что он кажется бесконечным. Вы можете создавать собственные скайбоксы или использовать стандартный скайбокс Unity, который автоматически появляется при создании 3D-сцены. В этой книге мы будем преимущественно использовать встроенные скайбоксы.

За стандарт приняты «процедурные скайбоксы». Это означает, что у них цвет не постоянный, а вычисляется и может изменяться. Вы можете увидеть это, поворачивая направленный свет в разные стороны. Обратите внимание, как изменяется цвет неба и виртуальное солнце по мере вращения источника света. Процедурные скайбоксы зависят от главного источника направленного света по умолчанию.

Создать и использовать собственный процедурный скайбокс довольно просто.

1. Щелкните правой кнопкой мыши по панели **Project** и выберите команду меню **Create** ⇒ **Material**. (Скайбоксы — это на самом деле всего лишь материалы, примененные к «гигантской коробке в небе».)
2. На панели **Inspector** откройте свойства этого нового материала, откройте раскрывающийся список **Shader** и выберите пункт **Skybox** ⇒ **Procedural**. Обратите внимание, что здесь же вы можете создать шестисторонний скайбокс, кубмап или панорамный скайбокс.
3. Примените скайбокс к сцене в окне настроек освещения (которое можно открыть, выбрав команду меню **Window** ⇒ **Lightning** ⇒ **Settings**). Кроме

того, вы можете перетащить материал скайбокса в пустое пространство на сцене. При применении скайбокса к сцене вы увидите какие-либо изменения не сразу. Вы только что создали скайбокс, который обладает теми же свойствами, что и скайбокс по умолчанию, поэтому он будет выглядеть точно так же.

4. Поэкспериментируйте с различными свойствами скайбокса. Вы можете изменить свойства солнца, поэкспериментировать с тем, как свет рассеивается в атмосфере, а также изменить цвет неба и экспозицию. Вы можете сделать для вашей игры что-то по-настоящему необычное и уникальное.

Скайбоксы не обязательно должны быть процедурными. Можно также использовать шесть текстур для создания высоко детализированного неба (которое обычно называют кубмапом). На скайбоксе также могут быть HDR- (изображения с высоким динамическим диапазоном) или панорамные изображения. Все эти параметры доступны в зависимости от типа шейдера скайбокса, который вы выбираете. Однако в этой книге вы будете иметь дело в основном с процедурными скайбоксами, потому что их легко создать и они быстро работают.

Контроллер персонажа

Теперь нам пора добавить на ландшафт контроллер персонажа.

1. Импортируйте стандартный контроллер персонажа, выбрав команду **Assets** ⇒ **Import Package** ⇒ **Characters**.

2. Найдите ассет **FPSController** в папке *Assets\Standard Assets\Characters\FirstPersonCharacter\Prefabs* и перетащите его на сцену.

3. Поместите контроллер (**FPSController**, выделенный синим на панели **Hierarchy**) в позицию с координатами (165, 32, 125). Если контроллер попал не в ту точку ландшафта, куда должен был, убедитесь, что ландшафт находится в позиции с координатами (0, 0, 0), как указано в предыдущем упражнении. Теперь поверните контроллер на 260 градусов по оси *y*, чтобы он расположился в верном направлении. Переименуйте объект контроллера, присвоив ему имя **Player**.

4. Поэкспериментируйте с объектом **First Person Controller** и компонентом **Character Controller** игрового объекта **Player**. Эти два компонента контролируют бóльшую часть того, как ведет себя персонаж внутри игры. Например, если он может забираться на холмы, которые вы хотите сделать непроходимыми, вам нужно уменьшить значение свойства **Slope Limit** компонента **Character Controller**.

5. Поскольку контроллер **Player** имеет собственную камеру, вы можете удалить со сцены объект **Main Camera**.

После того как вы правильно разместите контроллер персонажа на сцене, запустите ее. Побродите там и найдите области, которые требуют исправления или сглаживания. Обратите внимание на границы. Посмотрите, где ваш персонаж может покинуть карту. В этих местах нужно поднять рельеф или изменить настройки контроллера, чтобы игрок не мог выпасть за карту. На этом этапе разработчик обычно исправляет все основные проблемы, связанные с ландшафтом.

СОВЕТ

Падение за край карты

Обычно на игровых уровнях есть стены или некоторые другие препятствия, которые не позволяют игроку выйти за край карты. Если в игре есть гравитация, персонаж может упасть «за край света». Вам всегда нужно создавать какой-то способ запретить ему попадать в определенные места. В данном проекте используется высокий уступ, не позволяющий покидать игровую зону. На карте высот, которую мы предоставили вам в приложенных к книге файлах примеров часа 6, намеренно оставлено несколько мест, где игрок может пролезть за край. От вас требуется найти и исправить их. Вы также можете установить допустимый наклон рельефа для объекта **FPSController** на панели **Inspector**, как объяснялось ранее в этом часе.

Игрофикация

Теперь у вас есть мир, в котором будет проводиться ваша игра. Вы можете побегать по миру и опробовать его. Теперь нам не хватает, собственно говоря, самой игры. То, что у вас есть сейчас, лишь игрушка для забавы. Но нам нужна игра, в которой есть правила и конечная цель. Процесс превращения в игру называется *игрофикацией*, и именно о ней мы поговорим в данном разделе. Если вы выполнили предыдущие шаги, ваш игровой проект должен выглядеть примерно так, как показано на рис. 6.2 (хотя ваши настройки тумана, скайбокса и растительности могут иметь некоторые отличия). В течение следующих нескольких этапов мы добавим объекты управления игры, с которыми можно взаимодействовать, применим игровые скрипты к этим объектам и соединим их друг с другом.

ПРИМЕЧАНИЕ

Скрипты

Скрипты — это фрагменты кода, которые определяют поведение игровых объектов. Мы пока не изучали скрипты в Unity. Однако, чтобы создать интерактивную игру, без них не обойтись. Принимая это во внимание, мы предоставили вам необходимые скрипты, чтобы доработать эту игру. Они созданы максимально простыми, чтобы у вас была возможность понять бóльшую часть этого проекта. Откройте скрипты в текстовом редакторе и прочитайте их код, чтобы изучить, что они делают. Мы рассмотрим их более подробно в часах 7 и 8.

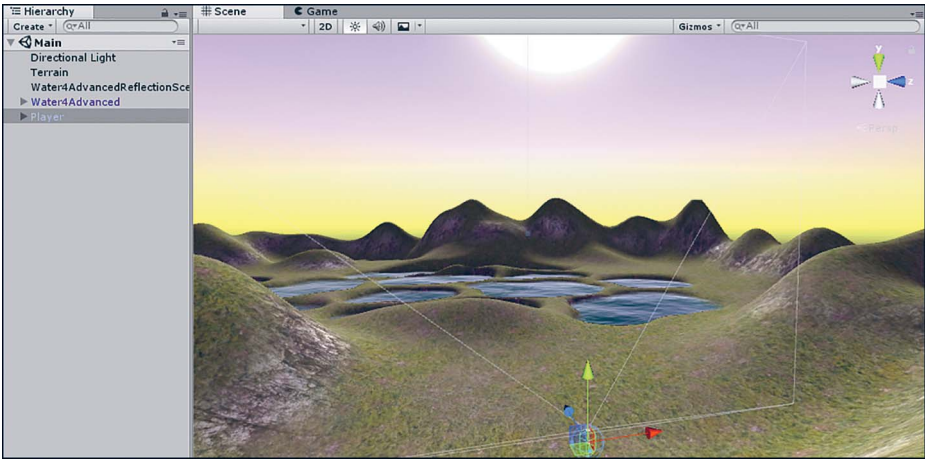


Рис. 6.2. Текущее состояние игры «Великолепный гонщик»

Добавление объектов управления игрой

Как мы сказали ранее в разделе «Требования», вам нужно четыре объекта для управления игрой. Первый — это точка спауна: простой игровой объект, который существует только для того, чтобы определить точку появления игрока. Чтобы создать точку спауна, выполните следующие действия.

1. Добавьте на сцену пустой игровой объект, выбрав команду GameObject ⇒ Create Empty.
2. Поместите этот объект в позицию с координатами (165, 32, 125) и задайте ему вращение (0, 260, 0).
3. Присвойте объекту имя **Spawn Point** на панели **Hierarchy**.

Далее вам нужно создать детектор водной опасности. Это будет обычная плоскость, расположенная под водой. Наделим эту плоскость триггером (более подробно описано в часе 9), который будет обнаруживать попадание игрока в воду. Для создания детектора выполните следующие действия.

1. Добавьте на сцену плоскость (выбрав команду **GameObject ⇒ 3D Object ⇒ Plane**) и расположите ее в позиции с координатами (100, 27, 100). Задайте плоскости масштаб (20, 1, 20).
2. Присвойте этой плоскости имя **Water Hazard Detector** на панели **Hierarchy**.
3. Установите флажки **Convex** (Выпуклая) и **Is Trigger** (Триггер) в компоненте **Mesh Collider** на панели **Inspector** (см. рис. 6.3).

4. Сделайте объект невидимым, отключив его компонент **Mesh Renderer**. Для этого сбросьте флажок **Mesh Renderer** на панели **Inspector** (рис. 6.3).

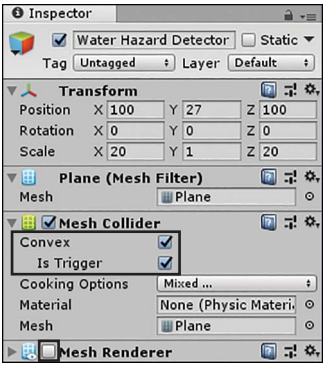


Рис. 6.3. Содержимое панели **Inspector** для объекта **Water Hazard Detector**

После этого вам нужно добавить в игру зону финиша. Это будет простой объект с точечным источником света, позволяющим игроку понять, куда нужно идти. Объект будет иметь прикрепленный к нему коллайдер, чтобы он мог среагировать, когда игрок войдет в зону. Чтобы добавить объект зоны финиша, выполните следующие действия.

1. Добавьте на сцену пустой игровой объект и поместите его в позицию с координатами (26, 32, 37).

2. Присвойте этому объекту имя **Finish Zone** на панели **Hierarchy**.

3. Добавьте объекту **Finish Zone** источник света (для этого выделите объект, а затем выберите команду **Component** ⇒ **Rendering** ⇒ **Light**). Задайте источнику света тип **Point**, если он еще не установлен, а также присвойте параметру **Range** значение **35**, а параметру **Intensity** — значение **3**.

4. Добавьте объекту **Finish Zone** компонент **Capsule Collider**, выделив объект и выбрав команду **Component** ⇒ **Physics** ⇒ **Capsule Collider**. Установите флажок **Is Trigger** и присвойте свойству **Radius** значение **9** на панели **Inspector** (рис. 6.4).

Последний объект, который нам нужно создать, — **Game Manager**, хотя его существование технически не обязательно. Вы могли бы назначить его свойства какому-либо другому объекту в игровом мире, например **Main Camera**. Но, как правило, специальный объект **Game Manager** все-таки создается, чтобы предотвратить случайное удаление. На этом этапе разработки объект **Game Manager** будет очень

простым. Мы еще воспользуемся им позднее. Чтобы создать его, выполните следующие действия.

1. Добавьте на сцену пустой игровой объект.
2. Присвойте этому объекту имя **Game Manager** на панели **Hierarchy**.

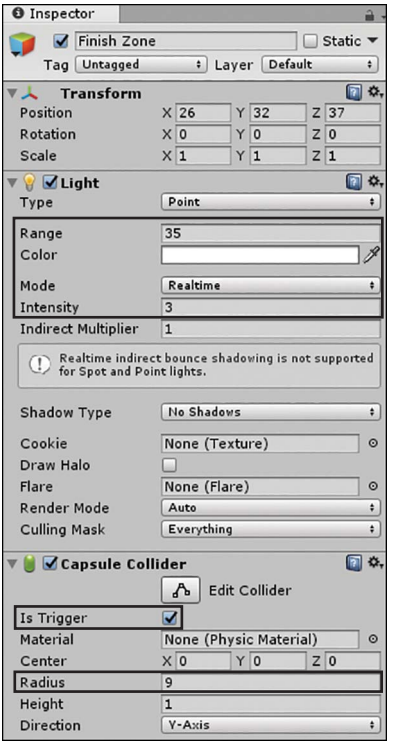


Рис. 6.4. Объект **Finish Zone** на панели **Inspector**

Добавление скриптов

Как упоминалось ранее, скрипты определяют поведение игровых объектов. В этом разделе мы применим их к нашим объектам. На данный момент понимать, что именно делают эти скрипты, для нас не так важно. Вы можете добавить скрипт в ваш проект одним из двух способов.

- Перетащить готовый скрипт на панель Project в вашем проекте.
- Создать новый скрипт в вашем проекте, щелкнув правой кнопкой мыши по панели Project и выбрав команду Create ⇒ C# Script.

Когда скрипт уже импортирован в проект, применить его проще простого. Для этого перетащите его из панели **Project** на любой объект, к которому вы хотите его применить, на панели **Hierarchy** или панели **Inspector** (рис. 6.5).

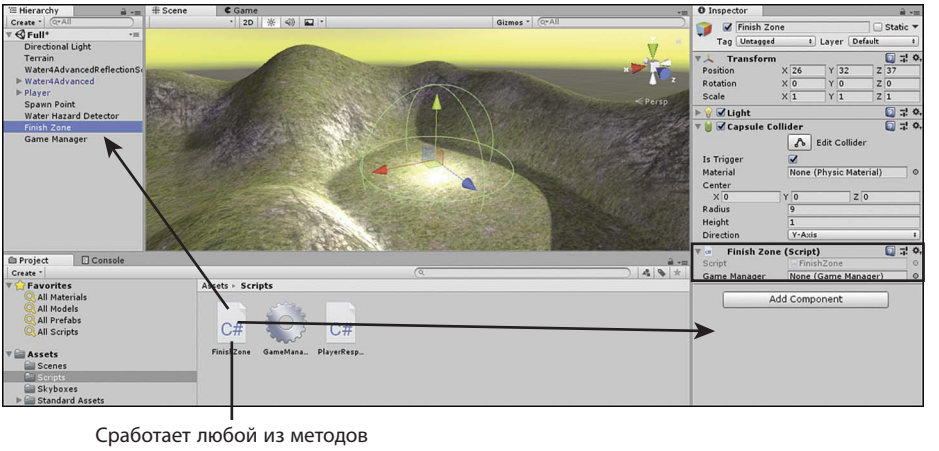


Рис. 6.5. Применение скриптов путем перетаскивания их на игровые объекты

Вы можете также применить скрипт, перетащив его на объект на сцене, но в этом случае вы рискуете попросту потерять его и случайно установить скрипт не на тот объект. Поэтому применять скрипты через панель **Scene** не рекомендуется.

СОВЕТ

Специальный значок для скрипта

Вы наверняка заметили, что у скрипта **GameManager** значок на панели **Project** не такой, как у других скриптов. Это связано с тем, что он назван именем, которое движок Unity автоматически распознает: **GameManager**. Есть несколько имен, которые, будучи использованными для скриптов, изменяют значок для облегчения их поиска.

ПРАКТИКУМ

Импорт и прикрепление скриптов

Выполните следующие действия, чтобы импортировать скрипты из файлов примеров к книге и прикрепить их к соответствующим объектам.

1. Создайте новую папку на панели **Project** и назовите ее **Scripts**. Найдите три скрипта в файлах примеров для часа 6 — *FinishZone.cs*, *GameManager.cs* и *PlayerRespawn.cs* — и перетащите их в созданную папку **Scripts**.
2. Перетащите скрипт **FinishZone.cs** с панели **Project** на игровой объект **Finish Zone** на панели **Hierarchy**.

3. Выделите объект **Game Manager** на панели **Hierarchy**. На панели **Inspector** выберите команду **Add Component** ⇒ **Scripts** ⇒ **GameManager**. (Это альтернативный способ добавления скрипта к игровому объекту.)
4. Перетащите скрипт **PlayerRespawn.cs** с панели **Project** на игровой объект **Water Hazard Detector** на панели **Hierarchy**.

Подключение скриптов

Если вы читали скрипты ранее, вы могли увидеть, что все они имеют поля для других объектов. Эти поля позволяют скриптам общаться между собой. У каждого поля есть свойство в компоненте для скрипта на панели **Inspector**. Как и в случае со скриптами, вам нужно перетащить объекты в поля (рис. 6.6).

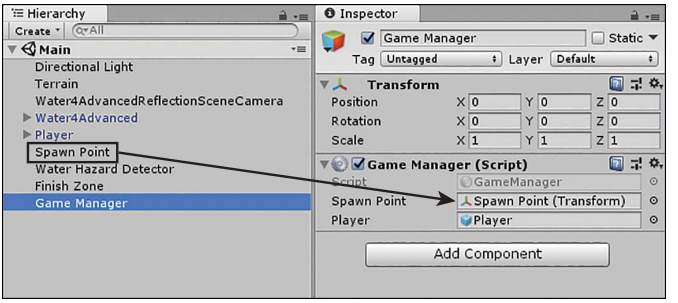


Рис. 6.6. Перемещение игрового объекта в поле

ПРАКТИКУМ

Соединение скриптов друг с другом

Выполните следующие действия, чтобы назначить игровым объектам скрипты, необходимые для их правильного функционирования.

1. Выделите объект **Water Hazard Detector** на панели **Hierarchy**. Обратите внимание, что компонент **Player Respawn** имеет свойство **Game Manager**. Это свойство выполняет роль заполнителя для объекта **Game Manager**, который вы создали ранее.
2. Перетащите объект **GameManager** с панели **Hierarchy** на свойство **Game Manager** компонента **Player Respawn (Script)**. Теперь, когда игрок попадает в водную опасность, она передаст объекту **Game Manager** информацию об этом, и игрок переместится обратно в точку спауна, в начало уровня.
3. Выберите игровой объект **Finish Zone**. Перетащите объект **Game Manager** с панели **Hierarchy** на свойство **Game Manager** компонента **Finish Zone**

(Script) на панели **Inspector**. Теперь, когда игрок попадет в зону финиша, **Game Manager** будет уведомлен об этом.

4. Выделите объект **Game Manager**. Перетащите объект **Spawn Point** на свойство **Spawn Point** компонента **Game Manager (Script)**.
5. Перетащите объект **Player** (это контроллер персонажа) на свойство **Player** объекта **Game Manager**.

Теперь все игровые объекты соединены друг с другом. Ваша игра полностью готова! Некоторые моменты, может быть, не совсем ясны вам прямо сейчас, но по мере изучения Unity все это станет интуитивно понятным.

Тестирование игрового процесса

Ваша игра готова, но отдыхать пока не время. Теперь вы должны перейти к процессу тестирования, когда вы играете в игру с целью обнаружения ошибок или моментов, которые оказались не такими веселыми, как вы ожидали. Часто бывает полезно поручить тестирование другим людям, чтобы они могли сказать вам, что им понравилось, и поделиться мыслями.

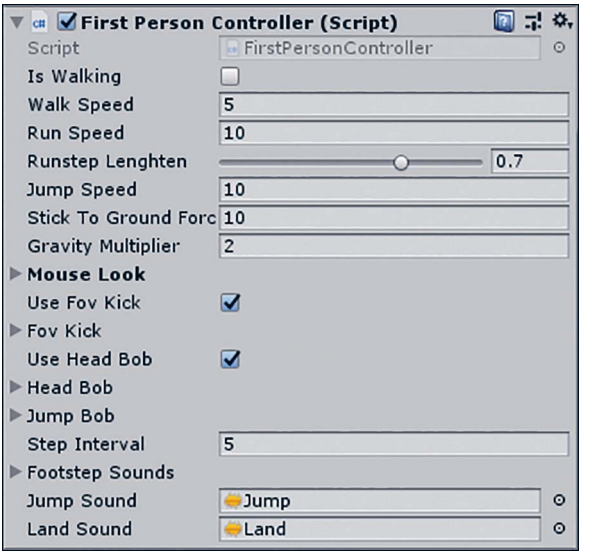


Рис. 6.7. Изменение скорости ходьбы и бега игрока

Если вы выполнили все описанные выше действия, в игре не должно быть никаких ошибок (обычно называемых багами). Определение того, что в игре весело,

а что нет, остается на усмотрение разработчика, то есть на вашей совести. Поиграйте и посмотрите, что вам не нравится. Записывайте такие моменты. Но не думайте об одном лишь негативе. Нужно найти и то, что вам нравится. Возможно, вы пока не в состоянии переделать неудачные моменты, поэтому просто запишите их. Подумайте, как вы изменили бы игру, если бы у вас была возможность.

Кое-что вы можете настроить прямо сейчас, чтобы сделать игру более приятной, а именно, скорость игрока. Поиграв несколько раз, вы, возможно, заметили, что персонаж движется довольно медленно, и игра может показаться занудной. Чтобы персонаж двигался быстрее, вам нужно изменить компонент **First Person Controller (Script)** объекта **Player**. Разверните свойство **Movement** на панели **Inspector** и измените скорость бега (см. рис. 6.7). В проекте используется значение 10. Попробуйте более быструю или медленную скорость и выберите, какая вам больше нравится. (Вы ведь обратили внимание, что, удерживая нажатой клавишу **Shift**, вы можете бегать?)

СОВЕТ

Где моя мышь?

Чтобы избежать неожиданных полетов указателя мыши, пока вы пытаетесь играть, объект **First Person Character Controller** (игровой объект **FPSController**) скрывает и «блокирует» ее. Это очень удобно во время игры, за исключением того, что вам нужна мышь, чтобы нажать кнопку или остановить игру. Если вам нужен указатель мыши, вы можете нажать клавишу **Esc**, чтобы вернуть его. С другой стороны, если вам необходимо выйти из режима игры, вы можете сделать это, нажав сочетание клавиш **Ctrl+P** (**⌘+P** в macOS).

Резюме

В этом часе вы создали свою первую игру в редакторе Unity. Сначала мы рассмотрели различные аспекты игровой концепции, а также правила и требования. Затем мы создали игровой мир и добавили эффекты среды. Затем мы добавили игровые объекты, необходимые для интерактивности. Мы применили к этим игровым объектам скрипты и соединили их между собой. И, наконец, мы протестировали готовую игру и отметили, что в ней нравится, а что нет.

Вопросы и ответы

Вопрос: **Этот урок показался мне сложноватым. Я делаю что-то неправильно?**

Ответ: Нет, что вы! Этот процесс пока что может показаться вам чуждым, так как вы не привыкли к нему. Продолжайте читать и изучать материалы,

и все ваши знания постепенно сольются воедино. Лучшее, что вы можете сделать, это обратить внимание на то, как объекты взаимодействуют друг с другом через скрипты.

Вопрос: **Вы не рассказали, как собрать и развернуть игру. Почему?**

Ответ: Сборка и развертывание игры рассматриваются далее в этой книге, в часе 23. Есть много моментов, которые необходимо учитывать при сборке игры, а нам с вами пока нужно сосредоточиться только на понятиях, необходимых для разработки.

Вопрос: **Почему нельзя создать игру без скриптов?**

Ответ: Как упоминалось ранее, скрипты определяют поведение объектов. Очень трудно получить последовательную игру без какой-либо формы интерактивности. Единственная причина, почему вы создаете игру в часе 6 до изучения часа 8 и 9, заключается в том, что нам нужно закрепить темы, которые мы уже рассмотрели, прежде чем двигаться дальше.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Что такое требования к игре?
2. Каково условие победы в игре «Великолепный гонщик»?
3. Какой объект отвечает за управление ходом игры?
4. Почему важно тестировать игру?

Ответы

1. Требования — это перечень ассетов, которые нужны, чтобы создать игру.
2. Вопрос с подвохом! В этой игре нет явного условия победы. Предполагается, что игрок выигрывает, когда достигает лучшего времени, чем в предыдущих попытках. Хотя это в игре никак не настроено.
3. Менеджер игры, который в игре «Великолепный гонщик» называется **Game Manager**.
4. Это нужно, чтобы выявить ошибки и определить, какие части игры работают правильно, а какие нет.

Упражнение

Лучшее в создании игр заключается в том, что вы делаете их так, как хотите. Следование указаниям может дать вам хороший опыт, но при этом вы не получаете удовольствия от своего творения. В данном упражнении попробуйте немного модифицировать игру «Великолепный гонщик», чтобы сделать ее уникальной. Что именно изменить, остается на ваше усмотрение. Ниже перечислены некоторые предложения.

- Попробуйте добавить несколько финишных зон и разместить их так, чтобы у игрока была альтернатива.
- Измените рельеф местности, чтобы на пути было больше различных опасностей. Если настраивать их так же, как водную преграду (включая скрипты), они будут работать нормально.
- Попробуйте создать несколько точек респауна. Сделайте так, чтобы некоторые из опасностей перемещали игрока в другую точку респауна.
- Измените небо и текстуры, чтобы создать непохожий ни на что мир. Сделайте свой мир уникальным.

7-Й ЧАС
Скрипты, часть 1

Что вы узнаете в этом часе

- Основы скриптов в Unity
- Как использовать переменные
- Как использовать операторы
- Как использовать условия
- Как использовать циклы

До сих пор, читая книгу, вы изучали, как создавать объекты в редакторе Unity. Тем не менее эти объекты были, так сказать, скучными. Насколько полезен для игры куб, который просто стоит и ничего не делает? Гораздо круче было бы наделить его какими-то пользовательскими свойствами, чтобы он стал в некотором роде интерактивным. Чтобы реализовать это, вам понадобятся скрипты: файлы с кодом, которые используются для определения сложного или нестандартного поведения объектов. В этом часе мы изучим написания скриптов. Сначала мы ознакомимся с тем, как приступить к работе с ними в Unity. Вы узнаете, как создавать скрипты и использовать среду скриптов. Затем вы узнаете о различных компонентах их языка, в том числе переменных, операторах, условиях и циклах.

СОВЕТ

Примеры скриптов

Несколько скриптов и примеров кода, перечисленных в этом часе, доступны в файлах примеров к книге для часа 7. Не забудьте изучить их, чтобы лучше понять материал.

ВНИМАНИЕ!

Новичкам в программировании

Если вы никогда не занимались программированием ранее, этот урок может показаться вам странным и непонятным. Изучая его, постарайтесь сосредоточиться на том, как структурированы новые понятия и почему именно так. Помните, что

программирование — это чистая логика. Если программа делает не то, что вы хотите, значит, вы не сказали ей, как это сделать правильно. Иногда для достижения результата вам нужно изменить свой ход мыслей. Изучайте этот час неспешно и обязательно как следует попрактикуйтесь.

Скрипты

Как упоминалось ранее в этом часе, с помощью скриптов разработчик определяет поведение объектов. Скрипты прикрепляются к объектам в Unity так же, как и другие компоненты, наделяя их интерактивностью. Как правило, работа со скриптами в Unity включает три этапа.

1. Создание скрипта.
2. Прикрепление скрипта к одному или нескольким объектам игры.
3. Если скрипт того требует, заполнение свойств значениями или другими игровыми объектами.

На протяжении урока мы обсудим эти шаги.

Создание скриптов

Прежде чем работать со скриптами, рекомендуется создать каталог **Scripts** в папке **Assets** на панели **Project**. Когда у вас появится папка, в которой будут находиться все ваши скрипты, щелкните по ней правой кнопкой мыши и выберите команду меню **Create** ⇒ **C# Script**. Затем присвойте скрипту имя.

Создав скрипт, вы можете просматривать и редактировать его. Щелкните по скрипту на панели **Project**, чтобы увидеть его содержимое на панели **Inspector** (см. рис. 7.1).

Дважды щелкнув по скрипту на панели **Project,** вы откроете редактор по умолчанию, в котором можно добавить код в скрипт. Если вы установили компоненты по умолчанию и ничего не изменяли, то по двойному щелчку откроется среда разработки Visual Studio (см. рис. 7.2).

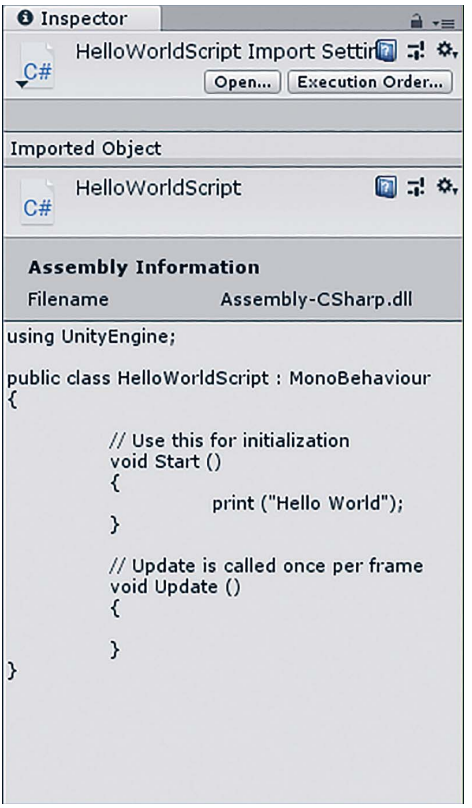


Рис. 7.1. Скрипт на панели **Inspector**

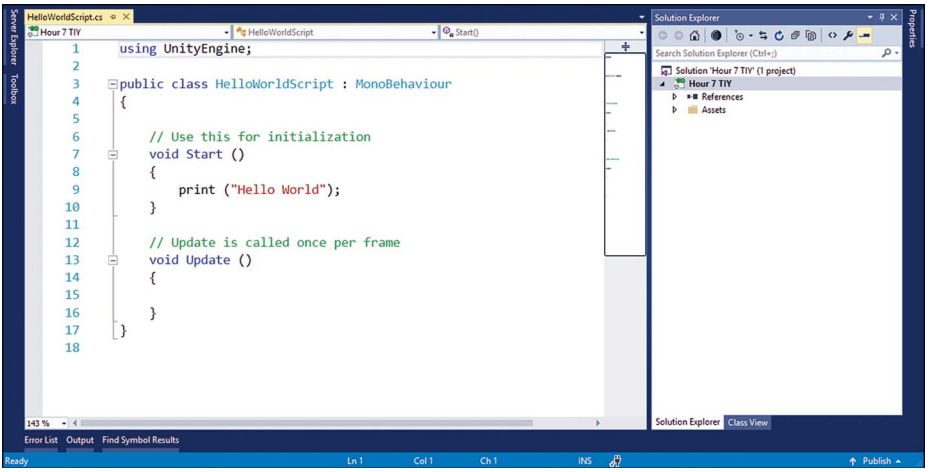


Рис. 7.2. Программное обеспечение Visual Studio с окном редактора

ПРАКТИКУМ

Создание скрипта

Выполните указанные ниже действия, чтобы создать скрипт для использования в этом разделе.

1. Создайте новый проект или сцену и создайте папку **Scripts** на панели **Project**.
2. Щелкните правой кнопкой мыши по папке **Scripts** и выберите команду **Create** ⇒ **C# Script**. Назовите скрипт **HelloWorldScript**.

Дважды щелкните по вновь созданному скрипту и дождитесь запуска среды Visual Studio. В окне редактора в Visual Studio (см. рис. 7.2) удалите весь текст и замените его следующим кодом:

```
using UnityEngine;

public class HelloWorldScript: MonoBehaviour
{
    // Инициализация
    void Start ()
    {
        print ("Hello World");
    }

    // Процедура Update вызывается 1 раз в каждом кадре
    void Update ()
    {

    }
}
```

3. Сохраните скрипт, выбрав команду меню **File** ⇒ **Save** или нажав сочетание клавиш **Ctrl+S** (⌘**+S** в macOS). Вернувшись в Unity, убедитесь, посмотрев на панель **Inspector**, что скрипт изменен, а затем запустите сцену. Обратите внимание: ничего не происходит. Мы создали скрипт, но он не работает, пока мы не прикрепим его к объекту, как описано в следующем разделе.

ПРИМЕЧАНИЕ

Имена скриптов

Вы только что создали скрипт с именем **HelloWordScript**. Важное значение имеет имя фактического файла скрипта. В Unity и C# имя файла должно совпадать с именем класса, который находится внутри него. Мы поговорим о классах позже в этом часе, но сейчас достаточно сказать, что, если у вас есть скрипт, содержащий класс с именем `MyAwesomeClass`, файл, содержащий скрипт, будет называться *MyAwesomeClass.cs*. Стоит также отметить, что классы и имена файлов скриптов не могут содержать пробелы.

ПРИМЕЧАНИЕ

О средах разработки

Visual Studio представляет собой надежный и сложный программный комплекс, который поставляется в комплекте с Unity. Подобные редакторы называются IDE (Integrated Development Environment — интегрированная среда разработки), и они помогут вам с написанием кода для игр. Поскольку IDE фактически не являются частью Unity, мы не рассматриваем их в этой книге глубоко. Единственная часть Visual Studio, с которой вам нужно познакомиться прямо сейчас, — это окно редактора. Все остальное, что вам нужно знать об IDE, мы будем рассматривать по мере необходимости. (Примечание: до версии Unity 2018.1 в комплекте с Unity поставлялась IDE под названием MonoDevelop. Сейчас вы можете приобрести и использовать это программное обеспечение отдельно, MonoDevelop больше не поставляется вместе с движком Unity.)

Назначение скрипта

Чтобы прикрепить скрипт к игровому объекту, выберите его на панели **Project** и перетащите на объект (рис. 7.3). Вы можете сделать это на панели **Hierarchy**, **Scene** или **Inspector** (при условии, что объект выделен). Прикрепившись, скрипт становится компонентом этого объекта и отображается на панели **Inspector**.

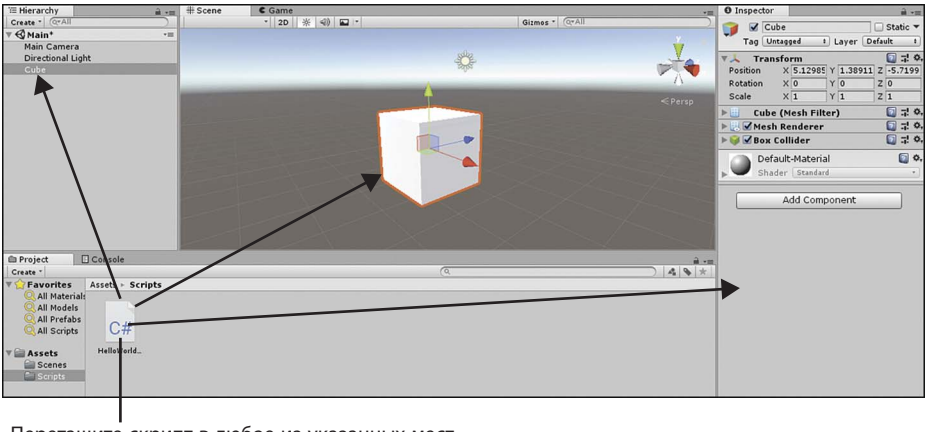


Рис. 7.3. Перетаскивание скрипта на желаемый объект

Прикрепите созданный ранее скрипт **HelloWorldScript** к объекту **Main Camera**. Теперь вы должны увидеть компонент под названием **HelloWorldScript (Script)** на панели **Inspector**. При запуске этой сцены появится надпись Hello World в нижней части редактора под панелью **Project** (рис. 7.4).

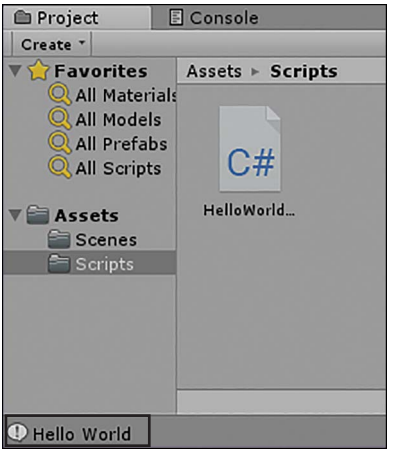


Рис. 7.4. Сообщение Hello World, выводимое при запуске сцены

Структура простого скрипта

В предыдущем разделе мы отредактировали скрипт, чтобы он выводил некий текст на экране, но не объяснили, что к чему в этом скрипте. Теперь мы рассмотрим стандартный шаблон, который применяется к каждому скрипту на языке C#. В листинге 7.1 показан полный код, который Unity генерирует для вас, когда вы создаете новый скрипт под названием **HelloWorldScript**.

ЛИСТИНГ 7.1. Код скрипта по умолчанию

```
using UnityEngine;
using System.Collections;

public class HelloWorldScript: MonoBehaviour
{
    // Инициализация
    void Start () {

    }

    // Процедура Update вызывается 1 раз в каждом кадре
    void Update () {

    }
}
```

Этот код можно разбить на три части: раздел подключения библиотек, раздел объявления классов и содержимое класса.

Раздел подключения библиотек

В первой части скрипта перечислены библиотеки, которые будут использоваться в нем. Это выглядит следующим образом:

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

Как правило, изменять этот раздел требуется редко, поэтому можно оставить его в покое на некоторое время. Эти строки обычно добавляются автоматически, когда вы создаете скрипт в редакторе Unity. Библиотека `System.Collections` необязательна и часто опускается, если скрипт не использует какой-либо ее функционал.

Раздел объявления классов

Следующая часть скрипта называется разделом объявления класса. Каждый скрипт содержит класс, имеющий то же имя. Это выглядит следующим образом:

```
public class HelloWorldScript: MonoBehaviour
```

Весь код между открывающей скобкой `{` и закрывающей скобкой `}` является частью этого класса и, следовательно, частью скрипта. Весь ваш код должен находиться между этими скобками. Как и в случае с предыдущим разделом, вам редко придется редактировать раздел объявления класса, и на данный момент вы можете вообще не трогать его.

Структура класса

Код между открывающей и закрывающей скобками помещен «внутри» класса. Весь ваш код находится здесь. По умолчанию скрипт содержит внутри класса два метода, `Start()` и `Update()`:

```
// Инициализация
void Start () {

}

// Процедура Update вызывается 1 раз в каждом кадре
void Update () {

}
```

Методы рассмотрены более подробно в часе 8. Сейчас нам достаточно знать, что любой код внутри метода `Start()` запускается сразу вместе с запуском сцены. Любой код внутри метода `Update()` работает настолько часто, насколько это возможно, иногда даже сотни раз в секунду.

СОВЕТ

Комментарии

Языки программирования позволяют программистам оставлять сообщения для тех, кто будет читать код позднее. Эти сообщения называются *комментариями*. Все, что написано после двух слешей (`//`) — комментарий. Это означает, что компьютер будет пропускать их, не пытаясь прочитать как код. Вы можете увидеть пример комментирования в практикуме «Создание скрипта», приведенном ранее в этом часе.

ПРИМЕЧАНИЕ

Консоль

В редакторе Unity есть еще одно окно, которое мы ранее ни разу не упоминали: окно консоли. В нем выводится текстовая информация игры. Часто, когда возникает ошибка или выход из скрипта, сообщения об этом выводятся в консоль. Она показана на рис. 7.5. Если окно консоли не отображается, вы можете открыть его, выбрав команду меню Window ⇒ Console.

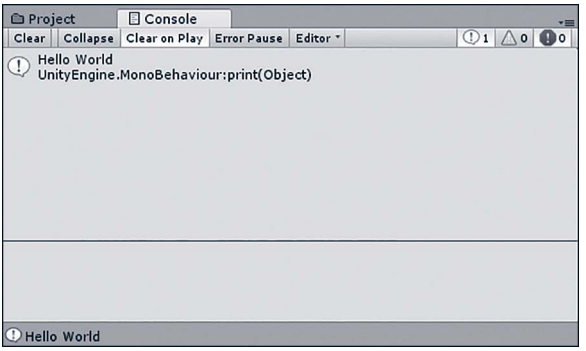


Рис. 7.5. Окно консоли

ПРАКТИКУМ

Использование встроенных методов

Теперь нам пора попробовать встроенные методы `Start()` и `Update()` и посмотреть, как они работают. Готовый скрипт ImportantFunctions доступен в приложенных к книге файлах примеров для часа 7. Постарайтесь выполнить данное упражнение самостоятельно, но если зайдете в тупик, обратитесь к файлам примеров.

1. Создайте новый проект или сцену. Добавьте в проект скрипт с именем ImportantFunctions. Дважды щелкните мышью по скрипту, чтобы открыть его в редакторе кода.
2. Внутри скрипта добавьте следующую строку кода в метод `Start()`:

```
print ("Start runs before an object Updates");
```

3. Сохраните скрипт и прикрепите его объекту Main Camera в редакторе Unity. Запустите сцену и обратите внимание на сообщение, которое появилось в окне консоли.
4. Вернитесь в Visual Studio и добавьте следующую строку кода в метод `Update()`:

```
print ("This is called once a frame");
```

5. Сохраните скрипт, а затем быстро запустите и остановите сцену в редакторе Unity. Обратите внимание, что в консоли появилась одна строка текста из метода `Start()` и много строк из метода `Update()`.

Переменные

Иногда требуется использовать один и тот же фрагмент данных в скрипте многократно. В этом случае вам необходимо создать контейнер для данных, которые можно будет использовать повторно. Такие контейнеры называются *переменными*. В отличие от традиционной математики, переменные в программировании не обязательно должны содержать числа. Они могут содержать слова, сложные объекты или другие скрипты.

Создание переменных

Каждая переменная имеет имя и тип, которые назначаются при ее создании. Вы можете создать переменную, используя следующий синтаксис:

```
<тип переменной> <имя переменной>;
```

Таким образом, чтобы создать целочисленную переменную с именем `num1`, наберите следующее:

```
int num1;
```

В таблице 7.1 перечислены все базовые (или основные) типы переменных и типы данных, которые они могут содержать.

ПРИМЕЧАНИЕ

Синтаксис

Термин *синтаксис* означает правила языка программирования. Синтаксис определяет, как нужно структурировать и писать код, чтобы компьютер понимал, как его читать. Вы, возможно, заметили, что каждый оператор или команда в скрипте заканчивается точкой с запятой. Это тоже часть синтаксиса языка C#. Если вы забудете поставить точку с запятой, то ваш скрипт не будет работать. Если хотите узнать больше, вы можете найти руководство по C# по адресу **docs.microsoft.com/ru-ru/dotnet/csharp/**.

ТАБЛ. 7.1. Типы данных в C#

Тип	Описание
`int`	Сокращенное название от integer, тип `int` хранит положительные или отрицательные целые числа
`float`	Хранит числа с плавающей точкой (например, 3,4) и используется по умолчанию в Unity. Числа с плавающей точкой в Unity всегда пишутся с буквой `f` в конце, например, `3.4f`, `0f`, `.5f` и так далее
`double`	В переменных этого типа также хранятся числа с плавающей точкой. Однако он не является типом по умолчанию в Unity. Позволяет хранить больший диапазон значений, чем `float`
`bool`	Логический тип. Сокращение от Boolean. Переменные этого типа имеют значения «правда» или «ложь» (в коде так и пишется — `true` или `false`)
`char`	Сокращение от Character. Переменные этого типа могут содержать одну букву, пробел или специальный символ (к примеру, `a`, `5` или `!`). Значения переменной заключаются в одинарные кавычки (`'a'`)
`string`	Строковый тип, содержит целые слова или предложения. Строковые значения записываются в двойных кавычках (`"Hello World"`)

Область видимости переменной

Область видимости переменной — это части кода, где переменная может быть использована. Как вы уже видели в скриптах, в классах и методах применяются открывающие и закрывающие скобки для обозначения их тела. Область между двумя скобками часто называется *блоком*. Для нас это важно, так как переменные могут быть использованы только в блоках, в которых они созданы. Таким образом, если переменная создается внутри метода `Start()`, она не будет доступна в методе `Update()`, поскольку это два различных блока. Попытка использовать переменную в блоке, где она не доступна, приведет к ошибке.

Если переменная создается в классе, но вне метода, она будет доступна для обоих методов, поскольку в этом случае они находятся в том же блоке, что и переменная (блок класса). Это показано в листинге 7.2.

ЛИСТИНГ 7.2. Демонстрация уровней блоков и класса

```
//Эта переменная находится в блоке класса
//и будет доступна везде внутри него
private int num1;

void Start ()
{
    //Эта переменная находится в "локальном блоке"
    // и будет доступна только внутри метода Start
    int num2;
}
```

Модификаторы доступа `public` и `private`

В листинге 7.2 перед переменной `num1` указано ключевое слово `private`. Это *модификатор доступа*, использование которого требуется только для переменных, объявленных на уровне класса. Вам необходимы два модификатора доступа: `private` и `public`. О них можно говорить долго, но сейчас вам достаточно знать, как они влияют на работу переменных. Принцип такой: приватная переменная (со словом `private` перед ней) может использоваться только внутри файла, в котором она создана. Другие скрипты и редактор не могут каким-либо образом видеть ее значение или изменять его. Приватные переменные предназначены только для внутреннего использования. Публичные переменные, напротив, видны другим скриптам и даже редактору Unity. Это позволяет вам легко изменить значения переменных в Unity «на лету». Если вы не обозначите переменную как публичную или приватную, то по умолчанию она будет объявлена приватной.

ПРАКТИКУМ

Изменение значений публичных переменных в Unity

Выполните следующие действия, чтобы узнать, как отображаются публичные переменные в редакторе Unity.

Создайте новый скрипт на языке C# и в Visual Studio добавьте следующую строку в классе над методом `Start()`:

```
public int runSpeed;
```

1. Сохраните скрипт, а затем в редакторе Unity прикрепите его к объекту **Main Camera**.
2. Выделите объект **Main Camera** и посмотрите на панель **Inspector**. Обратите внимание на скрипт, который вы только что прикрепили в качестве компонента. Далее обратите внимание, что у компонента появилось новое свойство: `Run Speed`. Вы можете изменить его на панели **Inspector**, и это отразится в скриптах во время выполнения. На рис. 7.6 показан компонент с новым свойством. На данном рисунке созданный скрипт назван **Important Functions**.

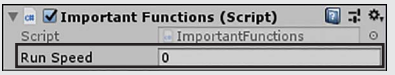


Рис. 7.6. Новое свойство `Run Speed` компонента скрипта

Операторы

Любые данные в переменных будут бесполезны, если у вас нет никакой возможности работать с ними и изменять их. *Операторы* — это специальные символы, которые позволяют модифицировать данные. В целом операторы можно разделить на четыре категории: арифметические операторы, операторы присваивания, операторы сравнения и логические операторы.

Арифметические операторы

Арифметические операторы позволяют выполнять некоторые стандартные математические действия над переменными. Они, как правило, используются только на числовых переменных, хотя есть некоторые исключения. Арифметические операторы описаны в таблице 7.2.

ТАБЛ. 7.2. Арифметические операторы

Оператор	Описание
+	Сложение. Складывает два числа. При использовании со строками текста оператор + выполняет их объединение. Ниже приведен пример: `"Hello" + "World"; //результат: "HelloWorld"`
–	Вычитание. Вычитает число справа из числа слева
*	Умножение. Умножение двух чисел
/	Деление. Делит число слева на число справа

Оператор	Описание

%	Деление по модулю. Делит число слева на число справа, но возвращает не результат, а остаток от деления. Рассмотрим следующие примеры:
	`10% 2; // возвращает 0`
	`6% 5; // возвращает 1`
	`24% 7; // возвращает 3`

Арифметические операторы могут использоваться в комбинациях, чтобы обсчитывать более сложные математические выражения, как в этом примере:

```
x + (5 * (6 – y) / 3);
```

Арифметические операторы работают в стандартном математическом порядке операций. Вычисление осуществляется слева направо, сначала выражения в скобках, затем умножение и деление и в последнюю очередь — сложение и вычитание.

Операторы присваивания

Операторы присваивания делают ровно то, что следует из названия: присваивают переменным значения. Самый часто используемый оператор присваивания — это знак равенства, но есть и такие, которые сочетают в себе несколько операций. Присваивание в языке C# выполняется справа налево. Это означает, что все, что находится справа, перемещается влево. Рассмотрим следующие примеры:

```
x = 5; // Правильно. Переменная x становится равна 5.
5 = x; // Неправильно. Вы не можете присвоить переменную значению (5).
```

В таблице 7.3 описаны операторы присваивания.

ТАБЛ. 7.3. Операторы присваивания

Оператор	Описание
=	Присваивает значение справа переменной слева

+=, –=, *=, /=	Сокращенный оператор присваивания, который выполняет некоторую арифметическую операцию, в зависимости используемого символа, а затем присваивает результат переменной слева. Рассмотрим следующие примеры:
	`x = x + 5; // Добавляет 5 к x, а затем присваивает результат x`
	`x += 5; // Делает то же самое, что и выше, только короче`

++, —	Сокращенные операторы, называемые операторами инкремента и декремента. Они увеличивают или уменьшают число на 1. Рассмотрим следующие примеры:
	`x = x + 1; // Добавляет 1 к переменной x, а затем присваивает результат переменной x`
	`x++; // Делает то же самое, что и выше, только короче`

Операторы сравнения

Эти операторы используются для сравнения двух значений. Результат оператора сравнения всегда либо истинный, либо ложный. Таким образом, единственный тип переменной, которая может содержать результат оператора сравнения — логический. (Напомню, что такая переменная может иметь значение `true` или `false`.) В таблице 7.4 описаны операторы сравнения.

ТАБЛ. 7.4. Операторы сравнения

Оператор	Описание

==	Не следует путать с оператором присваивания (=). Этот оператор возвращает истину только в случае, если два значения равны. В противном случае возвращает ложь. Рассмотрим следующие примеры:
	`5 == 6; // Возвращает false`
	`9 == 9; // Возвращает true`

>, <	Это операторы «больше» и «меньше». Рассмотрим следующие примеры:
	`5 > 3; // Возвращает true`
	`5 < 3; // Возвращает false`

>=, <=	Операторы похожи на «больше» и «меньше» за исключением того, что они означают «больше или равно» и «меньше или равно». Рассмотрим следующие примеры:
	`3 >= 3; // Возвращает true`
	`5 <= 9; // Возвращает true`

!=	Это оператор «не равно», и он возвращает истину, если два значения не равны. В противном случае возвращает ложь. Рассмотрим следующие примеры:
	`5!= 6; // Возвращает true`
	`9!= 9; // Возвращает false`

СОВЕТ

Дополнительная практика

В приложенных к книге файлах примеров для часа 7 есть скрипт с именем *EqualityAndOperations.cs*. Обязательно просмотрите его, чтобы дополнительно попрактиковаться с различными операторами.

Логические операторы

Логические операторы позволяют объединить два (или более) логических значения (`true` или `false`) в одно логическое. Эти операторы полезны для определения сложных условий. Они описаны в таблице 7.5.

ТАБЛ. 7.5. Логические операторы

Оператор	Описание

`&&`	Также известный как оператор «И», он сравнивает два логических значения и определяет, истинны ли оба. Если одно или оба ложны, этот оператор возвращает ложь. Рассмотрим следующие примеры:
	`true && false; // Возвращает false`
	`false && true; // Возвращает false`
	`false && false; // Возвращает false`
	`true && true; // Возвращает true`

`||`	Также известен как оператор «ИЛИ», сравнивает два логических значения и определяет, истинны ли оба. Если одно или оба значения истинны, этот оператор возвращает истину. Рассмотрим следующие примеры
	`true || false; // Возвращает true`
	`false || true; // Возвращает true`
	`false || false; // Возвращает false`
	`true || true; // Возвращает true`

!	Известен как оператор «НЕ», возвращает противоположное логическое значение. Рассмотрим следующие примеры:
	`!true; // Возвращает false`
	`!false; // Возвращает true`

Условные операторы

Большая часть преимуществ и мощи компьютера заключается в его способности принимать элементарные решения. В основе этой способности лежат логическая истина и ложь. Вы можете использовать логические значения для создания условий и тем самым задать программе уникальный курс. Выстраивая логический поток в коде, помните, что машина за один раз может принять только одно простое решение. Сложив достаточное количество этих решений вместе, вы можете создавать сложные взаимодействия.

Оператор `if`

Синтаксис условного оператора `if` выглядит следующим образом:

```
if (<логическое условие>)
{
    // действия
}
```

Структуру оператора `if` можно прочитать как «если это правда, то сделать это». Например, если вы хотите вывести сообщение «Hello World» в консоли, если значение переменной `x` больше `5`, можно написать следующее:

```
if (x > 5)
{
    print("Hello World");
}
```

Помните, что содержимое условия оператора должно вычисляться либо как истинное, либо как ложное. Ввод чисел, слов или чего-либо еще не даст результата:

```
if ("Hello" == "Hello")        // Правильно
if (x + y)                     // Неправильно
```

Наконец, отметим, что любой код, который вы хотите запустить, если условие истинно, должен находиться между скобками после определения оператора `if`.

СОВЕТ

Странное поведение

Условные операторы имеют специфический синтаксис и могут вести себя странно, если вы не будете соблюдать его в точности. Например, у вас в коде может быть оператор `if`, но при этом вы замечаете, что что-то работает не совсем правильно. Может быть, условие выполняется все время, даже когда вроде бы не должно. Также может сложиться ситуация, когда условие, наоборот, никогда не работает, даже если должно. Вам следует знать две самые распространенные причины такого поведения. Во-первых, у условия оператора `if` не должно быть точки с запятой после него. Если вы

укажете условие оператора с точкой с запятой, то следующий код будет работать всегда. Во-вторых, убедитесь, что вы используете оператор равенства (==), а не оператор присваивания (=) в коде условия. Нарушение этого правила приводит к странному поведению:

```
if (x > 5);  // Неправильно
if (x = 5)   // Неправильно
```

Помня об этих двух распространенных ошибках, вы сэкономите уйму времени на отладке в будущем.

Оператор `if/else`

Оператор `if` удобно использовать для условного кода, но как быть, если вы хотите разветвить вашу программу на две разные ветви? Оператор `if/else` позволяет сделать это. Такой же базовый оператор, как `if`, в смысловом плане он читается скорее как «если условие соблюдается, то делать это, а в противном случае делать что-то другое». Синтаксис оператора выглядит следующим образом:

```
if (<логическое условие>)
{
    // действия
}
else
{
    // другие действия
}
```

Например, если вы хотите вывести в консоли сообщение «X больше Y», если переменная `x` больше, чем `y`, а также вывести «Y больше, чем X», если `x` меньше `y`, то можно написать следующее:

```
if (x > y)
{
    print("X больше Y");
}
else
{
    print("Y больше X");
}
```

Оператор `if/else if`

Иногда требуется разветвить код на множество ветвей. Например, если вы хотите дать пользователю возможность выбрать что-то из большого набора опций (меню). Оператор `if/else if` имеет ту же структуру, что и предыдущие два, за исключением того, что в нем указывается несколько условий:

```
if(<условие>)
{
    // действия
}
else if (<другое условие>)
{
    // другие действия
}
else
{
    // оператор else в данном случае необязателен
    // еще другие действия
}
```

Например, если вы хотите вывести оценку на консоли в зависимости от процента выполнения задания, вы можете написать следующее:

```
if (grade >= 90) {
    print ("Оценка 5");
} else if (grade >= 80) {
    print ("Оценка 4");
} else if(grade >= 70) {
    print ("Оценка 3");
} else if (grade >= 60) {
    print ("Оценка 2");
} else {
    print ("Оценка 1");
}
```

СОВЕТ

Скобочные войны

Вы наверняка заметили, что иногда открывающие фигурные скобки размещены на отдельных строках, а иногда — на той же строке, что и имя класса, метода или оператора. Правда в том, что это совершенно не имеет значения и зависит сугубо от личных предпочтений. На тему того, как лучше писать, до сих пор ведутся грандиозные дебаты и кровавые войны. Конечно, эта проблема имеет колоссальное значение. И отсидеться в уголке здесь не получится. Каждый должен рано или поздно выбрать свою сторону!

СОВЕТ

Оператор `if` в одной строке

Строго говоря, если ваш оператор `if` написан в одну строку, вам даже фигурные скобки не нужны. Код, который выглядит следующим образом:

```
if (x > y)
{
    print("X больше Y");
}
```

может быть записан и так:

```
if (x > y)
    print("X больше Y");
```

Тем не менее на данном этапе обучения рекомендуется использовать фигурные скобки. Это избавит вас от лишней путаницы, которая может возникнуть по мере усложнения кода. Код внутри фигурных скобок называется кодовым блоком и выполняется вместе.

Циклы

Ранее вы изучили, как работать с переменными и принимать решения. Это, конечно, полезно, если вы хотите сделать что-то простое вроде сложения двух чисел. Но как быть, если вам нужно, например, сложить все числа от 1 до 100? А если от 1 до 1000? Вам вряд ли понравится печатать такой громоздкий код. Вместо этого вы можете использовать *цикл*. Есть два основных типа циклов, с которыми мы будем работать: `while` и `for`.

Цикл `while`

Цикл `while` используется чаще всего. Его структура во многом похожа на структуру оператора `if`:

```
while (<логическое условие>)
{
    // действия
}
```

Единственное отличие состоит в том, что оператор `if` выполняет код из своего тела только один раз, а цикл `while` делает это снова и снова, пока условие не станет ложным. Поэтому, если вы хотите сложить все числа от 1 до 100, а затем вывести результат в консоль, вы можете написать что-то вроде этого:

```
int sum = 0;
int count = 1;

while (count <= 100)
{
    sum += count;
    count++;
}

print(sum);
```

Как вы видите, значение переменной `count` начинается с `1` и увеличивается на `1` за каждую *итерацию* — или проход — цикла, пока не достигнет `101`. Когда

переменная `count` станет равна `101`, ее значение уже не будет меньше или равно 100, поэтому цикл завершится. Если убрать строчку `count++`, то цикл будет работать бесконечно — в этом не сомневайтесь. На каждой итерации цикла значение переменной `count` добавляется к переменной `sum`. Когда цикл заканчивается, полученная сумма выводится в консоль.

Обобщая вышесказанное, цикл `while` выполняет свой код снова и снова, пока его условие остается истинным. Когда условие станет ложным, цикл остановится.

Цикл `for`

Цикл `for` реализует ту же идею, что и цикл `while`, но имеет немного другую структуру. В коде цикла `while` нужно было создать переменную `count`, проверять ее (как условие) и увеличивать ее значение — а это три строки кода. Цикл `for` собирает весь синтаксис в одну строку. Это выглядит следующим образом:

```
for (<объявление счетчика>; <логическое условие>; <инкремент счетчика>)
{
    // действия
}
```

В заголовке цикла `for` есть три секции. Обратите внимание на точки с запятой, которыми они разделены. В первой секции создается переменная, которая будет использоваться в качестве счетчика (обычно ей присваивается имя `i`, сокращенно от iterator — счетчик итераций). Во второй секции указано условие оператора цикла. В третьей находится инкремент или декремент счетчика. Предыдущий пример можно переписать с использованием цикла `for` следующим образом:

```
int sum = 0;

for (int count = 1; count <= 100; count++)
{
    sum += count;
}

print(sum);
```

Различные части цикла теперь оформлены компактно и занимают меньше места. Очевидно, что цикл `for` действительно хорош в подсчетах.

Резюме

В этом часе мы сделали первые шаги в программировании видеоигр. Мы начали с изучения основ скриптов на движке Unity. Вы узнали, как создавать и прикреплять скрипты. Мы также взглянули на основные компоненты скрипта. Затем мы

изучили основные логические компоненты программы. Мы поработали с переменными, операторами, условиями и циклами.

Вопросы и ответы

Вопрос: **Сколько нужно программировать, чтобы создать игру?**

Ответ: В большей части игр используются некоторые формы программирования, позволяющие задать сложные модели поведения. Чем более сложные формы поведения вам нужны, тем сложнее программирование. Если хотите создавать игры, вы обязательно должны как следует ознакомиться с понятиями программирования. Это необходимо, даже если вы не собираетесь становиться основным разработчиком игры. И будьте уверены, в этой книге есть все, что вам нужно знать, чтобы создать несколько простых игр.

Вопрос: **Вы узнаете о скриптах все, что нужно?**

Ответ: И да, и нет. В данной книге вы изучите основные понятия программирования. Они всегда одинаковы, но постоянно применяются в новых и уникальных ситуациях. Тем не менее многое из того, что представлено здесь, сильно упрощено из-за сложности программирования как такового. Если вы хотите узнать больше о программировании, вам нужно читать книги или статьи по теме.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Какой язык можно использовать для программирования в Unity?
2. Верно или нет: код в методе `Start()` работает в начале каждого кадра.
3. Какой тип переменной с плавающей точкой используется в Unity по умолчанию?
4. Какой оператор возвращает остаток от деления?
5. Что такое условный оператор?
6. Какой тип цикла лучше всего подходит для подсчета?

Ответы

1. C#.
2. Нет. Метод `Start()` запускается в начале сцены. Метод `Update()` запускается каждый кадр.
3. Тип `float`.
4. Оператор деления по модулю (`%`).
5. Структура кода, которая позволяет компьютеру выбрать ветвь кода на основе оценки простого условия.
6. Цикл `for`.

Упражнение

Часто полезно представлять структуру кода в виде строительных блоков. По отдельности каждый фрагмент прост и понятен. Однако вместе они могут формировать сложные объекты. В данном упражнении вы столкнетесь с множеством проблем программирования. Используйте знания, которые вы получили в этом часе, чтобы найти решение для каждой проблемы. Поместите каждое решение в отдельный скрипт и прикрепите скрипты к объекту **Main Camera**, чтобы убедиться, что они работают. Затем вы можете посмотреть, как выполняется это упражнение, в приложенных к книге файлах примеров для часа 7.

1. Напишите скрипт, который складывает все четные числа от 2 до 499. Выведите результат в консоль.
2. Напишите скрипт, который выводит в консоль все числа от 1 до 100, кроме кратных 3 или 5. Вместо них выводите сообщение «Программирование — это прекрасно!» (Вы можете проверить кратность одного числа другому, если результат операции деления по модулю будет равен 0, например `12 % 3 == 0`, так как 12 кратно 3.)
3. В последовательности Фибоначчи каждый следующий член определяется путем сложения двух предыдущих. Начало последовательности: 0, 1, 1, 2, 3, 5 и так далее. Напишите скрипт, который вычисляет первые 20 членов последовательности Фибоначчи и выводит их в консоль.

8-Й ЧАС
Скрипты, часть 2

Что вы узнаете в этом часе

- Как писать методы
- Как захватить ввод пользователя
- Как работать с локальными компонентами
- Как работать с игровыми объектами

В часе 7 вы познакомились с основами написания скриптов в редакторе Unity. Теперь вы будете использовать полученные знания для решения более серьезных задач. Во-первых, мы изучим, что такое методы, как они работают и как их писать. Затем вы освоите несколько практических навыков работы с пользовательским вводом данных. Затем мы изучим, как получить доступ к компонентам из скриптов. В завершении этого часа вы узнаете, как получить доступ к другим игровым объектам и их компонентам с помощью скриптов.

СОВЕТ

Примеры скриптов

Несколько скриптов и примеров кода, упомянутых в этом часе, доступны в приложенных файлах примеров часа 8. Не забудьте ознакомиться с ними, чтобы получить дополнительные знания.

Методы

Методы, часто называемые *функциями*, — это блоки кода, которые могут вызываться и использоваться независимо друг от друга. Каждый метод обычно представляет собой отдельную задачу или цель, и зачастую несколько методов работают вместе для достижения сложных целей. Рассмотрим два метода, которые мы уже видели: `Start()` и `Update()`. Каждый из них решает одну короткую задачу. Метод `Start()` содержит весь код, который запускается для объекта во время запуска сцены. Метод `Update()` содержит код, который запускается каждый кадр

сцены (если он не совпадает с кадрами рендеринга, то есть «кадрами в секунду» или fps — frames per second, фреймами в секунду).

ПРИМЕЧАНИЕ

В целях сокращения...

Как вы видели до сих пор, когда мы упоминаем метод `Start()`, всегда используется слово «метод». Довольно громоздко каждый раз говорить, что речь идет именно о методе. Но вы не можете просто писать слово `Start`, потому что будет непонятно, что вы имели в виду: название кнопки, переменную или метод. Чтобы сократить запись, мы используем круглые скобки. Таким образом, фразу «метод `Start`» можно переписать как `Start()`. Если вы увидите запись `НекоеСлово()`, вы сразу поймете, что автор говорит о методе под названием `НекоеСлово`.

Структура метода

Перед началом работы с методами нужно изучить их составные части. Ниже приведен общий формат метода:

```
<возвращаемый тип> <имя метода> (<список параметров>)
{
    <код внутри блока метода>
}
```

Имя метода

Каждый метод должен иметь уникальное имя (см. раздел «Сигнатура метода» ниже). Хотя правила, регулирующие применение имен, определяются используемым языком, существуют следующие общие рекомендации.

- Сделайте имя метода описательным. Это должно быть действие или глагол.
- Пробелы не допускаются. Русские буквы тоже.
- Избегайте использования специальных символов (например, !, @, *, %, $) в именах методов. Различные языки могут применять различные символы. Отказавшись от них, можно избежать проблем.

Имена важны, поскольку они позволяют одновременно идентифицировать и использовать методы.

Возвращаемый тип

Каждый метод может возвращать переменную обратно в вызывающий код. Тип этой переменной называется *возвращаемым типом*. Если метод возвращает целое число, то тип возвращаемого значения — `int`. Аналогично, если метод возвращает значение `true` или `false`, то возвращаемый тип — `bool`. Если метод не возвращает никакого значения, у него все равно определен тип возвращаемого

значения — `void` (означает «ничто»). Любой метод, который возвращает значение, делает это с помощью ключевого слова `return`.

Список параметров

Подобно тому как методы могут передавать переменные обратно в любой код, вызвавший их, вызывающий код тоже может передавать переменные, они называются *параметрами*. Переменные, посылаемые в метод, перечислены в виде списка параметров метода. Например, метод с именем `Attack()`, который принимает целочисленную переменную с именем `enemyID` в качестве параметра, будет выглядеть следующим образом:

```
void Attack (int enemyID)
{}
```

Как видно в коде, при указании параметров необходимо указать и тип, и имя переменной. Несколько параметров разделяются запятыми.

Сигнатура метода

Сочетание типа возвращаемого значения, имени и списка параметров метода часто называют *сигнатурой метода*. Ранее в этом часе уже упоминалось, что у метода должно быть уникальное имя, хотя это не совсем верно. На самом деле метод должен иметь уникальную сигнатуру. Рассмотрим два метода:

```
void MyMethod()
{}

void MyMethod(int number)
{}
```

Несмотря на одинаковые имена, у них разные списки параметров и, следовательно, это два разных метода. Ситуация, когда существует несколько методов с одним и тем же именем, называется *перегрузкой метода*.

Блок метода

Блок метода — это место, отведенное под его код. Каждый раз, когда метод используется, выполняется код внутри блока.

ПРАКТИКУМ

Идентификация частей метода

Уделите некоторое время, чтобы изучить различные части метода, а затем рассмотрите следующий метод:

```
int TakeDamage(int damageAmount)
{
    int health = 100;
    return health — damageAmount;
}
```

Можете ли вы определить следующие части?

1. Где находится имя метода?
2. Какой тип переменной возвращает метод?
3. Каковы параметры метода? Сколько их?
4. Какой код находится в блоке метода?

СОВЕТ

Методы — это заводы

Понятие методов может показаться сложным для тех, кто только начал изучать программирование. Часто новички делают ошибки, касающиеся параметров метода и возвращаемых значений. Хороший способ разобраться в вопросе — представить, что метод — это завод. Заводы получают сырье, которое они используют для производства продукции. Методы работают точно так же. Параметры — это материалы, которые поставляются на «завод», а возвращаемое значение — конечный продукт. Представьте методы, которые не принимают никаких параметров, как заводы, которые не требуют сырья. Тогда методы, которые ничего не возвращают, — это заводы, которые не производят конечного продукта. Считая методы маленькими заводами, вы сможете легко запомнить всю эту логику.

Написание методов

Если вы понимаете компоненты метода, записать их будет легко. Но сначала попробуйте ответить на три основных вопроса:

- Для каких целей создается метод?
- Нужны ли методу какие-либо внешние данные, чтобы выполнить свою задачу?
- Должен ли метод возвращать какие-либо данные?

Ответы помогут вам определиться с именем, параметрами и возвращаемыми данными для этого метода.

Рассмотрим следующий пример: в игрока попал огненный шар. Вам нужно написать метод, чтобы учесть его воздействие путем отнятия 5 очков здоровья. Вы знаете, какова конкретная задача этого метода. Вы понимаете, что она не нуждается в каких-либо данных (потому что знаете, что игрок получит ровно 5 единиц

урона) и, вероятно, метод должен вернуть новое значение здоровья. Вы могли бы написать метод примерно так:

```
int TakeDamageFromFireball()
{
    int playerHealth = 100;
    return playerHealth – 5;
}
```

Здесь здоровье игрока составляет 100, и из него вычитается 5. Результат (95 здоровья) передается обратно. Очевидно, метод можно улучшить. Во-первых, как быть, если вы хотите, чтобы огненный шар наносил больше чем 5 очков повреждения? Вам нужно точно знать, сколько урона огненный шар может нанести в каждый момент времени. Вам понадобится переменная, или, в данном случае, параметр. Новый метод может быть записан следующим образом:

```
int TakeDamageFromFireball(int damage)
{
    int playerHealth = 100;
    return playerHealth – damage;
}
```

Теперь вы видите, что значение урона считывается из метода и применяется к здоровью. Еще можно улучшить значение здоровья. В данный момент игрок вообще не будет погибать, поскольку его здоровье всегда будет становиться равным 100 до того, как будут вычитаться повреждения. Было бы лучше, если бы здоровье игрока сохранялось в другом месте, чтобы его значение оставалось прежним. Тогда вы сможете считать значение и вычесть урон правильно. Ваш метод может выглядеть следующим образом:

```
int TakeDamageFromFireball(int damage, int playerHealth)
{
    return playerHealth – damage;
}
```

Поняв свои потребности, вы сможете создавать более совершенные и более надежные методы для вашей игры.

ПРИМЕЧАНИЕ

Упрощение

В предыдущем примере созданный нами метод выполняет обычное вычитание. Мы упростили процесс для наглядности. На самом деле существует много способов справиться с этой задачей. Здоровье игрока может храниться в переменной, принадлежащей к скрипту. В этом случае считывать его не придется. Другой вариант — использовать сложный алгоритм в методе `TakeDamageFromFireball()`, который еще снижал бы входящий урон на определенное значение в зависимости от брони,

способности игрока к уклонению или сопротивлению магии. Если наши примеры кажутся вам глупыми, имейте в виду, что они предназначены лишь для демонстрации различных аспектов темы.

Использование методов

После того как метод написан, нам остается лишь использовать его. Такой процесс часто называют *вызовом метода*. Для этого нужно указать имя метода, а затем поставить скобки и привести список параметров. Например, если вам необходимо использовать метод с именем `SomeMethod()`, можно написать следующее:

```
SomeMethod();
```

Если методу `SomeMethod()` требуется целочисленный параметр, вы можете написать так:

```
// Вызов метода со значением 5
SomeMethod(5);
// Вызов метода с помощью переменной
int x = 5;
SomeMethod(x); // здесь "int x" не пишется.
```

Обратите внимание, что при вызове метода не нужно указывать тип переменной вместе с самой переменной, которую вы передаете. Если метод `SomeMethod()` возвращает значение, вам нужно *захватить* его в переменную. Код может выглядеть примерно так (с логическим типом возвращаемого значения исключительно для примера):

```
bool result = SomeMethod();
```

Этот базовый синтаксис — все, что нужно для вызова метода.

ПРАКТИКУМ

Вызов методов

Давайте продолжим работу с методом `TakeDamageFromFireball()`, описанным в предыдущем разделе. В данном упражнении показано, как вызывать различные формы методов (вы можете найти решение в скрипте *FireBallScript* в приложенных файлах примеров для часа 8). Выполните следующие действия.

1. Создайте новый проект или сцену. Создайте скрипт C# под названием **FireBallScript** и добавьте в него три метода `TakeDamageFromFireball()`, описанных ранее. Они должны находиться внутри определения класса, на том же уровне отступа, что и методы `Start()` и `Update()`, но за пределами этих двух методов.

2. В методе `Start()` вызовите первый метод `TakeDamageFromFireball()` с помощью следующей инструкции:

```
int x = TakeDamageFromFireball();
print ("Player health: " + x);
```

3. Прикрепите скрипт к камере **Main Camera** и запустите сцену. Обратите внимание на сообщение в консоли. Теперь вызовите второй метод `TakeDamageFromFireball()` в методе `Start()`, введя следующую команду (введите ее ниже первой части кода, который вы набрали, не удаляя его):

```
int y = TakeDamageFromFireball(25);
print ("Player health: " + y);
```

4. Запустите сцену снова и обратите внимание на сообщения в консоли. Наконец, вызовите последний метод `TakeDamageFromFireball()` в методе `Start()`, набрав следующее:

```
int z = TakeDamageFromFireball(30, 50);
print ("Player health: " + z);
```

5. Запустите сцену и оцените конечный результат. Все три метода ведут себя по-разному. Также обратите внимание, что вы вызывали каждый метод отдельно, и нужная версия метода `TakeDamageFromFireball()` выбиралась в зависимости от параметров, которые вы вводили.

СОВЕТ

Помощь в поиске ошибок

Если у вас выводятся ошибки при попытке запуска скрипта, обратите внимание на номер строки и номер символа в конце сообщения об ошибке в консоли. Кроме того, вы можете «собрать» код внутри среды Visual Studio с помощью сочетания клавиш **Ctrl+Shift+B** (**⌘+Shift+B** в macOS). Когда вы это сделаете, Visual Studio проверит код и покажет вам сами ошибки и где они находятся. Попробуйте!

Ввод данных

Без ввода данных со стороны игрока видеоигра превращается в обычный фильм. Такой ввод данных может иметь самые разные формы. Устройства ввода могут быть физические, например геймпады, джойстики, клавиатуры и мыши. Существуют емкостные контроллеры, такие как относительно новые сенсорные экраны в современных мобильных устройствах. Это могут быть устройства захвата движения, такие как Wii Remote, PlayStation Move и Microsoft Kinect. Реже это аудиовход, который использует микрофоны и голос игрока для управления игрой. В этом

разделе вы узнаете все о написании кода, позволяющего игроку взаимодействовать с игрой с помощью физических устройств.

Основы ввода данных

В Unity (как и в большинстве других игровых движков) вы можете захватывать в коде нажатие определенных клавиш, чтобы сделать его интерактивным. Однако это ограничивает возможность игроков перенастроить управление под свои предпочтения, так что лучше такого не делать. К счастью, Unity имеет простую систему общего сопоставления элементов управления. В редакторе Unity вы смотрите на определенную ось, чтобы узнать, собирается ли игрок выполнить то или иное действие. Затем, когда пользователь запускает игру, он сможет задавать различные элементы управления для разных осей.

Вы можете просматривать, редактировать и добавлять оси с помощью панели **Input Manager**. Чтобы открыть ее, выберите команду меню **Edit** ⇒ **Project Settings** ⇒ **Input**. На панели **Input Manager** вы увидите различные оси, связанные с различными действиями ввода. По умолчанию существует 18 осей ввода, но вы можете добавлять и свои. На рис. 8.1 показана панель **Input Manager** с развернутой горизонтальной осью.

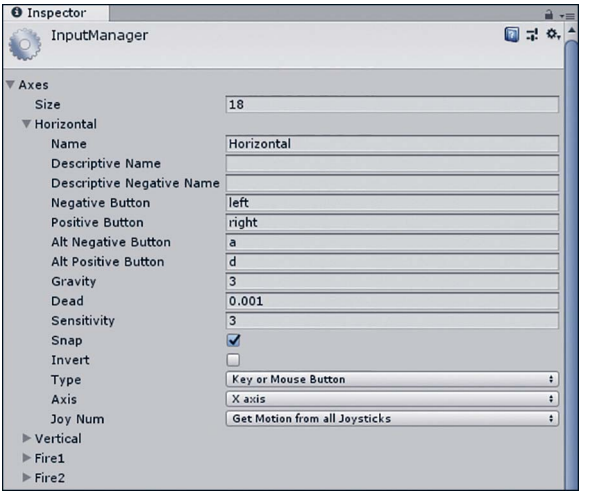


Рис. 8.1. Панель **Input Manager**

Хотя горизонтальная ось напрямую не контролирует ничего (позже вы будете писать скрипты, чтобы изменить это), она означает движение игрока вбок. В таблице 8.1 описаны свойства оси.

ТАБЛ. 8.1. Свойства оси

Свойство	Описание
Name	Имя оси. На него вы ссылаетесь в коде
Descriptive Name / Descriptive Negative Name	Подробное название оси, которое будет отображаться для игрока на экране настройки игры. Отрицательное имя — это противоположное имя. Например: «Идите налево» и «Идите направо» — это имя и противоположное имя
Negative Button / Positive Button	Клавиши, которые передают положительные и отрицательные значения оси. Для горизонтальной оси это клавиши ← и →
Alt Negative Button / Alt Positive Button	Альтернативные клавиши, которые передают значения оси. Для горизонтальной оси это клавиши A и D
Gravity	Определяет, как быстро ось возвращается к значению 0 после того, как клавиша отпущена
Dead	Значение, любой ввод ниже которого будет проигнорирован. Это помогает предотвратить ложные сигналы от дрожания, например, в джойстиках
Sensitivity	Определяет, как быстро ось реагирует на ввод
Snap	Будучи выбранной, опция Snap позволяет оси сразу перейти в положение 0, если нажата противоположная клавиша
Invert	Выбор этого свойства инвертирует управление
Type	Тип ввода. Это может быть клавиатура/мышь, движение мыши, движение джойстика
Axis	Соответствующая ось устройства ввода. Не относится к клавишам
Joy Num	Номер джойстика, от которого захватывается информация. По умолчанию получает информацию от всех джойстиков

Пользовательский ввод в скриптах

После того как ваши оси будут настроены на панели **Input Manager**, работать с ними в коде будет просто. Чтобы получить доступ к любому из входов со стороны игрока, вы будете использовать объект `Input`. Точнее говоря, метод `GetAxis()` объекта `Input`. Метод `GetAxis()` считывает имя оси в виде строки

и возвращает значение этой оси. То есть если вы хотите получить значение горизонтальной оси, введите следующую команду:

```
float hVal = Input.GetAxis("Horizontal");
```

В случае с горизонтальной осью, если игрок нажимает клавишу ← (или клавишу **A**), то метод `GetAxis()` возвращает отрицательное число. Если игрок нажимает клавишу → (или клавишу **D**), метод возвращает положительное значение.

ПРАКТИКУМ

Считывание ввода со стороны игрока

Выполните следующие действия, чтобы поработать с вертикальными и горизонтальными осями и получить лучшее представление о том, как использовать ввод со стороны игрока.

1. Создайте новый проект или сцену. Добавьте в проект скрипт под названием **PlayerInput** и прикрепите его к объекту **Main Camera**.
2. Добавьте следующий код в метод `Update()` в скрипте **PlayerInput**:

   ```
   float hVal = Input.GetAxis("Horizontal");
   float vVal = Input.GetAxis("Vertical");
   if(hVal!= 0)
       print("Horizontal movement selected: " + hVal);
   if(vVal!= 0)
       print("Vertical movement selected: " + vVal);
   ```

   Этот код должен быть в методе `Update()`, чтобы входные данные считывались непрерывно.
3. Сохраните скрипт и запустите сцену. Обратите внимание, что происходит в консоли, когда вы нажимаете клавиши со стрелками. Теперь попробуйте нажимать клавиши **W**, **A**, **S** и **D**. Если вы не видите сообщений в консоли, переключитесь на панель **Game** с помощью мыши и попробуйте еще раз.

Клавиатура

Несмотря на то что обычно хочется работать с общими осями для ввода, иногда требуется определить, была ли нажата определенная клавиша. Чтобы сделать это, снова воспользуемся объектом `Input`. Однако на этот раз задействуем метод `GetKey()`, который считывает специальный код, соответствующий определенной клавише. Затем он возвращает `true`, если клавиша нажата, или `false`, если клавиша не нажата. Чтобы определить, нажата ли в данный момент клавиша **K**, наберите следующее:

```
bool isKeyDown = Input.GetKey(KeyCode.K);
```

СОВЕТ

Поиск кодов клавиш

У каждой клавиши есть свой конкретный код. Вы можете определить код нужной вам клавиши, изучая документацию к Unity. Кроме того, вы можете использовать встроенные инструменты Visual Studio. Когда вы пишете скрипт в Visual Studio, вы всегда можете ввести имя объекта и подождать, пока появится меню со всеми возможными вариантами. Аналогичным образом, если вы ставите открывающую скобку после имени метода, появится такое же меню. На рис. 8.2 показано использование этого всплывающего меню для определения кода клавиши **Esc**.

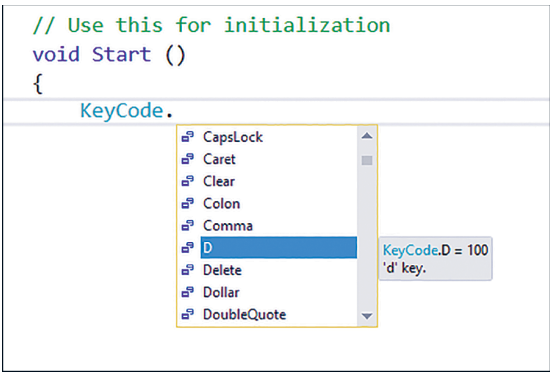


Рис. 8.2. Всплывающая подсказка в среде Visual Studio

ПРАКТИКУМ

Считывание нажатия определенных клавиш

Выполните следующие действия, чтобы написать скрипт, который определяет, нажата ли данная клавиша.

1. Создайте новый проект или сцену. Добавьте в проект скрипт под названием **PlayerInput** (или измените существующий) и прикрепите его к объекту **Main Camera**.
2. Добавьте следующий код в метод `Update()` в скрипте **PlayerInput**:

```
if(Input.GetKey(KeyCode.M))
    print("The 'M' key is pressed down");
if(Input.GetKeyDown(KeyCode.O))
    print("The 'O' key was pressed");
```

3. Сохраните скрипт и запустите сцену. Посмотрите, что происходит, когда вы нажимаете клавишу **M** и когда вы нажимаете клавишу **O** — сравните результаты. В частности, обратите внимание, что клавиша **M** выводит сообщения все время, пока она удерживается, а нажатие клавиши **O** выводится только тогда, когда она нажата первый раз.

СОВЕТ

Нажатия и удерживания

В практикуме «Считывание нажатия определенных клавиш» вы проверяли нажатие клавиши двумя различными способами. Если говорить точнее, вы использовали метод `Input.GetKey()`, чтобы проверить, нажата ли клавиша. Вы также использовали метод `Input.GetKeyDown()`, чтобы проверить, нажималась ли клавиша в текущем кадре. Во втором случае фиксировалось только первое нажатие клавиши и игнорировался тот факт, что вы удерживали клавишу нажатой. В Unity есть три метода для работы с клавишами: `GetKey()`, `GetKeyDown()` и `GetKeyUp()`. Есть и другие подобные методы: `GetButton()`, `GetButtonDown()`, GetButtonUp(), `GetMouseButton()`, `GetMouseButtonDown()` и так далее. Определение времени первого нажатия клавиши по сравнению с удерживанием клавиши весьма важно. Но какой бы тип ввода вам ни понадобился, для него найдется метод.

Мышь

Помимо нажатий клавиш, полезно захватывать нажатие кнопок мыши со стороны пользователя. Есть два компонента для ввода мыши: нажатие кнопки и движение. Определение того, нажата ли кнопка мыши, очень похоже на обнаружение нажатий клавиш, которое мы рассмотрели ранее. В этом разделе мы снова будем работать с объектом `Input`. На этот раз мы будем использовать метод `GetMouseButtonDown()`, который принимает целое число от `0` до `2`, определяющее, какая кнопка мыши вам нужна. Метод возвращает логическое значение, указывающее, нажата ли кнопка. Код для захвата нажатий кнопок мыши выглядит следующим образом:

```
bool isButtonDown;
isButtonDown = Input.GetMouseButtonDown(0);    // левая кнопка мыши
isButtonDown = Input.GetMouseButtonDown(1);    // правая кнопка мыши
isButtonDown = Input.GetMouseButtonDown(2);    // средняя кнопка мыши
```

Движение мыши осуществляется только по двум осям: *x* и *y*. Чтобы захватить движение мыши, следует задействовать метод `GetAxis()` объекта `Input`. Вы можете использовать имена `Mouse X` и `Mouse Y` для захвата движения вдоль оси *x* и оси *y* соответственно. Код для захвата движения мыши будет выглядеть следующим образом:

```
float value;
value = Input.GetAxis("Mouse X"); // движение вдоль оси x
value = Input.GetAxis("Mouse Y"); // движение вдоль оси y
```

В отличие от нажатия кнопок, движение мыши измеряется по тому, как далеко она переместилась в последнем кадре. Если вы удерживаете клавишу, значение оси будет увеличиваться, пока не достигнет `-1` или `1` (в зависимости от того,

является ли оно положительным или отрицательным). Движение мыши, как правило, возвращает меньшее число, так как оно измеряется и сбрасывается в каждом кадре.

ПРАКТИКУМ

Считывание движения мыши

В этом упражнении мы попробуем считать движение мыши и вывести результат на консоль.

1. Создайте новый проект или сцену. Добавьте в проект скрипт под названием **PlayerInput** (или измените существующий) и прикрепите его к объекту **Main Camera**.
2. Добавьте следующий код в метод `Update()` в скрипте **PlayerInput**:

```
float mxVal = Input.GetAxis("Mouse X");
float myVal = Input.GetAxis("Mouse Y");
if(mxVal != 0)
    print("Mouse X movement selected: " + mxVal);
if(myVal != 0)
    print("Mouse Y movement selected: " + myVal);
```

3. Сохраните скрипт и запустите сцену. Посмотрите, что выводится в консоли, когда вы перемещаете мышь.

Доступ к локальным компонентам

Как вы много раз видели на панели **Inspector**, объекты состоят из различных компонентов. У объекта всегда есть компонент **Transform**, а также может быть любое количество других необязательных компонентов, таких как **Renderer**, **Light** и **Camera**. Скрипты также являются компонентами, и все вместе они определяют поведение игрового объекта.

Использование метода `GetComponent()`

Вы можете взаимодействовать с компонентами во время запуска игры с помощью скриптов. Первое, что вы должны сделать, это получить ссылку на компонент, с которым хотите работать, в методе `Start()` и сохранить результат в переменной. Таким образом, вам не придется тратить время на повторение этой относительно медленной операции.

Метод `GetComponent<`*Тип*`>()` имеет несколько иной синтаксис, чем все, что вы видели ранее. В треугольных скобках указан тип компонента, который вы ищете (например, `Light`, `Camera` или имя скрипта).

Метод `GetComponent()` возвращает первый компонент указанного типа, который прикреплен к тому же игровому объекту, что и ваш скрипт. Как упоминалось ранее, вы должны назначить этот компонент локальной переменной, чтобы можно было получить доступ к нему позже. Вот как это делается:

```
Light lightComponent; // переменная, где хранится компонент Light.
Start ()
{
    lightComponent = GetComponent<Light> ();
    lightComponent.type = LightType.Directional;
}
```

Если у вас используется ссылка на компонент, вы можете легко изменить его свойства с помощью кода. Это можно сделать, введя имя переменной, ссылающейся на компонент, с точкой и именем свойства, которое хотите изменить. В приведенном выше примере можно поменять тип (`type`) свойства компонента **Light** на `Directional`.

Доступ к компоненту Transform

Компонент, с которым вы будете работать чаще всего, — это **Transform**. Изменяя его свойства, вы можете перемещать объекты по экрану. Помните, что **Transform** объекта включает его положение, углы поворота и масштаб. Вы можете изменять эти свойства непосредственно, но проще использовать некоторые встроенные методы: `Translate()`, `Rotate()` и переменную `localScale`, как показано ниже:

```
//Перемещает объект в положительном направлении оси x
// "0f" означает, что 0 имеет тип float (число с плавающей запятой).
// Так Unity считывает значения этого типа.
transform.Translate(0.05f, 0f, 0f);
// Поворот объекта вокруг оси z
transform.Rotate(0f, 0f, 1f);
// Масштабирует объект вдвое во всех направлениях
transform.localScale = new Vector3(2f, 2f, 2f);
```

ПРИМЕЧАНИЕ

Поиск компонента Transform

Поскольку каждый игровой объект имеет компонент **Transform**, нет никакой необходимости выполнять явную операцию поиска. Вы можете получить доступ к **Transform** напрямую, как описано выше. Это единственный компонент, который работает таким образом. Доступ к остальным компонентам осуществляется с помощью метода `GetComponent()`.

Поскольку `Translate()` и `Rotate()` — методы, то, если предыдущий код поместить в метод `Update()`, объект будет непрерывно перемещаться вдоль положительной оси *x* и вращаться вдоль оси *z*.

ПРАКТИКУМ

Преобразование объекта

Выполните следующие действия, чтобы попробовать в деле предыдущий код и применить его к объекту на сцене.

1. Создайте новый проект или сцену. Добавьте на сцену куб и поместите его в позицию с координатами (0, –1, 0).
2. Создайте новый скрипт и присвойте ему имя **CubeScript**. Прикрепите скрипт к кубу. В редакторе Visual Studio добавьте следующий код в метод `Update()`:

```
transform.Translate(.05f, 0f, 0f);
transform.Rotate(0f, 0f, 1f);
transform.localScale = new Vector3(1.5f, 1.5f, 1.5f);
```

3. Сохраните скрипт и запустите сцену. Вам, возможно, потребуется перейти на панель **Scene**, чтобы увидеть происходящее. Обратите внимание, что эффекты методов `Translate()` и `Rotate()` кумулятивны, а переменная `localScale` изменилась лишь один раз.

Доступ к другим объектам

Вам постоянно будет нужно, чтобы скрипт находил другие объекты и их компоненты и манипулировал ими. Весь вопрос в том, чтобы можно было найти объект, который вам нужен, и вызвать его соответствующий компонент. Есть несколько основных способов поиска объектов, которые не являются локальными для скрипта, или объектов, к которым прикреплены скрипты.

Поиск других объектов

Первый и самый простой способ находить другие объекты и работать с ними заключается в использовании редактора. Создав глобальную переменную типа `GameObject` на уровне класса, вы можете перетащить нужный объект на компонент скрипта на панели **Inspector**. Реализующий это код выглядит следующим образом:

```
// Комментарий для примера
public class SomeClassScript: MonoBehaviour
{
    // Это игровой объект, который вам нужен
```

```
    public GameObject objectYouWant;
    // Комментарий для примера
    void Start() {}
}
```

После того как вы прикрепили скрипт к игровому объекту, вы увидите на панели **Inspector** свойство под названием **Object You Want** (рис. 8.3). Перетащите любой игровой объект на это свойство, чтобы скрипт мог получить к нему доступ.

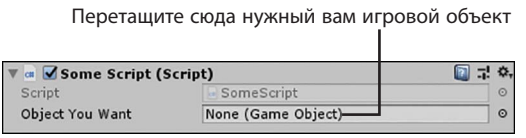


Рис. 8.3. Свойство **Object You Want** на панели **Inspector**

Другой способ найти игровой объект заключается в использовании одного из методов `Find()`. Как правило, если вы хотите, чтобы у разработчика была возможность подключить объект, или если это необязательно, вам нужно выполнять соединение с помощью панели **Inspector**. Если отсоединение приведет к неработоспособности игры, используйте метод `Find()`. Существует три основных способа поиска объекта с помощью скрипта: по имени, тегу и типу.

Первый способ — поиск по имени объекта. Имя объекта — это то, как он называется на панели **Hierarchy**. Если вы ищете объект под названием `Cube`, код будет выглядеть следующим образом:

```
// Комментарий для примера
public class SomeClassScript: MonoBehaviour
{
    // Это игровой объект, который вам нужен
    private GameObject target; // Обратите внимание, что полю не нужно
    //быть публичным при использовании метода Find
    // Комментарий для примера
    void Start()
    {
        target = GameObject.Find("Cube");
    }
}
```

Недостаток этого метода в том, что он лишь возвращает первый найденный элемент с таким именем. Если у вас есть несколько объектов с именем `Cube`, вы не будете точно знать, какой из них окажется в переменной `target`.

СОВЕТ

Эффективность поиска

Имейте в виду, что метод `Find()` снижает быстродействие, так как он проверяет каждый игровой объект в сцене, пока не найдет совпадение. На очень больших сценах количество времени, которое затрачивает этот метод, может вызвать заметное падение частоты кадров в игре. Желательно не использовать метод `Find()`, если вы можете избежать этого. Однако есть много ситуаций, когда он необходим. В таких случаях очень полезно вызывать этот метод внутри метода `Start()` и сохранять результаты поиска в переменной для использования в будущем, чтобы свести к минимуму ущерб быстродействию.

Другой способ найти объект — поиск по тегу. Тег объекта очень похож на его слой (мы говорили об этом ранее). Разница только в семантике. Слой используется для широкого спектра взаимодействий, а теги — только для базовой идентификации. Для создания тегов применяется инструмент **Tag Manager**, доступный с помощью команды меню **Edit** ⇒ **Project Settings** ⇒ **Tags & Layers**. На рис. 8.4 показано, как добавить новый тег с помощью инструмента **Tag Manager**.

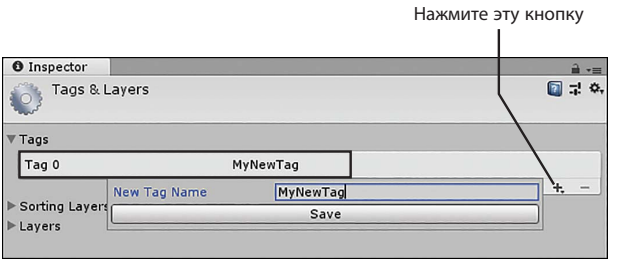


Рис. 8.4. Добавление нового тега

После создания тега можно применить его к объекту, используя раскрывающийся список тегов на панели **Inspector** (рис. 8.5).

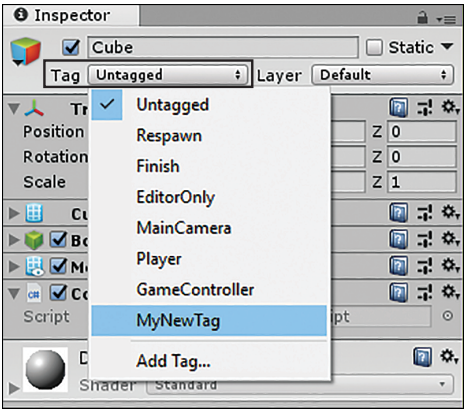


Рис. 8.5. Выбор тега

Когда тег добавлен к объекту, вы можете найти его с помощью метода `FindWithTag()`:

```
// Комментарий для примера
public class SomeClassScript: MonoBehaviour
{
    // Это игровой объект, который вам нужен
    private GameObject target;
    // Комментарий для примера
    void Start()
    {
        target = GameObject.FindWithTag("MyNewTag");
    }
}
```

Есть и другие методы `Find()`, но рассмотренных здесь вариантов должно хватить вам в большинстве ситуаций.

Изменение компонентов объектов

Если у вас есть ссылка на другой объект, то работа с его компонентами выполняется почти точно так же, как и с локальными компонентами. Единственное отличие состоит в том, что теперь вместо простого написания имени компонента вам нужно записать переменную объекта и точку перед ним, например так:

```
//Доступ к локальному компоненту — не то
transform.Translate(0, 0, 0);
//Доступ к целевому объекту — то, что надо
targetObject.transform.Translate(0, 0, 0);
```

ПРАКТИКУМ

Преобразование целевого объекта

Выполните следующие действия, чтобы преобразовать целевой объект с помощью скрипта.

1. Создайте новый проект или сцену. Добавьте на сцену куб и поместите его в позицию с координатами (0, –1, 0).
2. Создайте новый скрипт и присвойте ему имя **TargetCubeScript**. Перетащите его на объект **Main Camera**. В редакторе Visual Studio введите следующий код в скрипт **TargetCubeScript**:

```
//Игровой объект, к которому вам нужен доступ
private GameObject target;
// Комментарий для примера
void Start()
{
    target = GameObject.Find("Cube");
}
```

```
void Update()
{
    target.transform.Translate(.05f, 0f, 0f);
    target.transform.Rotate(0f, 0f, 1f);
    target.transform.localScale = new Vector3(1.5f, 1.5f, 1.5f);
}
```

3. Сохраните скрипт и запустите сцену. Обратите внимание, что куб движется по окружности, несмотря на то что скрипт прикреплен к объекту **Main Camera**.

Резюме

В этом часе мы поглубже погрузились в исследование скриптов на движке Unity. Вы узнали все о методах и изучили некоторые способы, которыми можно написать собственный метод. Мы также поработали с вводом данных со стороны игрока с помощью клавиатуры и мыши. Мы изучили, как изменять компоненты объекта с помощью скриптов, а в конце часа вы узнали, как искать и взаимодействовать с другими объектами игр с помощью скриптов.

Вопросы и ответы

Вопрос: **Как понять, сколько методов потребуется для задачи?**

Ответ: Метод должен выполнить одну короткую функцию. Плохо иметь слишком мало методов с большим количеством кода, поскольку каждый метод в таком случае будет реализовывать несколько задач. С другой стороны, нельзя иметь слишком много мелких методов, потому что это сводит на нет саму идею модульности блоков кода. Если для каждого процесса используется свой специфический метод — у вас их достаточно.

Вопрос: **Почему мы не поработали побольше с геймпадами в этом часе?**

Ответ: Проблема геймпадов заключается в том, что все они уникальны. Кроме того, различные операционные системы обрабатывают их по-разному. Мы не рассматривали их подробно в этом часе, так как они слишком разнообразны, и обсуждение разных видов нарушит последовательность изучения материала. (Кроме того, не у всех есть геймпады.)

Вопрос: **Каждый ли компонент можно отредактировать с помощью скрипта?**

Ответ: Да, по крайней мере, все встроенные компоненты.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Верно или нет: методы можно также называть функциями.
2. Верно или нет: не у каждого метода определен тип возвращаемого значения.
3. Почему плохо задавать действия игрока на отдельные кнопки?
4. В практикумах, касающихся работы с локальными и целевыми компонентами, куб перемещался вдоль положительного направления оси *x* и вращался вдоль оси *z*. При этом куб перемещался по большой окружности. Почему?

Ответы

1. Верно.
2. Нет. У каждого метода есть тип возвращаемого значения. Если метод ничего не возвращает, то тип — `void`.
3. Если вы сопоставите действия игрока с конкретными клавишами, у игроков будут проблемы с переназначением элементов управления под свои предпочтения. Если установить управление на общие оси, игроки смогут легко задать, какие клавиши будут управлять этими осями.
4. Преобразование происходит в локальной системе координат (см. час 2). Таким образом, куб передвигается вдоль положительного направления оси *x*. Но сама ось при этом поворачивается и в каждый момент времени направлена в разные стороны.

Упражнение

Было бы хорошо объединить все уроки одного часа вместе, чтобы увидеть их работу для лучшего понимания. В этом упражнении вы напишете скрипты, которые позволят игроку управлять игровым объектом. Вы можете найти решение этого упражнения в приложенных к книге файлах примеров для часа 8, если потребуется.

1. Создайте новый проект или сцену. Добавьте на сцену куб и поместите его в позицию с координатами (0, 0, −5).
2. Создайте новую папку с названием **Scripts** и новый скрипт под названием **CubeControlScript**. Присоедините скрипт к кубу.

3. Попробуйте добавить в скрипт следующие функции. Если вы зайдете в тупик, найдите решение в файлах примеров.

   - Всякий раз, когда игрок нажимает клавишу ← или →, куб должен перемещаться вдоль оси x в отрицательном или положительном направлении соответственно. Всякий раз, когда игрок нажимает клавишу ↓ или ↑, куб должен перемещаться вдоль оси z в отрицательном или положительном направлении соответственно.
   - Когда игрок перемещает мышь вдоль оси y, камера должна поворачиваться вокруг оси x. Когда игрок перемещает мышь вдоль оси x, куб должен поворачиваться вокруг оси y.
   - Когда игрок нажимает клавишу M, куб должен удвоиться в размерах. При повторном нажатии куб должен вернуться к своему первоначальному размеру. Клавиша M таким образом играет роль переключателя между размерами куба.

9-Й ЧАС
Столкновения

Что вы узнаете в этом часе

- Основы работы с твердыми телами
- Как использовать коллайдеры
- Как использовать скрипты с триггерами
- Как использовать рейкастинг

В этом часе вы научитесь работать с самой распространенной физической концепцией в видеоиграх: столкновениями. Столкновение, говоря простым языком, — это определение того, когда граница одного объекта вступает в контакт с другим объектом. Мы начнем с того, что узнаем, что такое «твердые тела» и в чем их польза для нас. После этого мы поэкспериментируем с мощными встроенными в Unity физическими движками Box2D и PhysX. Затем мы потренируемся более тонко управлять столкновениями в сочетании с триггерами. В конце часа мы научимся использовать рейкастинг для обнаружения столкновений.

Твердые тела

Чтобы объекты могли взаимодействовать с встроенными в Unity физическими движками, у них должен быть компонент под названием **Rigidbody** (Твердое тело). Добавление компонента **Rigidbody** позволяет объекту вести себя как реальный физический предмет. Чтобы добавить компонент **Rigidbody**, выделите нужный объект (убедитесь, что вы выбираете вариант без приписки «2D») и выберите команду **Component** ⇒ **Physics** ⇒ **Rigidbody**.

Новый компонент **Rigidbody** будет добавлен к объекту на панели **Inspector** (см. рис. 9.1).

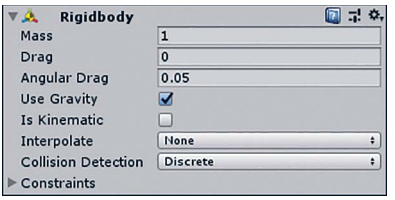


Рис. 9.1. Компонент **Rigidbody**

Компонент **Rigidbody** имеет несколько свойств, с которыми мы ранее не встречались. Они описаны в таблице 9.1.

ТАБЛ. 9.1. Свойства компонента Rigidbody

Свойство	Описание
Mass	Задает массу объекта в произвольных единицах. Используйте соотношение «1 единица = 1 кг», если нет разумной причины делать иначе. Для перемещения более тяжелого объекта потребуется бóльшая сила
Drag	Определяет величину сопротивления воздуха, применяемого к объекту при его перемещении. Высокое значение сопротивления означает, что для перемещения объекта требуется большее усилие, а остановка объекта происходит быстрее. Если значение параметра Drag равно 0, то сопротивление воздуха не применяется
Angular Drag	Аналогично с предыдущим параметром, определяет сопротивление воздуха, применяемое при вращении тела
Use Gravity	Определяет, будет ли движок Unity применять к этому объекту расчеты гравитации. Гравитация влияет на объект в большей или меньшей степени, в зависимости от его сопротивления
Is Kinematic	Позволяет вам самостоятельно контролировать перемещение твердого тела. Если объект кинематический, он не будет зависеть от примененных к нему сил
Interpolate	Определяет, сглаживается ли движение объекта и каким именно образом. По умолчанию это свойство отключено — имеет значение None. В случае с интерполяцией, сглаживание выполняется в зависимости от предыдущего кадра, а с экстраполяцией — в зависимости от следующего предполагаемого кадра. Рекомендуется включать эту опцию, только если вы замечаете задержки или рывки, а вам нужно, чтобы ваши физические объекты двигались плавно

Collision Detection	Определяет, как рассчитывается столкновение. По умолчанию выбран режим Discrete (Дискретное), и он определяет, как объекты проверяются друг относительно друга. Вариант Continuous (Непрерывное) может оказаться полезным, если у вас возникли проблемы при обнаружении столкновений с очень быстрыми объектами. Имейте в виду, что выбор режима Continuous может сильно сказаться на производительности. Режим Continuous Dynamic использует дискретное обнаружение при столкновении с другими дискретными объектами и непрерывное обнаружение для всех остальных
Constraints	Определяет ограничения движения, которые твердое тело накладывает на объект. По умолчанию эти ограничения отключены. Замораживание позиции оси не позволяет объекту перемещаться вдоль этой оси, а заморозка вращения не позволяет объекту вращаться вокруг этой оси

ПРАКТИКУМ

Использование твердых тел

Выполните следующие действия, чтобы увидеть, как работают твердые тела.

1. Создайте новый проект или сцену. Добавьте на сцену куб и поместите его в позицию с координатами (0, 1, –5).
2. Запустите сцену. Обратите внимание, как куб плавает перед камерой.
3. Добавьте кубу компонент **Rigidbody** (для этого выберите команду **Component** ⇒ **Physics** ⇒ **Rigidbody**).
4. Запустите сцену. Обратите внимание, что теперь объект падает под действием силы тяжести.
5. Поэкспериментируйте со свойствами **Drag** и **Constraints** и оцените их влияние.

Активация столкновений

Теперь, когда у вас есть движущиеся объекты, пора научить их врезаться друг в друга. Чтобы объект мог обнаруживать столкновение, ему нужен компонент **Collider**. **Collider** — это контур, который проектируется вокруг объекта и обнаруживает попадание других объектов внутрь него.

Коллайдеры

Геометрические объекты, такие как сферы, капсулы и кубы, уже имеют встроенный коллайдер сразу при создании. Вы можете добавить этот компонент объекту,

который изначально не имел его, выбрав команду **Component** ⇒ **Physics**, а затем форму коллайдера, которая вам нужна. На рис. 9.2 показаны различные формы коллайдера, доступные вам.

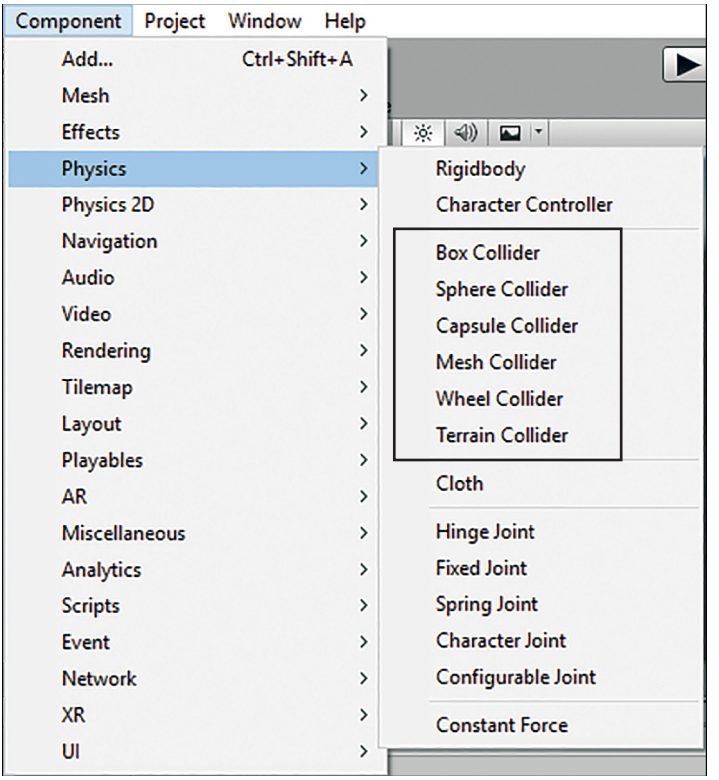


Рис. 9.2. Доступные формы коллайдера

После того как вы добавили коллайдер, на панели **Inspector** появится соответствующий объект. В таблице 9.2 описаны свойства коллайдера. Если коллайдер имеет форму сферы или капсулы, у него могут быть дополнительные свойства, например **Radius**. Подобные геометрические свойства ведут себя так, как следует из их названия.

ТАБЛ. 9.2. Свойства компонента Collider

Свойство	Описание
Edit Collider	Позволяет вам графически настраивать размер коллайдера (а в некоторых случаях и форму) на панели Scene
Is Trigger	Определяет, является коллайдер физическим коллайдером или триггером. Мы рассмотрим триггеры более подробно позже в этом часе
Material	Позволяет применять к объектам физические материалы, чтобы редактировать таким образом их поведение. Вы можете придать объекту свойства, например, дерева, металла или резины. Физика материалов будет рассмотрена далее в этом часе
Center	Определяет центральную точку коллайдера относительно содержащего его объекта
Size	Определяет размер коллайдера

СОВЕТ

Сложные комбинации коллайдеров

Используя различные формы коллайдеров, вы можете достичь весьма интересных эффектов. Например, сделав коллайдер на кубе гораздо больше самого куба, вы можете добиться того, чтобы куб словно «левитировал» над поверхностью. Аналогичным образом, используя коллайдер меньшего размера, вы можете «погрузить» куб в поверхность. Или, если вы установите сферический коллайдер, то куб сможет кататься, как мяч. Не откажите себе в удовольствии и поэкспериментируйте с различными способами создания коллайдеров для объектов.

ПРАКТИКУМ

Эксперименты с коллайдерами

Настало время опробовать некоторые виды коллайдеров и посмотреть, как они будут взаимодействовать между собой. Обязательно сохраните это упражнение, оно понадобится нам позже в этом часе. Выполните следующие действия.

1. Создайте новый проект или сцену. Добавьте на сцену два куба.
2. Поместите один из кубов в позицию с координатами (0, 1, –5) и добавьте ему компонент **Rigidbody**. Поместите второй куб в позицию с координатами (0, –1, –5) и задайте ему масштаб (4, 0,1, 4) и вращение (0, 0, 15). Добавьте компонент **Rigidbody** второму кубу, но сбросьте флажок **Use Gravity**.
3. Запустите сцену и обратите внимание, как верхний куб опустится на другой, а затем оба куба выпадут за пределы экрана.

4. Теперь выберите нижний куб и заморозьте его перемещение и вращение по всем трем осям с помощью свойств группы **Constraints**.
5. Запустите сцену и посмотрите, как верхний куб упадет на нижний и останется на нем.
6. Удалите у верхнего куба коллайдер (щелкнув правой кнопкой мыши по компоненту **Box Collider** и выбрав команду **Remove Component**) и добавьте сферический коллайдер в верхней части куба (выбрав команду **Component** ⇒ **Physics** ⇒ **Sphere Collider**). Задайте нижнему кубу угол вращения (0, 0, 350).
7. Запустите сцену. Обратите внимание, что коробка скатывается с уклона, словно это сфера, а не куб.
8. Поэкспериментируйте с различными коллайдерами. Будет довольно забавно побаловаться с ограничениями нижнего куба. Попробуйте заморозить положение по оси *y* и разморозить все остальное. Попробуйте столкнуть кубы разными способами.

СОВЕТ

Сложные коллайдеры

Вы, возможно, заметили коллайдер с названием **Mesh Collider**. Мы не будем говорить о нем отдельно в этой книге, так как сделать его трудно, а допустить ошибку — легко. В целом это такой коллайдер, форма которого определяется мешем. Если честно, другие коллайдеры (кубы, сферы и т. д.) тоже представляют собой **Mesh Collider**, но меши у них встроены сразу. **Mesh Collider** позволяет достичь очень точного обнаружения столкновений, но снижает производительность. Хорошей привычкой для вас (по крайней мере, на данном этапе обучения) будет создавать сложную форму объекта путем использования нескольких базовых коллайдеров. Если у вас есть, например, модель гуманоида, вы можете использовать сферы для головы и рук и капсулы для туловища, рук и ног. Таким образом вы сэкономите на производительности, по-прежнему получая весьма точное обнаружение столкновений.

Физические материалы

Физические материалы могут быть применены к коллайдерам объектов, чтобы наделить их различными физическими свойствами. Например, вы можете использовать резину, чтобы сделать объект упругим, или лед, чтобы сделать его скользким. Вы можете создавать и свои физические материалы для имитации нужных вам реальных.

В пакете стандартных ассетов **Characters** есть несколько готовых физических материалов. Чтобы импортировать их, выберите команду меню **Assets** ⇒ **Import Package** ⇒ **Characters**. В диалоговом окне **Import** нажмите кнопку **None**, чтобы снять выбор всех ассетов, а затем прокрутите окно вниз. Установите флажок **Physics Materials** в нижней части окна и нажмите кнопку **Import**. В результате в проекте появится папка *Assets\Standard Assets\PhysicsMaterials*, в которой

находятся физические материалы **Bouncy**, **Ice**, **Metal**, **Rubber** и **Tree** (т. е. упругий материал, лед, металл, резина и дерево). Чтобы создать новый физический материал, щелкните правой кнопкой мыши по папке **Assets** на панели **Project** и выберите команду **Create** ⇒ **Physic Material** или **Create** ⇒ **Physics Material 2D** (если вы работаете с двумерной физикой, о которой говорится в часе 12).

У каждого материала есть набор свойств, которые определяют его поведение на физическом уровне (рис. 9.3). В таблице 9.3 описаны эти свойства. Вы можете применить физический материал к объекту, перетащив его из панели **Project** на объект с коллайдером.

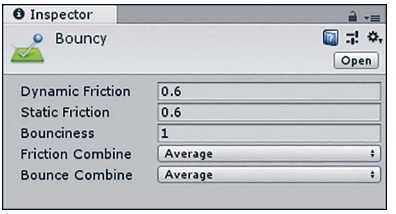


Рис. 9.3. Свойства физических материалов

ТАБЛ. 9.3. Свойства физических материалов

Свойство	Описание
Dynamic Friction	Задает трение, которое применяется, когда объект уже движется. Чем меньше значение, тем более скользким является объект
Static Friction	Задает трение, которое применяется, когда объект находится в неподвижном состоянии. Чем меньше значение, тем лучше скользит объект
Bounciness	Определяет количество энергии, которое объект сохраняет после столкновения. При значении 1 объект отскакивает без потери энергии, то есть он абсолютно упругий. Значение 0 делает объект абсолютно неупругим и не позволяет отскакивать
Friction Combine	Определяет, как вычисляется трение двух сталкивающихся объектов. Трение может быть усреднено, может использоваться наименьшее или наибольшее трение, а также произведение значений трения
Bounce Combine	Определяет, как рассчитывается упругость для двух сталкивающихся объектов. Может использоваться среднее, наименьшее или наибольшее значение упругости, а также произведение значений упругости

Эффекты физического материала могут быть более или менее выраженными, в зависимости от ваших предпочтений. Попробуйте сами и посмотрите, какое интересное поведение у вас получится создать.

Триггеры

До сих пор мы изучали физические коллайдеры, то есть те, которые взаимодействуют при перемещении и вращении с помощью встроенного в Unity физического движка. Однако в часе 6, создавая «Великолепного гонщика», мы использовали другой тип коллайдера. Помните, как игра обнаруживала попадание игрока в воду или финишную зону? Это было реализовано с помощью коллайдера-триггера. Триггер обнаруживает столкновение так же, как и обычный коллайдер, но затем он вызывает три метода, которые позволяют вам, то есть программисту, решить, что будет дальше:

```
void OnTriggerEnter(Collider other) // вызывается, когда объект входит в триггер
void OnTriggerStay(Collider other) // вызывается, когда объект находится внутри
                                   триггера
void OnTriggerExit(Collider other) // вызывается, когда объект покидает триггер
```

Таким образом вы можете определить, что должно происходить всякий раз, когда объект входит в коллайдер, находится внутри или покидает его. Например, если вы хотите вывести в консоль сообщение о том, что объект попал в контур куба, вы можете добавить в куб триггер, а затем прикрепить к кубу скрипт со следующим кодом:

```
void OnTriggerEnter(Collider other)
{
    print("Object has entered collider");
}
```

Вы могли заметить, что у метода триггера есть параметр: переменная `other` типа `Collider`. Это ссылка на объект, который попадает в триггер. Используя переменную, вы можете манипулировать объектом, как вам хочется. Например, вы можете изменить код, чтобы он выводил имя объекта, который попал в триггер. Для этого введите следующее:

```
void OnTriggerEnter(Collider other)
{
    print(other.gameObject.name + " has entered the trigger");
}
```

А если совсем разойдетесь, вы можете даже уничтожить объект, попавший в триггер:

```
void OnTriggerEnter(Collider other)
{
    Destroy(other.gameObject);
}
```

ПРАКТИКУМ

Работа с триггерами

В этом упражнении мы создадим интерактивную сцену с работающим триггером.

1. Создайте новый проект или сцену. Добавьте на сцену куб и сферу.
2. Поместите куб в позицию с координатами (–1, 1, –5), а сферу — в позицию с координатами (1, 1, –5).
3. Создайте два скрипта с именами **TriggerScript** и **MovementScript**. Прикрепите скрипт **TriggerScript** к кубу, а **MovementScript** — к сфере.
4. Включите параметр **Is Trigger** в компоненте **Collider** куба. Добавьте в сферу компонент **Rigidbody** и сбросьте флажок **Use Gravity**.
5. Добавьте следующий код в метод `Update()` скрипта **MovementScript**:

```
float mX = Input.GetAxis("Mouse X") / 10;
float mY = Input.GetAxis("Mouse Y") / 10;
transform.Translate(mX, mY, 0);
```

6. Добавьте следующий код в скрипт **TriggerScript**:

```
void OnTriggerEnter (Collider other)
{
    print(other.gameObject.name + " has entered the cube");
}
void OnTriggerStay (Collider other)
{
    print(other.gameObject.name + " is still in the cube");
}
void OnTriggerExit (Collider other)
{
    print(other.gameObject.name + " has left the cube");
}
```

7. Обязательно поместите этот код не в методы, а внутри класса, то есть на том же уровне, что и методы `Start()` и `Update()`.
8. Запустите сцену и убедитесь, что сфера перемещается вместе с движением мыши. Столкните сферу с кубом и просмотрите сообщение на консоли. Обратите внимание, что объекты физически не реагируют друг на друга, тем не менее взаимодействуют. Изучите таблицу на рис. 9.4 и догадайтесь, о каком взаимодействии идет речь?

	Статичный коллайдер	Твердое тело	Кинематическое твердое тело	Статичный триггер-коллайдер	Твердотельный триггер-коллайдер	Кинематический твердотельный триггер-коллайдер
Статичный коллайдер		столкновение			триггер	триггер
Твердое тело	столкновение	столкновение	столкновение	триггер	триггер	триггер
Кинематическое твердое тело		столкновение		триггер	триггер	триггер
Статичный триггер-коллайдер		триггер	триггер		триггер	триггер
Твердотельный триггер-коллайдер	триггер	триггер	триггер	триггер	триггер	триггер
Кинематический твердотельный триггер-коллайдер	триггер	триггер	триггер	триггер	триггер	триггер

Рис. 9.4. Матрица взаимодействия коллайдеров

ПРИМЕЧАНИЕ

Если столкновения не происходит

Не все скрипты столкновения способны на самом деле создавать столкновение. На рис. 9.4 приведена матрица, в которой указано, какое взаимодействие возникает между двумя объектами: столкновение, срабатывание триггера или вообще никакого.

Статичные коллайдеры — это коллайдеры на любом игровом объекте без компонента **Rigidbody**. Они становятся твердыми коллайдерами при добавлении компонента **Rigidbody** или кинематическими твердыми коллайдерами, если вы дополнительно установите флажок **Is Kinematic**. Для каждого из этих трех типов коллайдеров вы можете также установить флажок **Is Trigger**. Именно так мы получаем шесть типов коллайдеров, показанных на рис. 9.4.

Рейкастинг

Рейкастинг — это выстрел воображаемым лучом с целью определения того, во что он попадет. Представьте, например, что вы смотрите в телескоп. Ваша область видимости — это луч, и все, что вы можете разглядеть на другом конце — это то, во что луч попадает. Разработчики игр постоянно используют рейкастинг для реализации механик прицеливания, определения прямой видимости, оценки расстояния и многого другого. В редакторе Unity существует несколько методов рейкастинга. Два наиболее часто используемых приведены далее. Первый метод рейкастинга выглядит следующим образом:

```
bool Raycast(Vector3 origin, Vector3 direction, float distance, LayerMask
mask);
```

Обратите внимание, что у этого метода несколько параметров и для их определения используется переменная `Vector3` (с ней вы уже знакомы). `Vector3` — это тип переменной, хранящий три числа с плавающей запятой. Он отлично подходит для определения координат *x*, *y*, *z* без использования трех отдельных переменных. Первый параметр, `origin`, — это начальная точка луча. Второй, `direction`, — это направление движения луча. Третий, `distance`, определяет, как далеко луч будет распространяться, и наконец, параметр `mask` определяет, какие слои задеваются лучом. Вы можете не указывать параметры `distance` и `mask`. В этом случае луч будет путешествовать бесконечно далеко и проходить по всем слоям объектов.

Как уже упоминалось ранее, с помощью лучей вы можете решать много задач. Если хотите определить, находится ли что-нибудь перед камерой, можно использовать скрипт со следующим кодом:

```
void Update()
{
    // Выстрел лучом из положения камеры вперед
```

```
    if (Physics.Raycast(transform.position, transform.forward, 10))
        print("There is something in front of the camera!");
}
```

Также вы можете задействовать этот метод, чтобы определить объект, с которым столкнется луч. В таком случае вам понадобится специальный тип переменной под названием `RaycastHit`. Во многих версиях рейкастинга используются (или не используются) параметры `distance` и `layer mask`. Самый простой способ применения этой версии рейкастинга выглядит примерно так:

```
bool Raycast(Vector3 origin, Vector3 direction, out Raycast hit, float
distance);
```

Вы, возможно, заметили один интересный момент: здесь используется новое ключевое слово, которого вы не видели раньше: `out`. Оно означает, что, когда метод выполняется, переменная `hit` будет содержать любой задетый объект. Метод эффективно передает значение наружу, когда это будет сделано. Суть в том, что метод `Raycast()` возвращает логическую переменную, которая указывает, было ли столкновение с объектом. Так как метод не может возвращать две переменные, можно использовать `out`, чтобы получить больше информации.

ПРАКТИКУМ

Потренируемся с лучами

В этом упражнении мы создадим интерактивную «стрелялку». Программа будет выпускать луч из камеры и уничтожать любые объекты, с которыми он вступит в контакт. Выполните следующие действия.

1. Создайте новый проект или сцену и добавьте на сцену четыре сферы.
2. Поместите сферы в позиции с координатами (–1, 1, –5), (1, 1,5, –5), (–1, –2, 5) и (1,5, 0, 0).
3. Создайте новый скрипт под названием **RaycastScript** и прикрепите его к объекту **Main Camera**. Внутри метода `Update()` для данного скрипта добавьте следующий код:

```
float dirX = Input.GetAxis ("Mouse X");
float dirY = Input.GetAxis ("Mouse Y");
// оси переставлены, так как вращение вокруг осей
transform.Rotate (dirY, dirX, 0);
CheckForRaycastHit (); // описание данного метода добавим далее
```

4. Добавьте в скрипт метод `CheckForRaycastHit()`. Код должен находиться вне метода, но внутри класса:

```
void CheckForRaycastHit() {
    RaycastHit hit;
    if (Physics.Raycast (transform.position, transform.forward, out
            hit)) {
```

```
            print (hit.collider.gameObject.name + " destroyed!");
            Destroy (hit.collider.gameObject);
        }
    }
```

5. Запустите сцену. Убедитесь, что перемещение мыши приводит к повороту камеры. Попробуйте нацелить центр камеры на каждую сферу. Обратите внимание, что сферы уничтожаются, и сообщение об этом выводится в консоль.

Резюме

В этом часе вы узнали о взаимодействии объектов через столкновения. Сначала мы изучили основы физики и работы с твердыми телами на движке Unity. Затем мы разобрались с различными типами коллайдеров и столкновений. После этого вы узнали, что столкновение — это не только врезание предметов друг в друга, и поработали с триггерами. Наконец, вы научились находить объекты, используя рейкастинг.

Вопросы и ответы

Вопрос: **Все мои объекты должны иметь компонент Rigidbody?**

Ответ: Это полезный компонент, который в основном используется для физического взаимодействия. Тем не менее его добавление к каждому объекту может привести к странным последствиям и снижению производительности. Добавлять компоненты следует только тогда, когда в этом появляется необходимость, а действовать превентивно не нужно.

Вопрос: **Есть несколько коллайдеров, которые мы не обсудили. Почему?**

Ответ: Большинство коллайдеров либо ведут себя так же, как те, которые мы уже обсудили, либо здесь не применяются и опущены. Достаточно сказать, что в этой книге есть все, что вам нужно знать, чтобы создавать интересные игры.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Какой компонент необходимо применить к объекту, если вы хотите, чтобы он проявлял физические свойства, например падал?
2. Верно или нет: объект может иметь только один компонент **Collider**.
3. Для чего нужен рейкастинг?

Ответы

1. Компонент **Rigidbody**.
2. Нет. У объекта может быть множество разных коллайдеров.
3. Например, чтобы определять, на какой объект смотрит камера, или для измерения расстояния между объектами.

Упражнение

Сейчас мы создадим интерактивное приложение, в котором используется движение и триггеры. Упражнение требует, чтобы вы творчески подошли к поиску решения (потому что здесь оно не представлено). Если вы оказались в тупике и нуждаетесь в помощи, вы можете обратиться к файлам примеров для часа 9.

1. Создайте новый проект или сцену. Добавьте на сцену куб и поместите его в позицию с координатами (–1,5, 0, –5). Задайте кубу масштаб (0,1, 2, 2) и присвойте ему имя **LTrigger**.
2. Продублируйте куб (щелкнув правой кнопкой мыши по кубу на панели **Hierarchy** и выбрав команду **Duplicate**). Назовите новый куб **RTrigger** и поместите его в позицию с координатами (1,5, 0, –5).
3. Добавьте на сцену сферу и поместите ее в позицию с координатами (0, 0, –5). Добавьте в сферу компонент **Rigidbody** и сбросьте флажок **Use Gravity**.
4. Создайте скрипт под названием **TriggerScript** и прикрепите его к объектам **LTrigger** и **RTrigger**. Создайте скрипт под названием **MotionScript** и прикрепите его к сфере.

5. Теперь начинается самое интересное. Вам нужно создать в игре следующие функциональные возможности.
    - Игрок должен иметь возможность перемещать сферу с помощью клавиш со стрелками.
    - Когда сфера входит в один из триггеров, покидает его или остается в нем, в консоль должно выводиться соответствующее сообщение: имя триггера (LTrigger или RTrigger) и тип взаимодействия (вход, выход, нахождение внутри).
    - В этом упражнении есть небольшой подвох, который вам придется преодолеть, чтобы справиться с задачей.

Удачи!

10-Й ЧАС
Игра вторая: «Шар хаоса»

Что вы узнаете в этом часе

- Как создать игру «Шар хаоса»
- Как построить арену для игры «Шар хаоса»
- Как создать объекты для игры «Шар хаоса»
- Как создать управляющие объекты для игры «Шар хаоса»
- Как модернизировать игру «Шар хаоса»

Пришло время еще раз воспользоваться всеми накопленными знаниями и умениями. В этом часе мы создадим игру «Шар хаоса», которая представляет собой быструю игру в аркадном стиле. Мы начнем с изучения основных элементов игры. Затем мы создадим арену и игровые объекты. Каждый тип объекта будет уникальным, со своими особыми свойствами столкновения. Затем мы добавим в игру интерактивность. В конце этого часа мы поиграем в нашу игру и внесем мелкие поправки для улучшения ее работы.

СОВЕТ

Завершенный проект

Обязательно изучите материалы этого часа, чтобы создать полноценный игровой проект. Если у вас возникнут затруднения, найдите готовый вариант игры в файлах примеров для часа 10. Проанализируйте его, если вам нужна помощь или вдохновение!

Проектирование

В часе 6, создавая игру «Великолепный гонщик», вы многое узнали о составляющих игрового проекта. На этот раз мы погрузимся в них сами.

Концепция

Эта игра в чем-то похожа на Pinball или Breakout. Игрок находится на арене. Каждый из четырех углов окрашен в свой цвет, а по самой арене плавают четыре шара тех же цветов. Среди них — несколько желтых, называемых шарами хаоса. Их цель заключается в том, чтобы попадаться игроку под ноги и усложнить игру. Они меньше, чем четыре разноцветных, но двигаются быстрее. Игра происходит на плоской поверхности, где игроки будут пытаться вытолкать цветные шары в правильные углы.

Правила

Правила определяют, как играть, а также намекают на некоторые свойства объектов. Правила игры «Шар хаоса» будут следующими.

- Игрок выигрывает, когда все четыре шара находятся в правильных углах арены. Условия поражения нет.
- Попадание шара в правильный угол приводит к исчезновению шара и отключению освещения.
- Все объекты в игре абсолютно упругие и не теряют энергии при ударе.
- Ни шары, ни игрок не могут покинуть арену.

Требования

Требования здесь крайне просты. Игра не является графически сложной, а в своей ценности для развлечения больше опирается на скрипты и взаимодействия. Требования, предъявляемые к игре «Шар хаоса», будут следующими.

- Окруженная стеной арена, на которой происходит игра.
- Текстуры для арены и игровых объектов. Все они есть в стандартных ассетах Unity.
- Несколько цветных шаров и шары хаоса. Они будут созданы в Unity.
- Контроллер персонажа. Его возьмем в стандартных ассетах Unity.
- Игровой менеджер. Он будет создан в Unity.
- Физический материал для упругого тела. Он будет создан в Unity.
- Цветные индикаторы для углов. Они будут созданы в Unity.
- Интерактивные скрипты. Они разработаны в среде Visual Studio.

Арена

Первое, что вам понадобится создать, — это область, где будет происходить действие. Термин «арена» здесь намекает на идею, что уровень будет небольшого размера и замкнутый. Ни игрок, ни какие-либо шары не должны иметь возможности покинуть арену. Она будет выглядеть довольно аскетично, как показано на рис. 10.1.

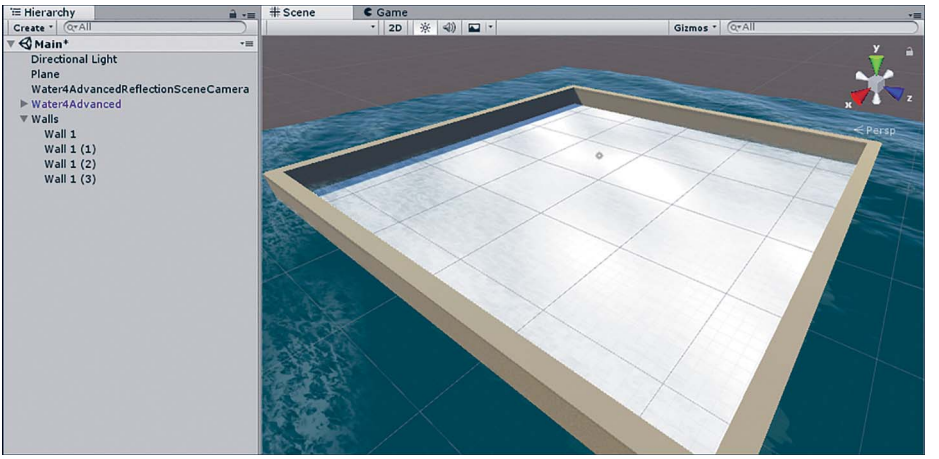


Рис. 10.1. Арена

Создание арены

Как уже упоминалось ранее, создать арену легко, так как она весьма проста. Выполните следующие действия.

1. Создайте новый проект под названием «Шар хаоса».
2. Выберите команду **Assets** ⇒ **Import Package** и импортируйте пакеты **Environment** и **Characters**.
3. Добавьте на сцену плоскость (выбрав команду **GameObject** ⇒ **3D** ⇒ **Plane**). Поместите плоскость в позицию с координатами (0, 0, 0) и задайте ей масштаб (5, 1, 5).
4. Удалите компонент **Main Camera**.
5. Добавьте на сцену куб. Поместите его в позицию с координатами (–25, 1,5, 0) и задайте ему масштаб (1,5, 3, 51). Теперь ваш куб превратился в боковую стенку для арены. Присвойте ему имя **Wall 1**.
6. Сохраните сцену под названием **Main** в папку **Scene**s.

СОВЕТ

Дублирование объектов

Вам наверняка интересно, почему мы создали только одну стену, когда для арены, очевидно, нужно четыре. Идея заключается в том, чтобы свести к минимуму излишнюю и утомительную работу, насколько возможно. Часто, если вам требуется несколько похожих объектов, вы можете создать один, а затем продублировать его. В нашем случае мы создаем одну стену с определенными материалами и свойствами, а затем сделаем три копии. Тот же процесс повторим для угловых зон, шаров хаоса и цветных шаров. Надеюсь, на этом моменте вам видно, что небольшое планирование поможет вам сэкономить неплохое количество времени. Стоит также отметить, что весь этот процесс можно еще упростить, если использовать префабы. Но поскольку кое-кто (не будем показывать пальцем) до сих пор не рассказал вам о префабах, а сделает это только в следующем часе, сейчас вы можете воспользоваться дублированием.

Текстурирование

Пока наша арена выглядит довольно скучной и пресной. Она вся белая, а из объектов — одна-единственная стена. На следующем шаге мы добавим текстуры, чтобы оживить сцену. Нам нужно текстурировать два объекта: стены и пол. Пробуйте экспериментировать с ними. Вы можете сделать нечто более интересное, как вам захочется, но для начала выполните следующие действия.

1. Создайте новую папку с названием **Materials** в каталоге **Assets** на панели **Project**. Добавьте в папку материал (для этого щелкните по ней правой кнопкой мыши и выберите команду **Create** ⇒ **Material**). Назовите его именем **Wall**.
2. Примените текстуру **Sand Albedo** к материалу **Wall** на панели **Inspector**. Вы можете сделать это путем перетаскивания текстуры на свойство **Albedo** или щелкнув мышью по кружку рядом со словом **Albedo** на панели **Inspector** (рис. 10.2).
3. Задайте свойству **Smoothness** значение **0**.
4. Задайте свойству **Tiling** для оси *x* значение **10**.
5. Нажав и удерживая кнопку мыши, перетащите материал **Wall** на стену в окне панели **Scene**.

Теперь нам нужно поработать над полом. В Unity есть крутые водные шейдеры, и вы можете снова использовать их здесь.

1. На панели **Project** откройте папку **Standard Assets\Environment\Water\Water4\Prefabs**. Перетащите ассет **Water4Advanced** на сцену.
2. Поместите воду в центре, в позиции с координатами (0, 0,5, 0).

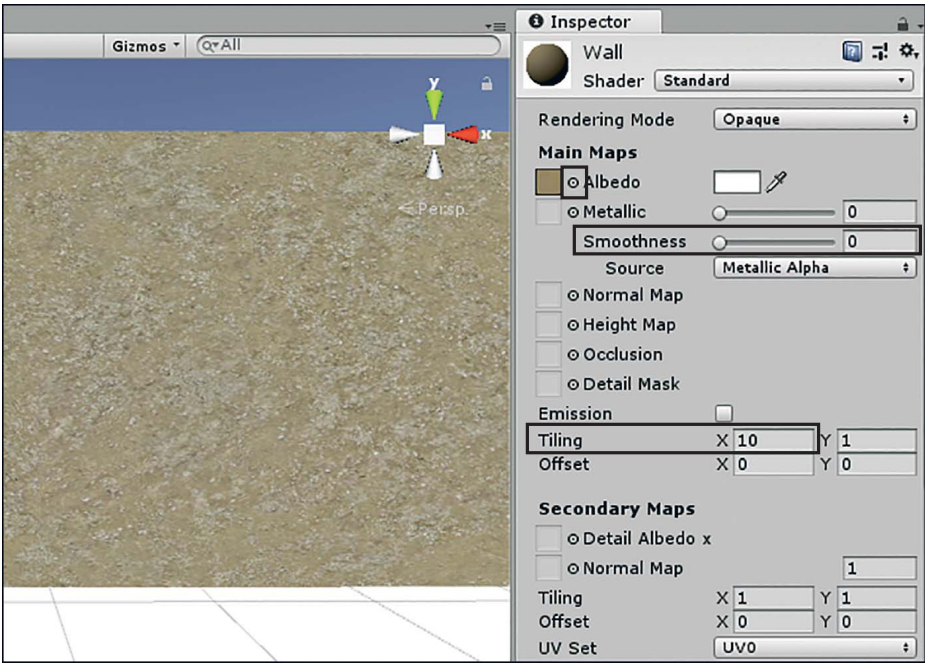


Рис. 10.2. Материал стены

Создание сверхупругого материала

Чтобы объекты отскакивали от стен без потери энергии, нам понадобится сверхупругий материал. Если вы помните, в Unity имеется набор готовых физических материалов. Однако встроенный упругий материал все же недостаточно упругий для нашей задачи. Таким образом, нам придется создать новый материал следующим образом.

1. Щелкните правой кнопкой мыши по папке **Materials** и выберите команду **Create** ⇒ **Physic Material**. Назовите материал **SuperBouncy**.

2. Установите свойства для сверхупругого материала, как показано на рис. 10.3. Идея в том, чтобы сделать шары абсолютно упругими, чтобы они не теряли скорость после столкновения.

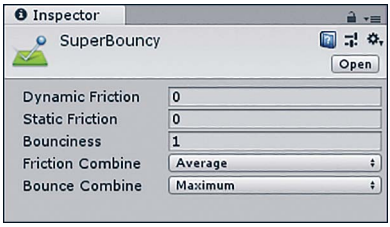


Рис. 10.3. Настройки материала **SuperBouncy**

Теперь вы можете установить физический материал **SuperBouncy** непосредственно на коллайдер вашей стены. Однако проблема в том, что вам нужно будет добавить этот материал на все стены, все шары и на игрока. В принципе в этой игре материал **SuperBouncy** нужен всем сталкивающимся объектам. Таким образом, вы можете применить его в качестве материала по умолчанию для всех коллайдеров с помощью панели **Physics Manager**. Выполните следующие действия.

1. Выберите команду меню **Edit** ⇒ **Project Settings** ⇒ **Physics**. На панели **Inspector** появится раздел **Physics Manager** (рис. 10.4).
2. Выберите материал **SuperBouncy** в раскрывающемся списке **Default Material** (Материал по умолчанию).

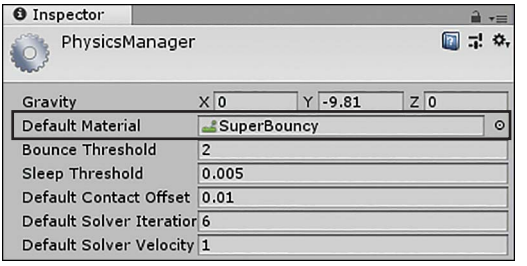


Рис. 10.4. Панель **Physics Manager**

На этой панели вы также можете изменить базовые параметры физики (столкновение, сила тяжести и так далее). Но пока воздержитесь от игры в Брюса Всемогущего и оставьте физику такой, какая она есть.

Финальные штрихи арены

Теперь, когда стены и земля готовы, вы можете закончить арену. Мы проделали серьезную работу, и теперь нам нужно лишь продублировать стены (для этого щелкните правой кнопкой мыши на панели **Hierarchy** и выберите команду **Duplicate**). Выполните следующие действия.

1. Однократно продублируйте стену. Поместите новый экземпляр в позицию с координатами (25, 1,5, 0).
2. Продублируйте стену еще раз. Поместите новый экземпляр в позицию с координатами (0, 1,5, 25) и задайте ему вращение (0, 90, 0).
3. Продублируйте стену, созданную на шаге 2 (повернутую) и поместите ее в позицию с координатами (0, 1,5, −25).
4. Создайте пустой игровой объект под названием **Walls**. Установите положение нового объекта в позицию с координатами (0, 0, 0). Сгруппируйте четыре стены внутрь этого нового объекта.

Теперь у вас на арене должно быть четыре стены без каких-либо зазоров или швов (см. рис. 10.1).

Игровые сущности

В этом разделе мы создадим различные игровые объекты, необходимые нам. Так же, как и в случае со стенами арены, проще создать один экземпляр каждого объекта, а затем продублировать его.

Игрок

Игрок будет представлен контроллером персонажа **First Person**, но с некоторыми изменениями. Во время создания этого проекта нужно импортировать пакет контроллера персонажа. Нам также понадобится поднять камеру контроллера таким образом, чтобы у игрока был хороший обзор. Выполните следующие действия.

1. Перетащите на сцену контроллер **FPSController** из папки **Assets\Characters\FirstPersonCharacter\Prefabs**.
2. Поместите контроллер в позицию с координатами (0, 1, 0).
3. Разверните объект **FPSController** на панели **Hierarchy** и найдите дочерний объект **FirstPersonCharacter** (на котором установлена камера).
4. Поместите объект **FirstPersonCharacter** в позицию с координатами (0, 5, −3,5) и задайте ему вращение (43, 0, 0). Теперь камера должна расположиться чуть выше и позади контроллера и смотреть немного вниз.

Следующее, что нужно сделать: добавить контроллер бампера. Это плоская поверхность, от которой игрок будет отталкивать шары. Выполните следующие действия.

1. Добавьте на сцену куб и переименуйте его в **Bumber**. Задайте ему масштаб (3,5, 3, 1).
2. На панели **Hierarchy** перетащите бампер на объект **FPSController**, чтобы он стал дочерним.
3. Задайте бамперу положение (0, 0, 1) и вращение (0, 0, 0). Теперь он находится немного впереди контроллера.
4. Задайте бамперу цвет, создав новый материал (не физический) под названием **BumperColor**. Установите цвет альбедо на свое усмотрение и перетащите материал на бампер.

Теперь нужно настроить объект **FPSController**, чтобы он стал более подходящим для этой игры. Установите все настройки, как показано на рис. 10.5. Параметры, которые отличаются от значений по умолчанию, выделены полужирным шрифтом.

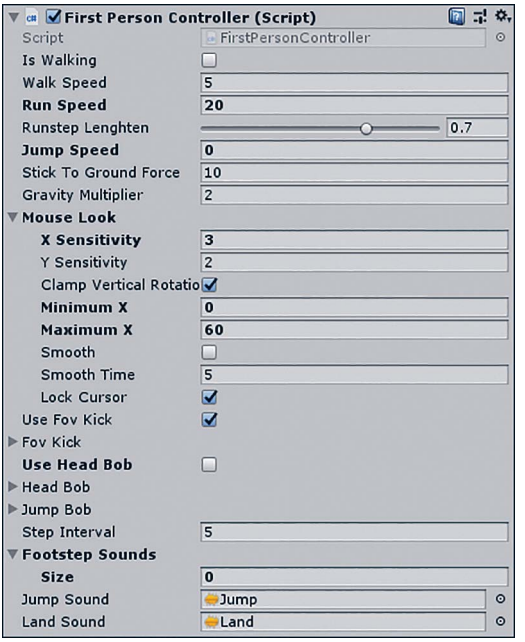


Рис. 10.5. Параметры скрипта **FPSController**

Шары хаоса

Роль шаров хаоса будут выполнять быстрые и агрессивные шары, летающие по арене и мешающие игроку. Во многом они похожи на цветные шары, так что вы будете работать с универсально применимыми ассетами. Чтобы создать первый шар хаоса, выполните следующие действия.

1. Добавьте на сцену сферу. Назовите ее **Chaos**, поместите в позицию с координатами (12, 2, 12) и задайте масштаб (0,5, 0,5, 0,5).

2. Создайте новый материал (не физический) для шара хаоса под названием **ChaosBall** и установите для него ярко-желтый цвет. Перетащите материал на сферу.

3. Добавьте в сферу компонент **Rigidbody**. Как показано на рис. 10.6, сбросьте флажок **Use Gravity**. Присвойте свойству **Collision Detection** значение **Continuous Dynamic**. В группе свойств **Constraints** заморозьте положение по оси *y*, потому что нам не требуется, чтобы шары могли перемещаться вверх или вниз.

4. Откройте инструмент **Tag Manager** (выбрав команду меню **Edit** ⇒ **Project Settings** ⇒ **Tags & Layers**). Откройте раскрывающийся список **Tags**

и выберите тег **Chaos**. Также добавьте теги **Green**, **Orange**, **Red** и **Blue**, они нам потребуются позже.

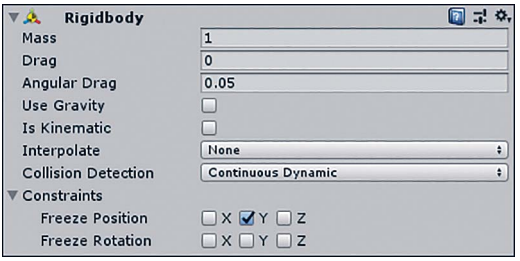


Рис. 10.6. Компонент **Rigidbody** для шара хаоса

5. Выберите сферу **Chaos** и задайте ей тег **Chaos** на панели **Inspector** (рис. 10.7).

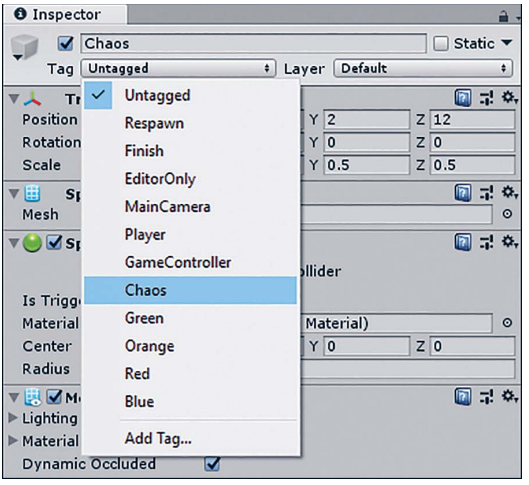


Рис. 10.7. Выбор тега **Chaos**

Шар готов, хотя пока он ничего не делает. Вам нужно написать скрипт, согласно которому шар будет перемещаться по арене. Создайте скрипт **VelocityScript** и прикрепите его к шару **Chaos**. Переместите скрипт в папку **Scripts**. В листинге 10.1 показан полный код скрипта **VelocityScript**.

ЛИСТИНГ 10.1. Скрипт *VelocityScript.cs*

```
using UnityEngine;
public class VelocityScript: MonoBehaviour
{
    public float startSpeed = 50f;
```

```
    void Start ()
    {
        Rigidbody rigidBody = GetComponent<Rigidbody> ();
        rigidBody.velocity = new Vector3 (startSpeed, 0, startSpeed);
    }
}
```

Запустите сцену, и вы увидите, как шар начнет летать по арене. С шарами хаоса на этом все. На панели **Hierarchy** продублируйте шар хаоса четыре раза. Разбросайте шары по арене (не меняя положения по высоте) и поверните их в случайных направлениях. Помните, что движение вдоль оси *y* должно быть запрещено, поэтому убедитесь, что каждый шар имеет положение вдоль оси *y* равное **2**. Наконец, создайте пустой объект **GameObject** под названием **Chaos Balls**, расположите его в позиции с координатами (0, 0, 0) и сделайте шары хаоса дочерними для него.

Цветные шары

Хотя шары хаоса, в общем-то, тоже цветные (желтые), они не относятся к цветным шарам в смысле этой игры. Цветные шары — это четыре специальных шара, которые нужны, чтобы выиграть. Они должны быть окрашены в красный, оранжевый, синий и зеленый цвета. Как и с шарами хаоса, мы сделаем один, а затем продублируем его, чтобы ускорить создание цветных шаров.

Чтобы создать первый шар, выполните следующие действия.

1. Добавьте на сцену сферу и назовите ее **Blue Ball**. Поместите шар где-то ближе к середине арены и убедитесь, что ее положение по оси *y* равно **2**.

2. Создайте новый материал под названием **BlueBall** и задайте ему синий цвет так же, как вы окрасили шар хаоса в желтый. Заодно создайте материалы **Redball**, **GreenBall** и **OrangeBall** и задайте каждому из них соответствующий цвет: красный, зеленый и оранжевый. Перетащите материал **BlueBall** на сферу.

3. Добавьте в сферу компонент **Rigidbody**. Сбросьте флажок **Use Gravity**, присвойте свойству **Collision Detection** значение **Continuous Dynamic** и заморозьте положение по оси *y*.

4. Ранее вы создали тег **Blue**. Теперь измените тег сферы так же, как делали это для шара хаоса (см. рис. 10.7).

5. Прикрепите к этой сфере скрипт **VelocityScript**. На панели **Inspector** найдите компонент **Velocity Script (Script)** и присвойте свойству **Start Speed** значение **25** (рис. 10.8). В результате сфера будет двигаться медленнее, чем шары хаоса.

Рис. 10.8. Изменение свойства **Start Speed**

Запустив сцену сейчас, вы увидите, как синий шар быстро движется по арене.

Теперь нужно создать остальные три шара. Каждый из них будет дубликатом синего. Выполните следующие действия.

1. Трижды продублируйте объект **Blue Ball**. Задайте новым шарам имена **Red Ball**, **Orange Ball**, **Green Ball**.
2. Присвойте новым шарам теги, соответствующие имени. Важно, чтобы имя и тег совпадали.
3. Перетащите на новые шары соответствующие материалы. Важно, чтобы реальный цвет и цвет в имени шара совпадали.
4. Расположите шары случайным образом, но на высоте по оси *y*, равной **2**.

На данный момент с активными игровыми объектами закончено. Если запустить сцену, вы увидите, как все шары будут летать по арене.

Управление

Теперь, когда все нужные объекты размещены, пришло время игрофикации. То есть настало время вдохнуть в игру жизнь. Чтобы сделать это, необходимо создать четыре угловые цели, скрипты и игровой контроллер. А потом мы сможем уже и поиграть.

Цели

В каждом из четырех углов должна быть особая цель, соответствующая шару по цвету. Идея заключается в том, что при попадании шара в цель должен проверяться тег. Если тег соответствует цвету цели, то есть совпадение. При этом шар уничтожается, а цель считается выполненной. Как вы делали с другими объектами до этого, вы можете настроить одну цель, а затем продублировать ее.

Чтобы настроить первую цель, выполните следующие действия.

1. Создайте пустой игровой объект (выбрав команду **GameObject** ⇒ **Create Empty**). Присвойте этому объекту имя **Blue Goal** и тег **Blue**. Поместите объект в позицию с координатами (−22, 2, −22).
2. Прикрепите к объекту коллайдер **Box Collider** и установите флажок **Is Trigger**. Задайте размер куба коллайдера (3, 2, 3).

3. Прикрепите к цели источник света (выбрав команду **Component** ⇒ **Rendering** ⇒ **Light**). Установите точечный источник света и задайте ему такой же цвет, как и у цели (рис. 10.9). Задайте интенсивность света (свойство **Intensity**), равную **3**, а параметру **Indirect Multiplier** присвойте значение **0**.

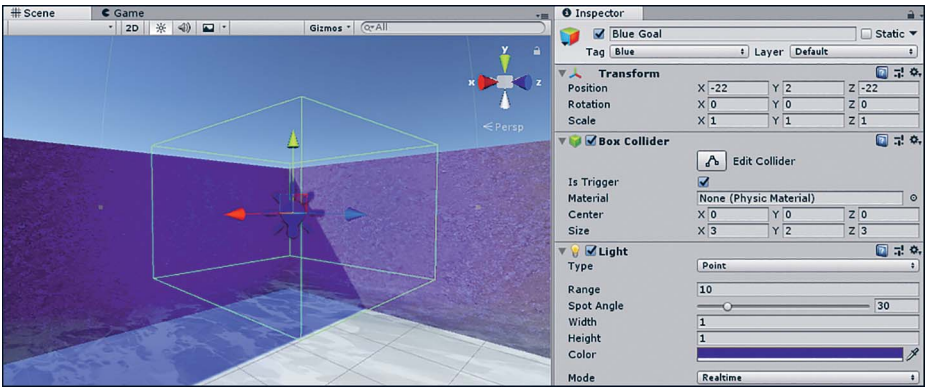


Рис. 10.9. Синяя цель

Далее вам нужно создать скрипт **GoalScript** и прикрепить его к синей цели. В листинге 10.2 приведено содержимое скрипта.

ЛИСТИНГ 10.2. Скрипт *GoalScript.cs*

```
using UnityEngine;
public class GoalScript: MonoBehaviour
{
    public bool isSolved = false;
    void OnTriggerEnter (Collider collider)
    {
        GameObject collidedWith = collider.gameObject;
        if (collidedWith.tag == gameObject.tag)
        {
            isSolved = true;
            GetComponent<Light>().enabled = false;
            Destroy (collidedWith);
        }
    }
}
```

Как видно в скрипте, метод `OnTriggerEnter()` сравнивает свой тег с тегами контактирующих с ним объектов. Если они совпадут, то объект будет уничтожен, цель выполнена, а источник света в этом углу отключится.

Когда вы прикрепите скрипт к цели, можно будет продублировать ее. Чтобы создать другие цели, выполните следующие действия.

1. Трижды продублируйте объект **Blue Goal**. Назовите новые цели в соответствии с их цветами: **Red Goal**, **Green Goal**, **Orange Goal**.
2. Задайте целям соответствующие им теги.
3. Задайте источникам света у целей соответствующие цвета.
4. Расставьте цели на арене. Цвета могут находиться в любом углу, и у каждой цели должен быть собственный угол. Координаты углов: (22, 2, −22), (22, 2, 22) и (−22, 2, 22).
5. Соберите все цели в пустой игровой объект под названием **Goals**.

Теперь все цели установлены и работоспособны.

Менеджер игры

Последний элемент, необходимый нашей игре, — менеджер. Этот контроллер будет отвечать за проверку выполнения всех целей в каждом кадре и подаст сигнал, когда все четыре цели будут выполнены. Для этой конкретной игры его скрипт будет крайне прост. Чтобы создать менеджер игры, выполните следующие действия.

1. Добавьте на сцену пустой игровой объект. Переместите его куда-нибудь в сторону. Присвойте ему имя **Game Manager**.
2. Создайте скрипт **GameManager** и добавьте в него код из листинга 10.3. Прикрепите скрипт к объекту **Game Manager**.
3. Выделив объект **Game Manager**, перетащите каждую цель на соответствующее свойство на компоненте **Game Manager (Script)** (см. рис. 10.10).

ЛИСТИНГ 10.3. Скрипт Game Manager

```
using UnityEngine;
public class GameManager: MonoBehaviour
{
    public GoalScript blue, green, red, orange;
    private bool isGameOver = true;
    void Update ()
    {
        // Если все четыре цели выполнены — игра завершается
        isGameOver = blue.isSolved && green.isSolved && red.isSolved &&
        orange.isSolved;
    }
void OnGUI()
```

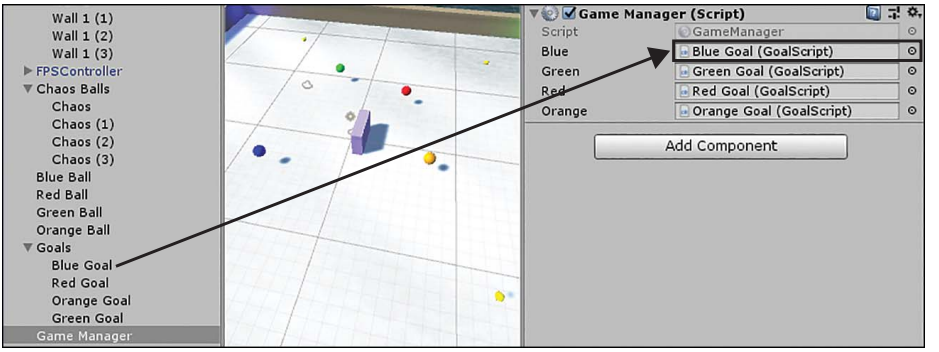


Рис. 10.10. Добавление целей в менеджер игры

Как видно в скрипте, показанном в листинге 10.3, менеджер игры имеет ссылку на каждый из четырех объектов-целей. В каждом кадре менеджер проверяет, все ли цели выполнены. Если да, менеджер присваивает переменной `isGameOver` значение `true` и выводит на экран сообщение Game Over.

Поздравляю! Игра «Шар хаоса» готова!

Улучшение игры

Хотя наша игра готова, она наверняка далека от идеала. Кое-какие особенности, которые могли бы значительно улучшить геймплей, были опущены. Мы сделали это для краткости, а также для того, чтобы дать вам возможность самим поэкспериментировать с игрой. В некотором смысле, можно сказать, что наша игра «Шар хаоса» дошла до стадии полноценного прототипа. В нее можно играть, но ей не хватает огранки. Мы предлагаем вам еще раз просмотреть этот час и найти способы сделать игру лучше. Поиграв в нее, продумайте следующие вопросы:

- Не является ли игра чересчур легкой или трудной?
- Как сделать игру легче или труднее?
- Как придать игре какую-нибудь удивительную особенность?
- Что в этой игре весело? А что скучно?

Упражнение в конце этого часа дает вам возможность улучшить игру и добавить некоторые функции. Обратите внимание, что если у вас выводится какая-нибудь ошибка, это означает, что вы пропустили шаг. Обязательно вернитесь к соответствующему практикуму и перепроверьте все шаги, чтобы устранить все возможные ошибки.

Резюме

В этом часе мы создали игру «Шар хаоса». Мы начали с проектирования игры, определили концепцию, правила и требования. Затем мы создали арену и узнали, что иногда удобнее сделать один объект и продублировать его, чтобы сэкономить время. Мы создали игрока, шары хаоса, цветные шары, цели и игровой контроллер. В конце часа мы поиграли в игру и подумали, как можно ее улучшить.

Вопросы и ответы

Вопрос: **Зачем использовать значение Continuous Dynamic в свойстве обнаружения столкновения для шаров хаоса? Мне казалось, что этот параметр снижает производительность.**

Ответ: Этот параметр и правда может негативно влиять на производительность. Но в данном случае он необходим. Шары хаоса маленькие и достаточно быстрые, поэтому иногда они могут проходить сквозь стены.

Вопрос: **Зачем я создавал тег Chaos, если так и не использовал его?**

Ответ: Этот тег понадобится вам для улучшения игры в упражнении ниже.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Как игрок может проиграть в игре «Шар хаоса»?
2. Вдоль какой оси было ограничено перемещение всех шаров?
3. Верно или нет: цели в игре «Шар хаоса» используют метод `OnTriggerEnter()`, чтобы определить, является ли объект нужным шаром.
4. Зачем при создании игры «Шар хаоса» мы опустили некоторые функции?

Ответы

1. Это вопрос с подвохом. Игрок не может проиграть.
2. Вдоль оси *y*.
3. Верно.
4. Чтобы дать вам возможность самим добавить их.

Упражнение

Лучший аспект в создании игр состоит в том, что вы делаете их так, как хотите. Следование указаниям может дать вам хороший опыт, но в этом случае вы не получаете удовольствия от своего творчества. В данном упражнении попробуйте немного модифицировать игру «Шар хаоса», чтобы сделать ее уникальной. Что именно изменить — остается на ваше усмотрение. Ниже перечислены некоторые предложения.

1. Попробуйте добавить кнопку, которая позволит игроку снова начать игру, когда она закончена. (Мы еще не рассматривали элементы графического пользовательского интерфейса, но эта функция есть в игре «Великолепный Гонщик», которую вы создали в часе 6, так что вы сможете разобраться с этим).
2. Попробуйте добавить таймер, чтобы игрок знал, сколько потребовалось времени, чтобы выиграть.
3. Попробуйте разнообразить шары хаоса.
4. Попробуйте добавить цель, в которую вы должны затолкать все шары хаоса.
5. Поэкспериментируйте с размером или формой бампера игрока.
6. Попробуйте сделать сложный бампер из разных фигур.
7. Попробуйте закрыть воду за пределами арены ландшафтом, плоскостями или как-нибудь еще.

11-Й ЧАС
Префабы

Что вы узнаете в этом часе

- Основы работы с префабами
- Как работать с пользовательскими префабами
- Как обрабатывать префабы в коде

Префаб представляет собой сложный объект, который можно создавать снова и снова с минимальным количеством усилий. В этом часе вы узнаете о нем все, от сущности префаба и до его создания в редакторе Unity. Также вы узнаете о *наследовании*. Мы закончим урок, изучив, как можно добавить на сцену префаб с помощью редактора и с помощью кода.

Введение в префабы

Как уже говорилось ранее, префаб — это специальный ассет, объединяющий несколько игровых объектов. В отличие от обычных игровых объектов, которые существуют только в рамках одной сцены, префабы сохраняются как ассеты. Таким образом, их можно увидеть на панели **Project** и повторно использовать на разных сценах. Вы можете, например, создать сложный объект, скажем, «противника», превратить его в префаб, а затем использовать этот префаб, чтобы создать армию. Кроме того, можно делать копии префабов с кодом. Это позволяет генерировать практически бесконечное число объектов во время выполнения. А лучшее, что в префаб можно поместить любой игровой объект или даже несколько. Ваши возможности поистине становятся безграничными!

ПРИМЕЧАНИЕ

Если сложно представить

Если вы не совсем понимаете важность префабов, представьте вот что: на предыдущем занятии вы создали игру «Шар хаоса». В процессе вы сделали один шар хаоса и продублировали его четыре раза. Что делать, если вы хотите изменить все шары

хаоса одновременно, независимо от того, где они находятся на вашей сцене или в проекте? Это может быть трудно, а порой и невозможно. Но с префабами это становится невероятно легко.

Допустим, у вас есть игра, в которой в качестве врагов выступают, например, орки. Опять же, вы можете настроить одного орка, а затем продублировать его много раз, но вдруг вам понадобится использовать этого орка снова в другом месте? Вам придется полностью воссоздать его в новой сцене. А если орк сохранен как префаб, он будет частью проекта и его можно будет повторно использовать снова в любом количестве сцен. Префабы крайне важны и удобны при разработке игр на движке Unity.

Терминология префабов

При работе с префабами вам понадобится специальная терминология. Если вы знакомы с понятиями объектно-ориентированного программирования, вы заметите некоторое сходство.

Префаб: префаб — это базовый объект. Он существует только на панели **Project**. Это что-то вроде чертежа.

Экземпляр: экземпляр представляет собой реализацию префаба на сцене. Если считать префаб, например, чертежом автомобиля, то экземпляр — это реальный автомобиль. Если объект на панели **Scene** называется префабом, то на самом деле это *экземпляр префаба*. Словосочетание «экземпляр префаба» является синонимом *объекта префаба* или даже *клона префаба*.

Создание экземпляра — это процесс создания экземпляра префаба.

Наследование: наследование у префабов не отличается от стандартного наследования в программировании. В этом случае под термином «наследование» понимается природа, которой объединены все экземпляры префаба между собой и с самим префабом. Позже мы поговорим об этом более подробно.

Структура префаба

Даже если вы этого не знали, вы уже работали с префабами. Контроллеры персонажей в Unity — это префабы. Чтобы создать экземпляр префаба на сцене, вам нужно всего лишь перетащить его на панель **Scene** или **Hierarchy** (рис. 11.1).

На панели **Hierarchy** всегда понятно, какие объекты являются экземплярами префабов, так как они отображаются синим цветом. Разница в цветах может быть незаметна, но вы можете определить, что объект является префабом, посмотрев на верхнюю часть панели **Inspector** (рис. 11.2).

Так же, как в случае с другими сложными объектами, у сложных экземпляров префабов есть стрелка, которая позволяет разворачивать их и редактировать объекты в составе префаба.

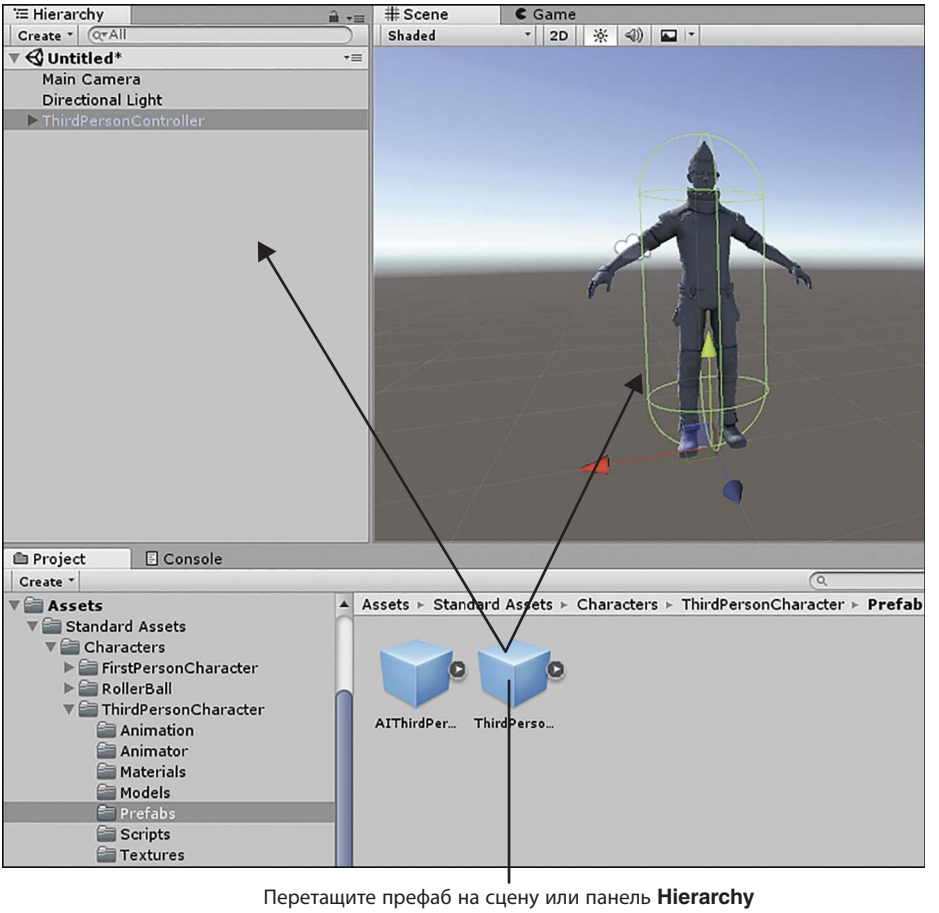


Рис. 11.1. Добавление экземпляра префаба на сцену

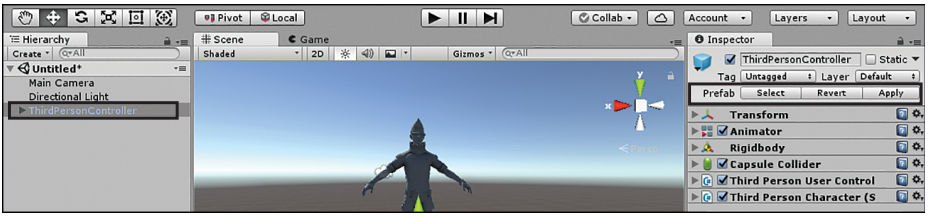


Рис. 11.2. Отображение экземпляров префаба на панели **Inspector**

Поскольку префаб — это ассет, который принадлежит проекту, а не конкретной сцене, вы можете отредактировать его на панели **Project** или **Scene**. При редактировании экземпляра префаба на панели **Scene** вы должны сохранить изменения в префабе, нажав кнопку **Apply** в правом верхнем углу панели **Inspector** (рис. 11.2).

Так же, как и игровые объекты, префабы могут быть комбинированными. Вы можете редактировать дочерние элементы префаба, щелкнув мышью по стрелке справа от префаба (рис. 11.3). При этом откроется список объектов для редактирования. Чтобы свернуть список, нажмите на стрелку еще раз.

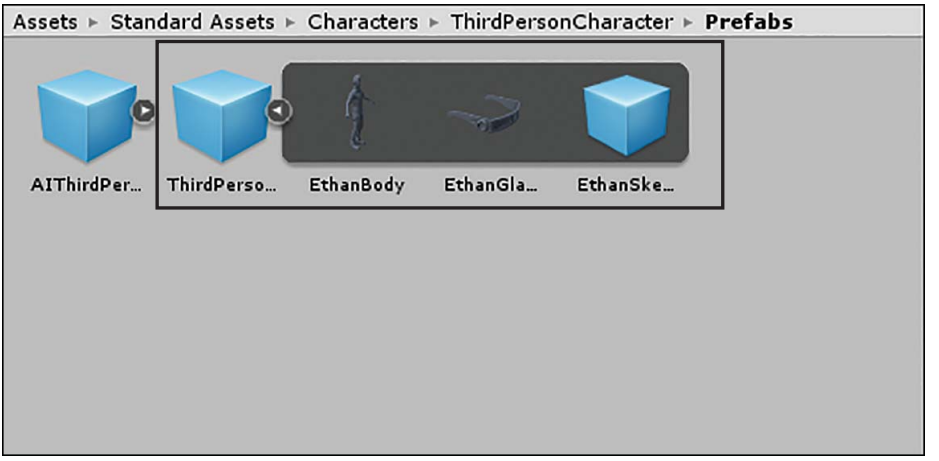


Рис. 11.3. Отображение содержимого префаба на панели **Project**

Управление префабами

Использование встроенных в Unity префабов — это хорошо, но часто возникает необходимость создать собственный. Процесс состоит из двух этапов. Сперва нужно создать ассет префаба, затем заполнить его содержимым.

Создать префаб очень легко. Для этого щелкните правой кнопкой мыши на панели **Project** и выберите команду **Create** ⇒ **Prefab** (рис. 11.4). Появится новый префаб, и вы можете присвоить ему имя на свое усмотрение. Поскольку префаб пока пуст, он выглядит как белая коробка.

Как и в случае со всеми другими ассетами, рекомендуется начать с создания для префабов подпапки в каталоге **Assets** на панели **Project**. Процедура не строго обязательна, но это очень важный организационный момент, который следует соблюсти, чтобы избежать путаницы между префабами и простыми ассетами (например, мешами или спрайтами).

Следующий шаг — заполнение префаба содержимым. В состав префаба может входить любой игровой объект. Вам нужно лишь один раз создать объект на панели **Scene**, а затем перетащить его в ассет префаба.

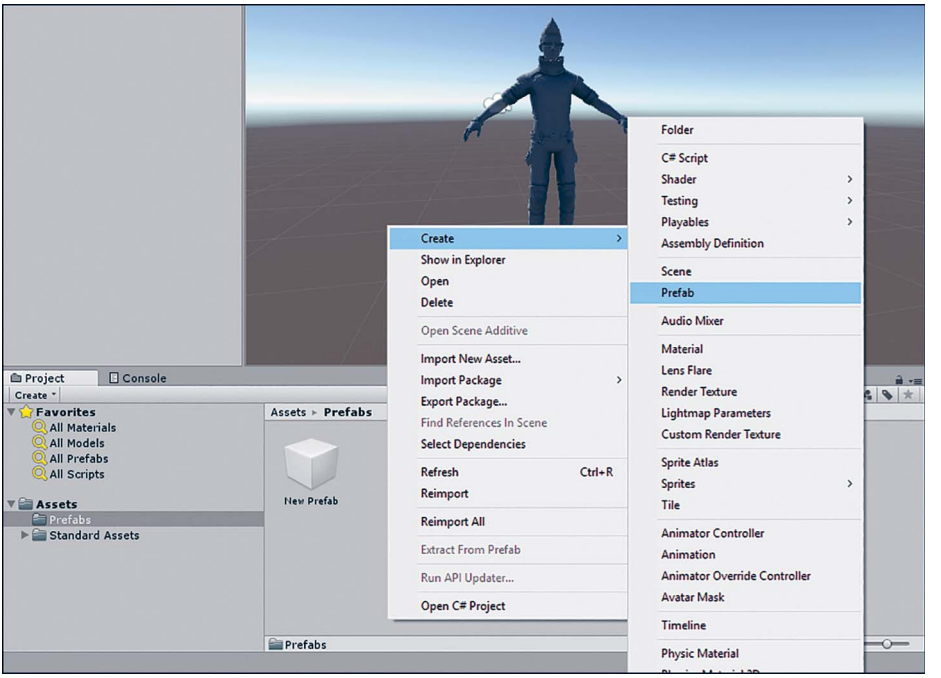
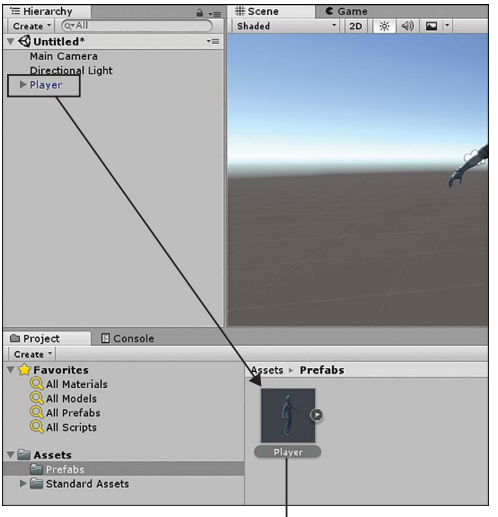


Рис. 11.4. Создание нового префаба

С другой стороны, вы можете сократить этот процесс до одного шага, перетащив любой объект из панели **Hierarchy** на панель **Project**. Это позволяет одновременно создать префаб, присвоить ему имя и заполнить содержимым (рис. 11.5).



Перетащите объект из панели **Hierarchy** на панель **Project**

Рис. 11.5. Более быстрый способ создания префабов

ПРАКТИКУМ

Создание префаба

Выполните предложенные шаги, чтобы создать префаб сложного игрового объекта, который вы будете использовать позже в этом часе (так что не удаляйте его!).

1. Создайте новый проект или сцену. Добавьте на сцену куб и сферу. Присвойте кубу имя **Lamp**.
2. Расположите куб в позиции с координатами (0, 1, 0) и масштабируйте (0,5, 2, 0,5). Добавьте в куб компонент **Rigidbody**. Поместите сферу в позицию с координатами (0, 2,2, 0). Добавьте в сферу компонент **Light**.
3. Перетащите сферу на панели **Hierarchy** на куб, чтобы разместить ее под кубом (рис. 11.6).

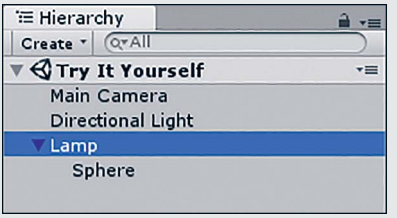


Рис. 11.6. Вложение сферы в куб

4. Создайте новую папку в каталоге **Assets** на панели **Project**. Присвойте ей имя **Prefabs**.
5. С панели **Hierarchy** перетащите игровой объект **Lamp** (который содержит сферу) на панель **Project** (рис. 11.7). Так вы создадите префаб **Lamp**. Обратите внимание, что куб и сфера на панели **Hierarchy** окрасились в синий цвет. Вы можете удалить куб и сферу со сцены, потому что теперь они объединены в префабе.

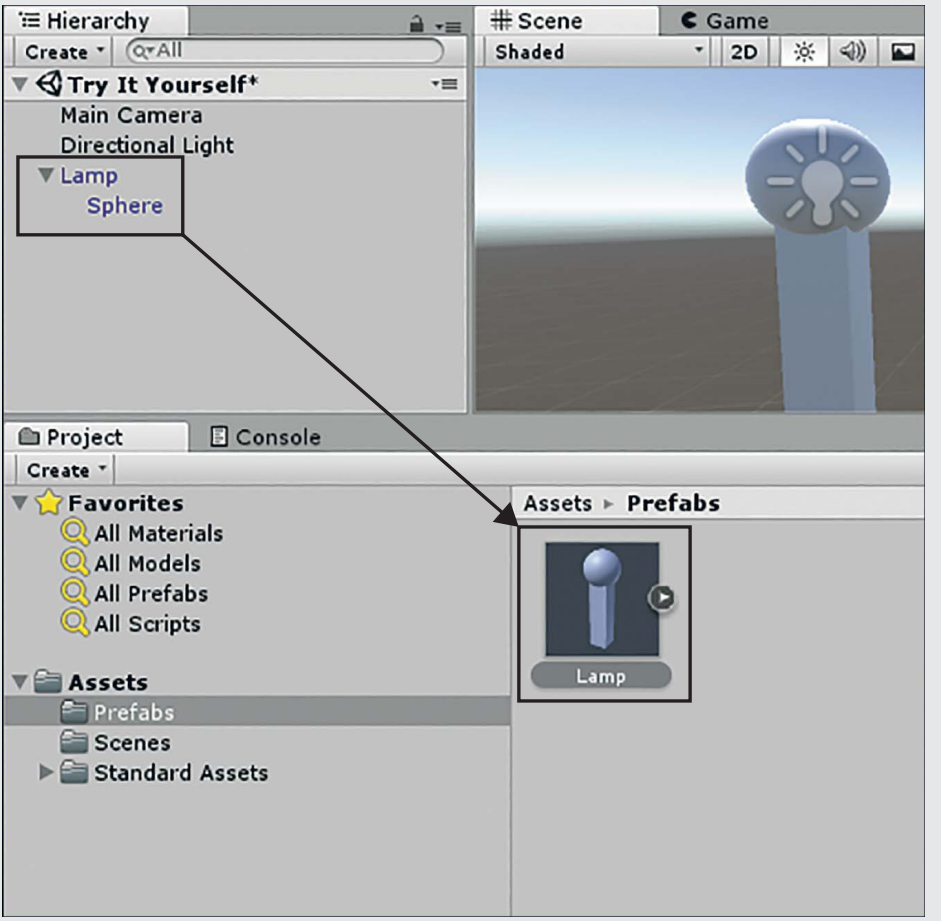


Рис. 11.7. Создание префаба из объекта

Добавление экземпляра префаба на сцену

После того как ассет префаба создан, вы сможете добавлять его сколько угодно раз в любой проект и на любую сцену. Чтобы добавить экземпляр префаба, нужно перетащить префаб с панели **Project** на сцену или панель **Hierarchy**.

На сцене префаб появляется там, где вы отпустите мышь. Если вы перетаскиваете его в пустую область панели **Hierarchy**, его первоначальное положение будет таким, как задано в префабе. Если перетащить префаб на другой объект на панели **Hierarchy**, он станет дочерним для этого объекта.

ПРАКТИКУМ

Создание нескольких экземпляров префаба

В практикуме «Создание префаба» мы сделали префаб **Lamp**. Теперь используем его, чтобы разместить на сцене несколько экземпляров префаба **Lamp**. Обязательно сохраните сцену, созданную здесь, потому что она потребуется нам позже в этом часе. Выполните следующие действия.

1. Создайте новую сцену в проекте из практикума «Создание префаба» и назовите ее **Lamps**.
2. Создайте плоскость и расположите ее в позиции с координатами (0, 0, 0). Назовите ее **Floor**. Можно задать полу серый материал, чтобы на нем лучше были видны тени.
3. Перетащите префаб **Lamp** на пол сцены. Обратите внимание, как экземпляр префаба **Lamp** взаимодействует с коллайдером пола.
4. Перетащите три экземпляра префаба **Lamp** на пол и расставьте их по углам (рис. 11.8).

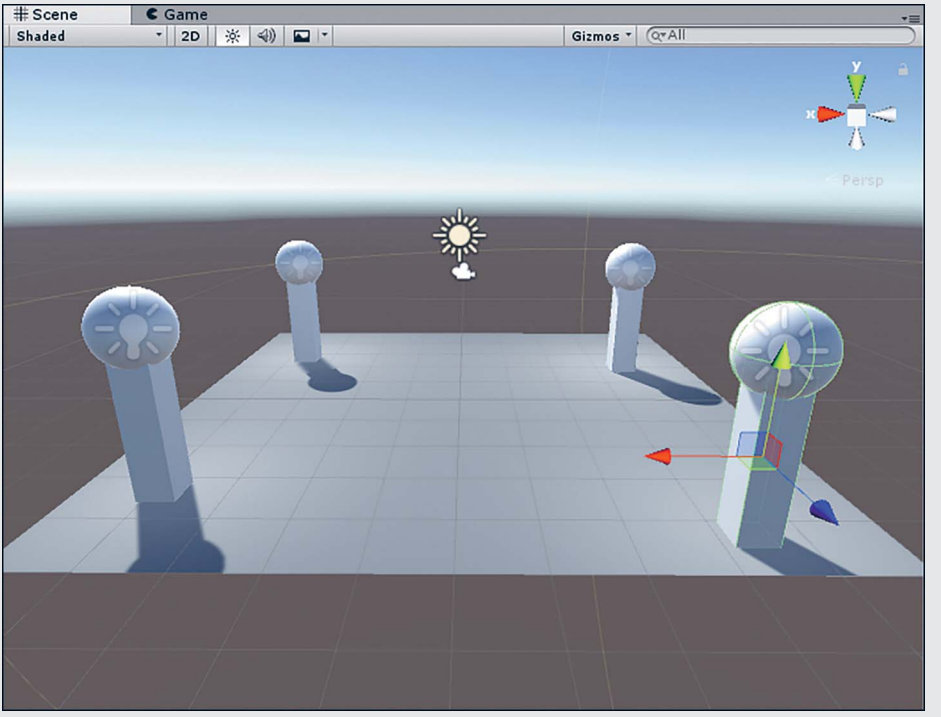


Рис. 11.8. Размещение экземпляров префаба **Lamp** на сцене

Наследование

Когда в отношении префабов используется термин *наследование*, имеется в виду связь, которой все экземпляры префаба объединены с фактическим ассетом префаба. То есть, если вы редактируете ассет префаба, все экземпляры префаба также автоматически изменяются. Это невероятно полезно. Вы столкнулись бы с распространенной проблемой многих программистов, которые помещают на сцену большое количество префабов, а потом понимают, что всем им нужны незначительные изменения. Без наследования пришлось бы редактировать каждый из них по отдельности.

Есть два способа отредактировать ассет префаба. Можно внести изменения на панели **Project**. Для этого выберите ассет префаба на панели **Project**, и вы увидите его компоненты и свойства на панели **Inspector**. Если вам необходимо переделать дочерний элемент, вы можете развернуть префаб (как описано ранее в этом часе) и изменить эти объекты таким же образом.

Также вы можете отредактировать префаб, перетащив его на сцену. Здесь можно вносить любые серьезные изменения, какие вам нужны. Когда вы закончите, нажмите кнопку **Apply** в правом верхнем углу панели **Inspector**.

ПРАКТИКУМ

Обновление префаба

Ранее в этом часе мы создали префаб и добавили несколько экземпляров префаба на сцену. Теперь можем отредактировать префаб и посмотреть, как изменения повлияют на уже размещенные на сцене объекты. Для этого упражнения нам потребуется сцена из практикума «Создание нескольких экземпляров префаба». Если вы не сделали это упражнение, то нужно выполнить сначала его, а затем — следующие действия.

1. Откройте сцену **Lamps** из практикума «Создание нескольких экземпляров префаба».
2. Выберите префаб **Lamp** на панели **Project** и разверните его (нажав стрелку справа). Выберите дочерний игровой объект **Sphere**. Измените цвет источника света на красный. Обратите внимание, как префабы на сцене автоматически изменятся (см. рис. 11.9).
3. Выберите один экземпляр префаба **Lamp** на сцене. Разверните его, нажав стрелку слева от его имени на панели **Hierarchy**, а затем выберите дочерний объект **Sphere**. Установите обратно белый цвет света. Обратите внимание, что другие объекты при этом не изменяются.
4. Не сбрасывая выделение с экземпляра префаба **Lamp**, нажмите кнопку **Apply**, чтобы обновить префаб изменениями модифицированного экземпляра префаба **Lamp**. Обратите внимание, что все экземпляры префаба **Lamp** тоже стали белыми.

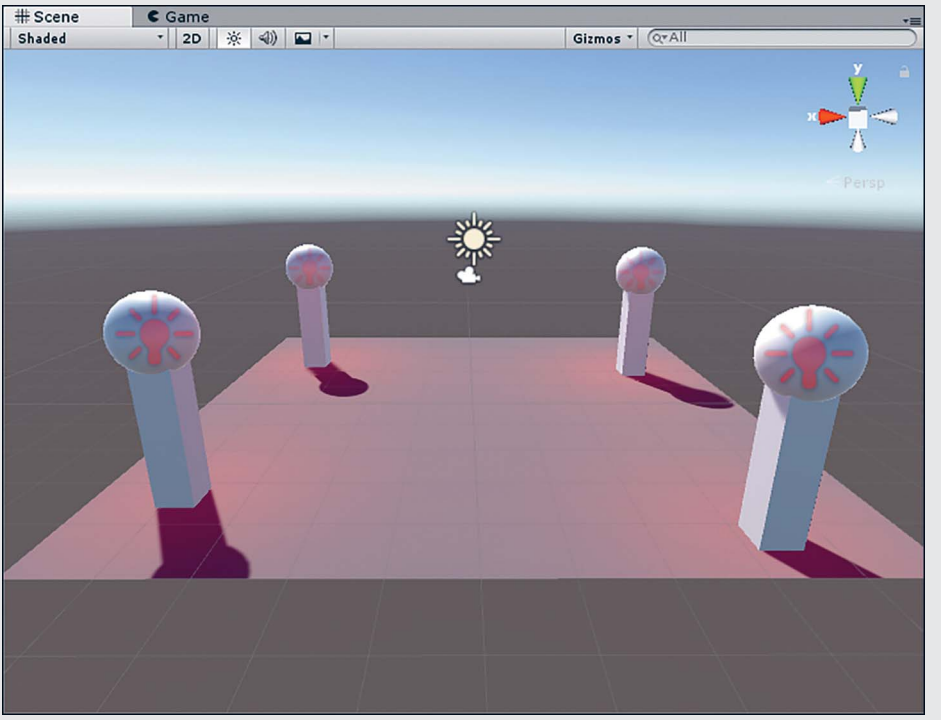


Рис. 11.9. Модифицированные экземпляры префаба **Lamp**

Разрыв связи префабов

Иногда требуется разорвать связь экземпляра префаба с ассетом префаба. Вам может потребоваться сделать это, если вам нужен объект префаба, но вы не хотите, чтобы изменения префаба затрагивали этот объект. Разрыв связи экземпляра и префаба не изменяет сам экземпляр. Он сохраняет все свои объекты, компоненты и свойства. Единственное отличие в том, что он перестает быть экземпляром префаба, и наследование его больше не касается.

Чтобы разорвать связь объекта и ассета префаба, выделите объект на панели **Hierarchy**. После этого выберите команду **GameObject ⇒ Break Prefab Instance**. Объект не изменится, но его имя снова станет черным, а не синим. После разрыва связи ее можно восстановить, нажав кнопку **Revert** на панели **Inspector**.

Создание экземпляра префаба с помощью кода

Использование на сцене префабов — отличный способ сделать последовательный и тщательно спланированный уровень. Однако иногда требуется создавать

экземпляры во время выполнения игры. Это может быть, например, возрождение врагов или их спаун в случайных местах. Также вам, возможно, потребуется столько экземпляров, что размещать их вручную уже не представляется возможным. Какой бы ни была причина, создание экземпляров префабов с помощью кода — самый удобный вариант.

Есть два способа создания объекта префаба на сцене, и в обоих из них используется метод `Instantiate()`. Первый способ заключается в следующем:

```
Instantiate(GameObject prefab);
```

Как видите, этот метод считывает переменную типа `GameObject` и создает ее копию. Координата, поворот и масштаб нового объекта будут такими же, как у исходного экземпляра.

Второй способ использования метода `Instantiate()` следующий:

```
Instantiate(GameObject prefab, Vector3 position, Quaternion rotation);
```

Здесь метод принимает три параметра. Первый — это копируемый объект. Второй и третий — положение и вращение нового объекта. Обратите внимание, что вращение хранится в *кватернионе*. Запомните, что в Unity так называется вектор, содержащий информацию об угловом положении (истинное назначение кватернионов в этом часе не рассматривается).

В упражнениях в конце часа показаны два примера создания экземпляров объекта с помощью кода.

Резюме

В этом часе вы узнали все о префабах в редакторе Unity. Мы начали с изучения основ префабов: понятие, терминология и структура. Затем вы научились создавать собственные префабы. Мы разобрались, как добавлять экземпляры на сцену, редактировать и разрывать наследственную связь. Наконец, вы узнали, как создать экземпляр префаба с помощью кода.

Вопросы и ответы

Вопрос: **Префабы, кажется, во многом похожи на классы в объектно-ориентированном программировании (ООП). Это правда?**

Ответ: Да. Между классами и префабами много общего. И те, и другие играют роль чертежей, а объекты создаются путем генерации экземпляров, при этом объекты в обоих случаях связаны с оригиналом.

Вопрос: **Сколько объектов префабов может существовать в сцене?**

Ответ: Сколько хотите. Однако имейте в виду, что при достижении определенного числа производительность игры начнет снижаться. Каждый раз, когда вы создаете экземпляр, он существует, пока не будет уничтожен. Поэтому, если вы создаете 10 тысяч экземпляров, то и на сцене их будет 10 тысяч.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Как называется размещение ассета префаба на сцене?
2. Какими двумя способами можно отредактировать ассет префаба?
3. Что такое наследование?
4. Сколькими способами можно использовать метод `Instantiate()`?

Ответы

1. Создание экземпляра.
2. Вы можете отредактировать ассет префаба с помощью панели **Project**, либо путем редактирования экземпляра на панели **Scene** и нажатия кнопки **Apply** на панели **Inspector**.
3. Это связь между ассетом префаба и его экземплярами. Ее суть в том, что при изменении ассета изменяются и сами объекты (экземпляры).
4. Два. Вы можете указать только сам префаб или дополнительно его положение и вращение.

Упражнение

Здесь мы еще раз поработаем с префабом, созданным ранее в этом часе. Теперь мы будем создавать экземпляры объектов из префабов с помощью кода и, надеюсь, повеселимся.

1. Создайте новую сцену в том же проекте, где вы храните префаб **Lamp**. Выберите префаб **Lamp** на панели **Project** и измените его положение на позицию с координатами (−1, 0, −5).

2. Удалите со сцены игровой объект **Directional Light**.
3. Добавьте на сцену пустой игровой объект и присвойте ему имя **Spawn Point**. Поместите его в позицию с координатами (1, 1, −5). Добавьте на сцену плоскость и расположите ее в позиции с координатами (0, 0, −4) с вращением (270, 0, 0).
4. Добавьте в проект скрипт. Присвойте ему имя **PrefabGenerator** и прикрепите его к объекту **Spawn Point**. В листинге 11.1 показан полный код скрипта **PrefabGenerator**.

ЛИСТИНГ 11.1. Содержимое файла *PrefabGenerator.cs*

```
using UnityEngine;

public class PrefabGenerator: MonoBehaviour
{
    public GameObject prefab;

    void Update()
    {
        // Каждый раз, нажимая клавишу B, мы будем генерировать префаб в
        // точке положения исходного префаба
        // Каждый раз, когда мы нажимаем клавишу «пробел», мы будем
        // генерировать префаб в точке спауна, к которой прикреплен этот
        // скрипт
        if (Input.GetKeyDown(KeyCode.B))
        {
            Instantiate(prefab);
        }

        if (Input.GetKeyDown(KeyCode.Space))
        {
            Instantiate(prefab, transform.position, transform.rotation);
        }
    }
}
```

5. Выделив объект **Spawn Point**, перетащите префаб **Lamp** на свойство **Prefab** компонента **Prefab Generator**. Теперь запустите сцену. Обратите внимание, как нажатие клавиши **B** создает экземпляр префаба **Lamp** в позиции по умолчанию, а нажатие клавиши **Пробел** создает объект в точке спауна.

12-Й ЧАС

Инструменты для создания 2D-игр

Что вы узнаете в этом часе

- Как работает ортографическая камера
- Как располагать спрайты в 3D-пространстве
- Как спрайты движутся и сталкиваются

Движок Unity обладает колоссальными возможностями создания 2D и 3D-игр. В полноценной 2D-игре все ассеты представляют собой простые плоские изображения — спрайты. В 3D-игре ассетами являются 3D-модели с наложенными на них 2D-текстурами. Как правило, в 2D-игре игрок может двигаться только в двух измерениях (например, влево, вправо, вверх, вниз). В этом часе вы освоите приемы создания 2D-игр на движке Unity.

Основы 2D-игр

Принципы проектирования 2D-игр точно такие же, как и у 3D-игр. При разработке требуется рассмотреть основные понятия, правила и требования к игре. У 2D-игр есть свои плюсы и минусы. С одной стороны, 2D-игра, скорее всего, будет проще и намного дешевле в производстве. С другой стороны, при ограничениях, связанных с 2D, некоторые типы игр создать невозможно. 2D-игры состоят из изображений, называемых *спрайтами*. Это своего рода картонные фигурки, вроде тех, которыми дети показывают свои маленькие спектакли. Вы можете перемещать их, располагать друг перед другом и заставлять взаимодействовать, создавая таким образом насыщенную среду.

При создании нового проекта на движке Unity вы можете выбрать шаблон 2D или 3D (рис. 12.1). Установка переключателя в положение **2D** задает для сцены 2D-вид по умолчанию и ортографическую камеру (как описано позже в этом часе). Еще один момент, характерный для 2D-проектов, как вы позже заметите, заключается

в том, что на сцене отсутствует направленный свет или скайбокс. На самом деле в 2D-играх, как правило, вообще нет света, так как спрайты отрисовываются простым типом рендера под названием «рендер спрайтов». В отличие от текстуры, на отрисовку спрайтов освещение обычно не влияет.

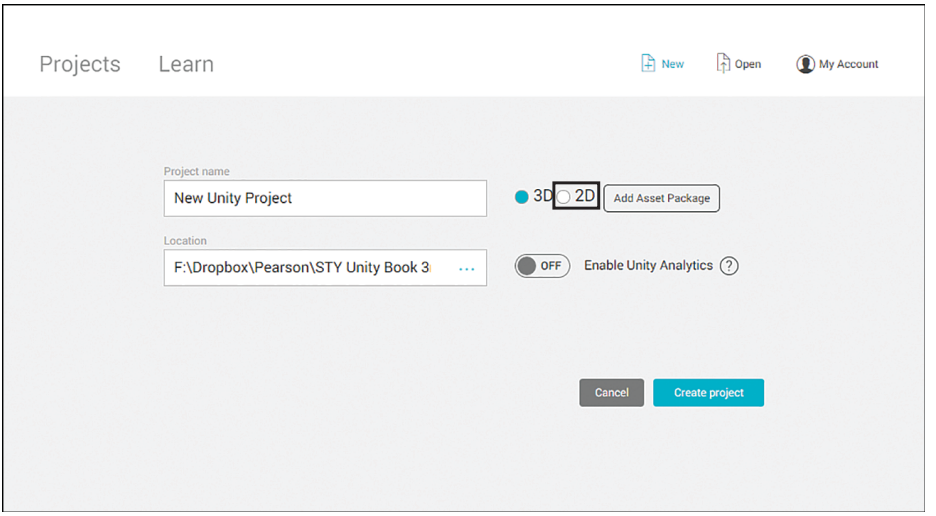


Рис. 12.1. Создание проекта с 2D-шаблоном

СОВЕТ

Проблемы 2D-игр

При проектировании 2D-игр порой возникают уникальные проблемы.

- Труднее создать погружения из-за отсутствия глубины.
- Многие игровые жанры в 2D вообще не реализуются.
- 2D-игры, как правило, не освещаются, поэтому спрайты нужно рисовать и организовывать с особой тщательностью.

Панель Scene в 2D-разработке

Проект с 2D-шаблоном по умолчанию сразу открывается с 2D-сценой. Для переключения между режимами 2D и 3D вы можете нажать кнопку **2D** в верхней части панели **Scene** (см. рис. 12.2). 3D-гизмо в режиме 2D не отображается.

Вы можете перемещаться в 2D-режиме, щелкнув правой (или средней) кнопкой мыши по сцене и перетащив ее. Для масштабирования вы можете использовать колесо мыши или жест прокрутки на сенсорной панели. В качестве ориентира

вам подойдет фоновая сетка. Гизмо сцены больше не отображается, но он вам и не нужен. Ориентация сцены в 2D-режиме не меняется.

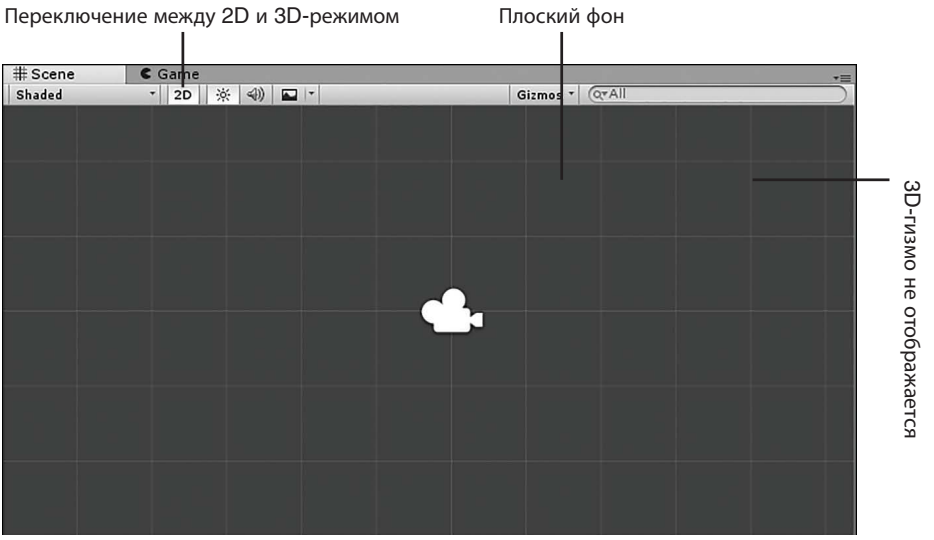


Рис. 12.2. 2D-вид сцены

ПРАКТИКУМ

Создание и размещение спрайтов

Вопросы импортирования и использования спрайтов мы рассмотрим позже. Для начала создадим простой спрайт и добавим его на сцену.

1. Создайте новый проект, на этот раз выбрав шаблон **2D** (см. рис. 12.1). Параметры проекта по умолчанию теперь установлены как для 2D-игры. (Обратите внимание, что в 2D-игре нет направленного света.)
2. Добавьте в проект новую папку и назовите ее **Sprites**. Щелкните правой кнопкой мыши на панели **Project** и выберите команду **Create** ⇒ **Sprites** ⇒ **Hexagon**.
3. Перетащите вновь созданный спрайт шестиугольника на панель **Scene**. Обратите внимание на компонент **Sprite Renderer** на панели **Inspector**.
4. Изначально спрайт будет белым, но вы можете изменить его цвет с помощью компонента **Sprite Renderer** — сделайте это (рис. 12.3)!

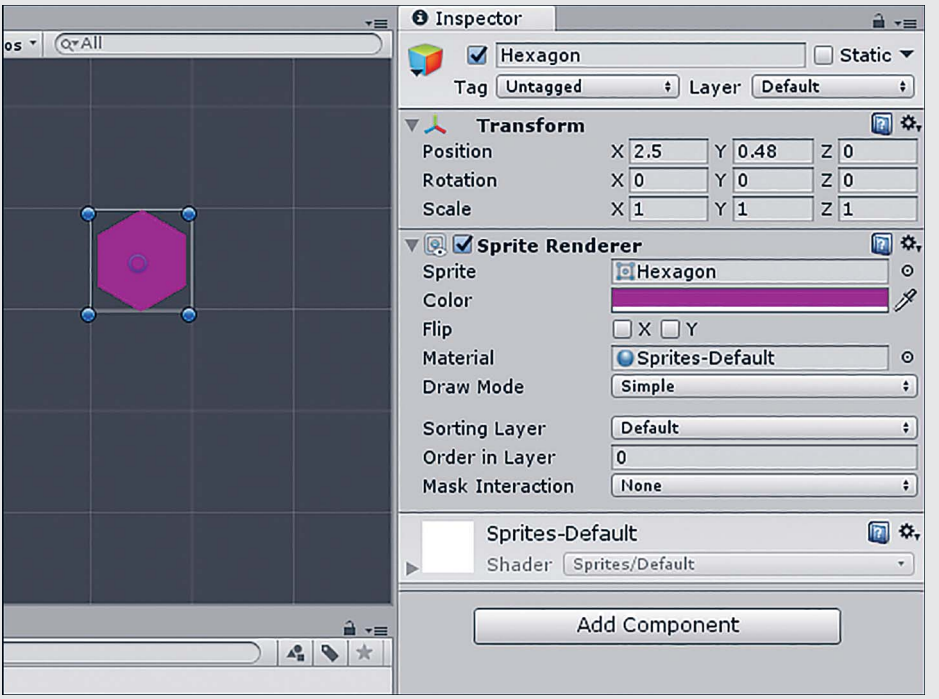


Рис. 12.3. Спрайт шестиугольника

СОВЕТ

Инструмент Rect

Как мы говорили в часе 2, инструмент преобразования **Rect** идеально подходит для работы с прямоугольными спрайтами. Этот инструмент расположен в верхнем левом углу редактора Unity, а также вызывается нажатием клавиши **T**. Попробуйте поэкспериментировать с ним, пока у вас есть спрайт на экране.

Ортографические камеры

Поскольку вы создали 2D-проект, камера по умолчанию имеет «ортографический» тип. Это означает, что она не отображает перспективные искажения, а размер и форма всех объектов остаются неизменными независимо от расстояния. Этот тип камеры подходит для 2D-игр, где кажущаяся глубина спрайтов изменяется за счет размера и расположения на слое сортировки.

Слой сортировки позволяет определить, какие спрайты и группы спрайтов отображаются впереди других. Представьте, что вы расставляете декорации на сцене в театре. Нужно, чтобы фоновые декорации находились за декорациями переднего плана. Если правильно отсортировать спрайты переднего плана и выбрать соответствующие размеры, можно создать ощущение глубины.

Вы можете увидеть тип камеры, выбрав объект **Main Camera**: свойство **Projection** на панели **Inspector** содержит значение **Orthographic** (рис. 12.4). (Остальные настройки камеры описаны в часе 5).

СОВЕТ

Размер ортографической камеры

Часто бывает нужно изменить относительные размеры сразу всех спрайтов на сцене, не затрагивая их реальных размеров. Если камера не воспринимает глубину, вы можете спросить — «а как, собственно, сделать это, если приблизить камеру нельзя?».

Здесь необходимо использовать свойство **Size** на панели **Inspector** (рис. 12.4), которое доступно только у ортографической камеры. Это дистанция в глобальных единицах от центра до верхней части поля зрения камеры. Такое может показаться странным, но тут все дело в соотношении сторон. По сути дела, бóльшие значения означают «отдалить», а меньшие — «приблизить».

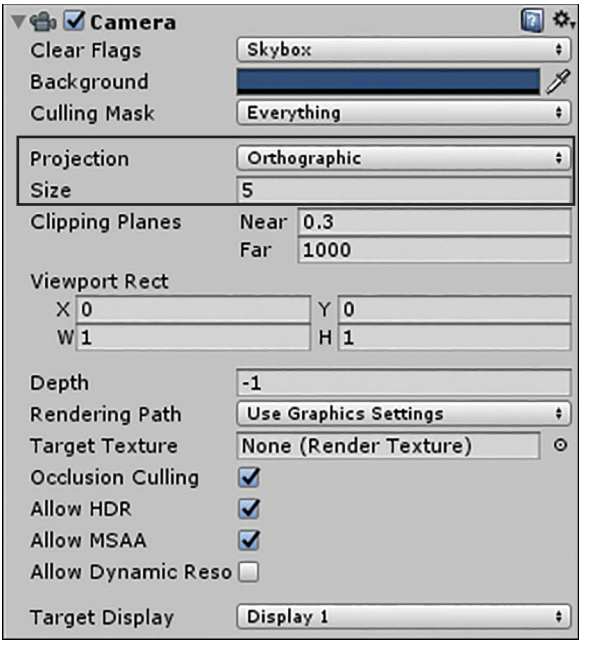


Рис. 12.4. Установка размера ортографической камеры

Добавление спрайтов

Добавить на сцену спрайт очень легко. После того как спрайт будет импортирован в проект (что, опять же, проще простого), вам нужно лишь перетащить его на сцену.

Импортирование спрайтов

Если вы хотите использовать собственные изображения в качестве спрайтов, вы должны сообщить Unity, что это спрайт. Импорт спрайта осуществляется следующим образом.

1. Найдите файл *ranger.png* в приложенных к книге файлах примеров (или используйте свой графический файл).
2. Перетащите изображение на панель **Project** в редакторе Unity. Если вы создали 2D-проект, оно автоматически импортируется как спрайт. Если вы создали 3D-проект, вам нужно сообщить программе Unity, чтобы она использовала изображение как спрайт. Для этого присвойте свойству **Texture Type** значение **Sprite** (рис. 12.5).

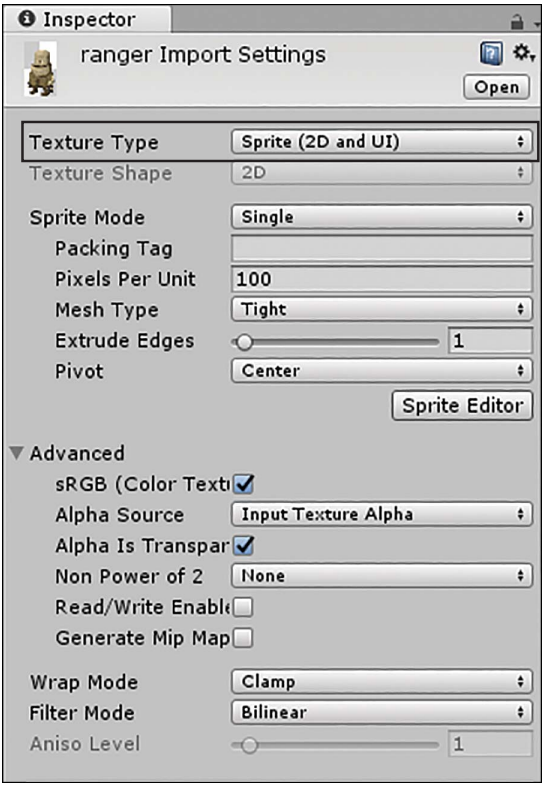


Рис. 12.5. Превращение текстуры в спрайт

3. Перетащите спрайт на панель **Scene**. Проще некуда! Обратите внимание, что если панель **Scene** не находится в 2D-режиме, то спрайт может оказаться в ненулевом положении по оси `z` (как и все остальные спрайты).

Режимы спрайтов

Одиночные спрайты — это хорошо, но настоящее веселье начинается тогда, когда вы создаете анимацию. В Unity есть мощные инструменты для использования листов спрайтов. *Лист спрайтов* представляет собой изображение с несколькими кадрами анимации, расположенными в сетке.

Панель **Sprite Editor** (которую мы как раз собираемся изучить) позволяет автоматически извлекать десятки или сотни различных кадров анимации из одного изображения.

ПРАКТИКУМ

Изучение режимов спрайтов

Выполните следующие действия, чтобы исследовать различие между режимами спрайтов **Single** и **Multiple**.

1. Создайте новый 2D-проект или новую сцену в существующем 2D-проекте.
2. Импортируйте файл *rangerTalking.png* из приложенных к книге файлов примеров (или используйте свой лист спрайтов).
3. Убедитесь в том, что параметру **Sprite Mode** присвоено значение **Single**, и разверните вложение спрайта на панели **Project**, чтобы убедиться, что движок Unity обрабатывает это изображение как одиночный спрайт (рис. 12.6).

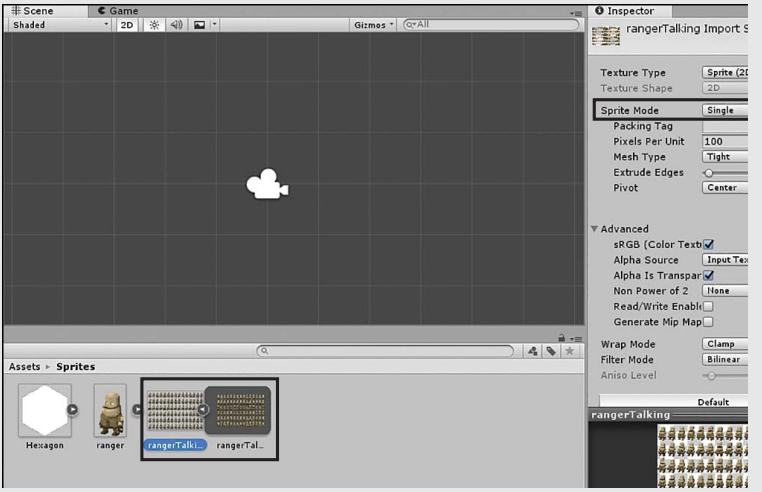


Рис. 12.6. Импорт спрайта в режиме **Single**

4. Измените режим спрайта на **Multiple** и нажмите кнопку **Apply** в нижней части панели **Inspector**. Обратите внимание, что стрелки вложения теперь нет.
5. Нажмите кнопку **Sprite Editor** на панели **Inspector**, после чего появится одноименная панель. Откройте раскрывающийся список **Slice**, задайте свойству **Type** значение **Automatic** и нажмите кнопку **Slice** (рис. 12.7). Обратите внимание, как контуры обнаруживаются автоматически, но в каждом кадре по-другому.
6. Задайте свойству **Type** значение **Grid by Cell Size** и настройте сетку под ваш спрайт. Для прилагаемого изображения настройки следующие: **–x = 62**, **y = 105**. Оставьте другие параметры как есть и нажмите кнопку **Slice**. Обратите внимание, что границы стали больше.
7. Нажмите кнопку **Apply**, чтобы сохранить изменения и закрыть панель **Sprite Editor**.
8. Посмотрите на свой спрайт на панели **Project** и обратите внимание, что теперь здесь отображается каждый отдельный кадр, готовый к анимации.

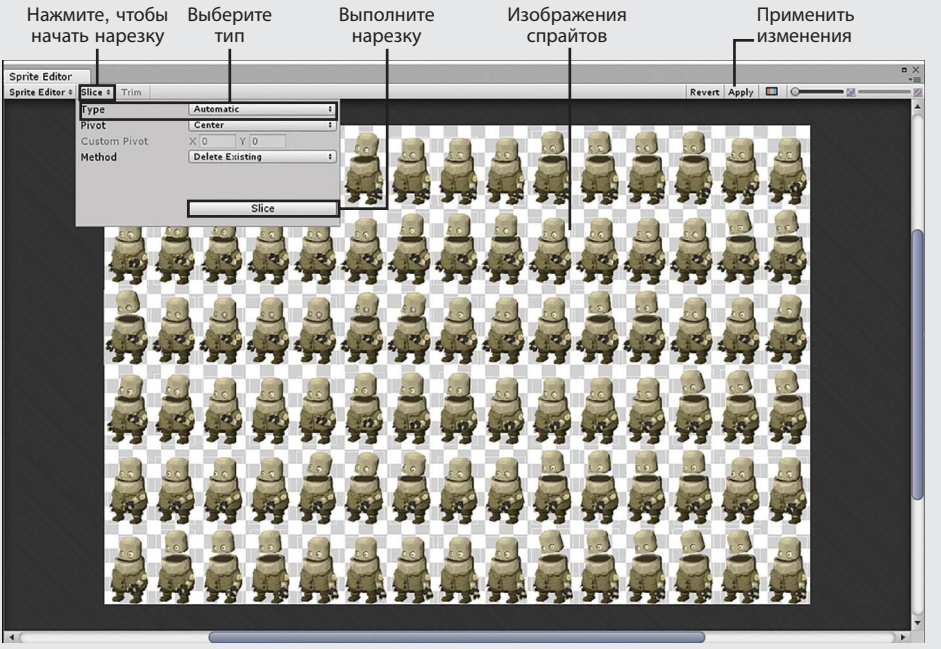


Рис. 12.7. Панель **Sprite Editor**

ПРИМЕЧАНИЕ

Анимирование листа спрайтов

В практикуме «Изучение режимов спрайтов» мы показали, как можно импортировать и настроить лист спрайтов. Как уже упоминалось ранее, листы спрайтов часто используются для 2D-анимации. Мы подробнее изучим анимацию спрайтов в часе 17.

Размеры импортированных спрайтов

Если вам нужно масштабировать изображения спрайтов, чтобы их размеры соответствовали необходимым, у вас есть два пути. Лучший способ отредактировать размеры спрайта — открыть его в графическом редакторе, например Photoshop, и внести соответствующие исправления. Однако не всегда есть такая возможность (или время). Как вариант, вы можете использовать инструмент масштабирования на сцене. Но это неэффективно и может стать причиной искажений при масштабировании в будущем.

Если необходимо на постоянной основе масштабировать спрайт, например **SpriteA**, который должен быть вдвое меньше, можно использовать параметр **Pixels per Unit** при импорте. Этот параметр определяет, сколько единиц мира спрайт занимает при заданном разрешении. Например, изображение с разрешением 640×480 и значением параметра **Pixels per Unit** равным **100** будет занимать 6,4×4,8 мировых единиц.

Порядок отрисовки

Чтобы задать, какие спрайты отрисовываются впереди, а какие — сзади, в Unity есть система слоев сортировки. Сортировка по слоям управляется двумя параметрами компонента **Sprite Renderer**: **Sorting Layer** (Слой сортировки) и **Order in Layer** (Порядок в слое) (рис. 12.8).

Рис. 12.8. Свойства слоев спрайта по умолчанию

Слои сортировки

Вы можете использовать слои сортировки, чтобы распределить спрайты по группам разной глубины. При создании нового проекта генерируется только один слой сортировки с именем **Default** (рис. 12.8). Представьте себе простенький 2D-платформер с фоном из холмов, деревьев и облаков. В этом случае вам наверняка понадобится слой сортировки под названием **Background**.

ПРАКТИКУМ

Создание слоев сортировки

Выполните следующие действия, чтобы создать новый слой сортировки и назначить его спрайту.

1. Создайте новый проект и импортируйте весь пакет 2D-ассетов (выбрав команду **Assets** ⇒ **Import Package** ⇒ **2D**).
2. Найдите спрайт **BackgroundGreyGridSprite** в папке **Standard Assets\2D\Sprites** и перетащите его на сцену. Установите его в позицию с координатами (−2,5, −2,5, 0) и задайте масштаб (2, 2, 1).
3. Добавьте новый слой сортировки, открыв раскрывающийся список **Sorting Layer** компонента **Sprite Renderer**, а затем нажав кнопку **Add Sorting Layer** (рис. 12.9).

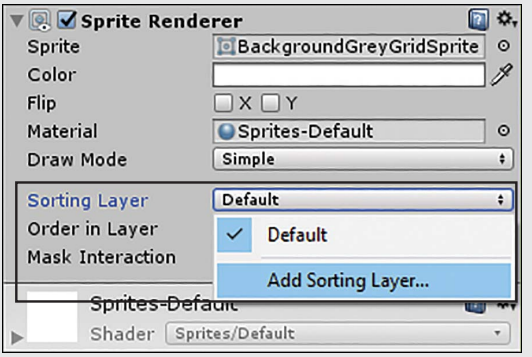


Рис. 12.9. Добавление нового слоя сортировки

4. Добавьте два новых слоя с именами **Background** и **Player**. Для этого нажмите кнопку **+** под списком слоев (рис. 12.10). Обратите внимание, что нижний слой в списке, в нашем случае, слой **Player**, будет отрисовываться последним и окажется поверх всех остальных слоев в игре. Хорошо бы переместить слой **default** вниз, чтобы новые элементы отрисовывались поверх других слоев.

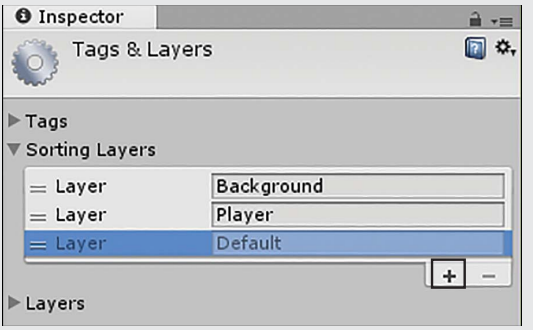


Рис. 12.10. Управление слоями сортировки

5. Еще раз выберите игровой объект **BackgroundGreyGridSprite** и задайте в качестве его слоя сортировки созданный слой **Background**.
6. На панели **Project** найдите спрайт **RobotBoyCrouch00** и перетащите на сцену. Поместите его в позицию с координатами (0, 0, 0).
7. Задайте ему слой **Player** в качестве слоя сортировки (рис. 12.11). Обратите внимание, что игрок отрисовывается поверх фона. Теперь попробуйте открыть инструмент **Tags & Layers Manager** (выбрав команду **Edit** ⇒ **Project Settings** ⇒ **Tags and Layers**) и переставить слои так, чтобы фон оказался на переднем плане.

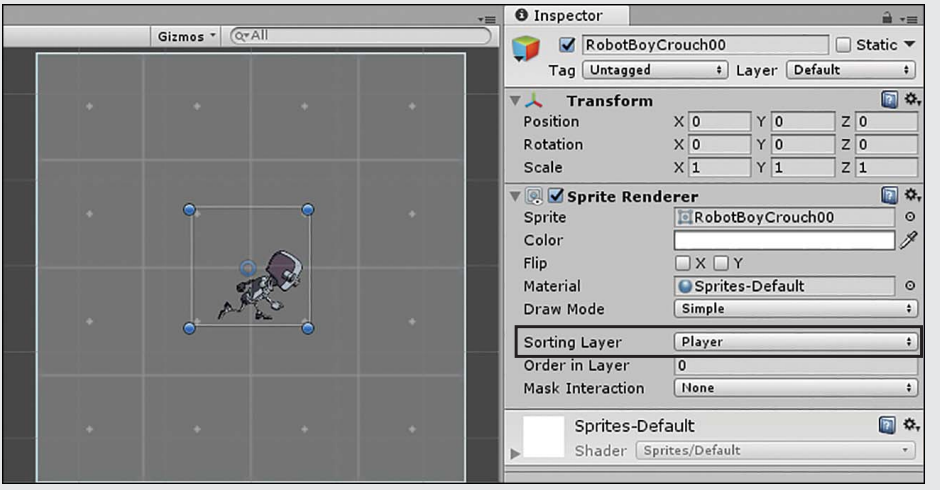


Рис. 12.11. Настройка слоя сортировки спрайта

Скорее всего, вам потребуется еще один слой для возможных бонусов, пуль и так далее. Земля или платформа могут находиться на другом слое. И наконец, если вы хотите добиться ощущения глубины, добавив тонкие спрайты для переднего плана, можно сделать слой **Foreground**.

Порядок спрайтов в слое

Создав основные слои и распределив спрайты по ним, можно настроить порядок спрайтов в слое. Суть проста — спрайты с более высокими значениями отрисовываются поверх спрайтов с меньшими значениями.

ВНИМАНИЕ!

Таинственное исчезновение спрайтов

В начале вашей работы спрайты могут внезапно пропадать. Это, как правило, связано с тем, что спрайт находится позади более крупного элемента или камеры.

Вы также можете обнаружить, что, если вы неправильно настроили слои сортировки и порядок на слое, положение по оси *z* может повлиять на порядок отображения. Поэтому важно всегда задавать слой для каждого спрайта.

2D-физика

Теперь, когда вы понимаете, как в вашей игре работают статичные спрайты, пришло время добавить немного магии и движения. В Unity есть несколько очень крутых инструментов, которые помогут вам с этим. Существует два основных способа «оживить» спрайт. Давайте рассмотрим их подробнее. В Unity есть мощная система 2D-физики, интегрированная в систему под названием Box2D. Так же, как и в 3D-играх, вы можете использовать компоненты **Rigidbody** с гравитацией, различные коллайдеры и другие физические эффекты, характерные для 2D-платформ и гонок.

Компонент Rigidbody в 2D-играх

В Unity предусмотрен отдельный тип компонента **Rigidbody** для 2D-игр. Многие свойства этого компонента отличаются от свойств его 3D-аналога, с которым вы уже работали. Выбор неправильного типа **Rigidbody** — распространенная ошибка, так что будьте внимательны и воспользуйтесь командой **Add Component** ⇒ **Physics 2D**, а не другой.

СОВЕТ

Смешивание типов физики

2D- и 3D-физика могут сосуществовать на одной сцене, но при этом они не будут взаимодействовать друг с другом. Кроме того, вы можете применить 2D-физику к 3D-объектам и наоборот!

2D-коллайдеры

Так же, как 3D-коллайдеры, 2D-коллайдеры позволяют игровым объектам взаимодействовать друг с другом при соприкосновении. В Unity есть много встроенных 2D-коллайдеров (см. таблицу 12.1).

ТАБЛ. 12.1. 2D-коллайдеры

Коллайдер	Описание
Circle Collider 2D	Круг с фиксированным радиусом и смещением
Box Collider 2D	Прямоугольник с регулируемой шириной, высотой и смещением
Edge Collider 2D	Ломаная линия, необязательно замкнутая
Capsule Collider 2D	Капсула с заданным смещением, размером и направлением
Composite Collider 2D	Особый коллайдер, который представляет собой совокупность нескольких простых коллайдеров, обрабатываемых как один
Polygon Collider 2D	Замкнутый многоугольник с тремя или более сторонами

ПРАКТИКУМ

Сталкивание двух спрайтов

Выполните следующие действия, чтобы попрактиковаться со столкновением 2D-спрайтов с помощью двумерных квадратов.

1. Создайте новую 2D-сцену и убедитесь, что пакет 2D-ассетов импортирован.
2. Найдите спрайт **RobotBoyCrouch00** в папке **Assets\Standard Assets\2D\Sprites** и перетащите его на панель **Hierarchy**. Расположите спрайт в позиции с координатами (0, 0, 0).
3. Добавьте в спрайт многоугольный коллайдер (выбрав команду **Add Component ⇒ Physics 2D ⇒ Polygon Collider 2D**). Обратите внимание, что этот коллайдер примерно соответствует контуру спрайта.
4. Продублируйте этот спрайт и переместите дубликат ниже и правее в позицию с координатами (0,3, –1, 0). В этом случае верхний спрайт упадет на него.
5. Чтобы верхний спрайт реагировал на гравитацию, добавьте к нему компонент **Rigidbody 2D** (выбрав команду **Add Component ⇒ Physics 2D ⇒ Rigidbody 2D**). Подходящие значения свойств показаны на рис. 12.12.
6. Запустите сцену и обратите внимание на поведение спрайта. Столкновения происходят точно так же, как в 3D-играх.

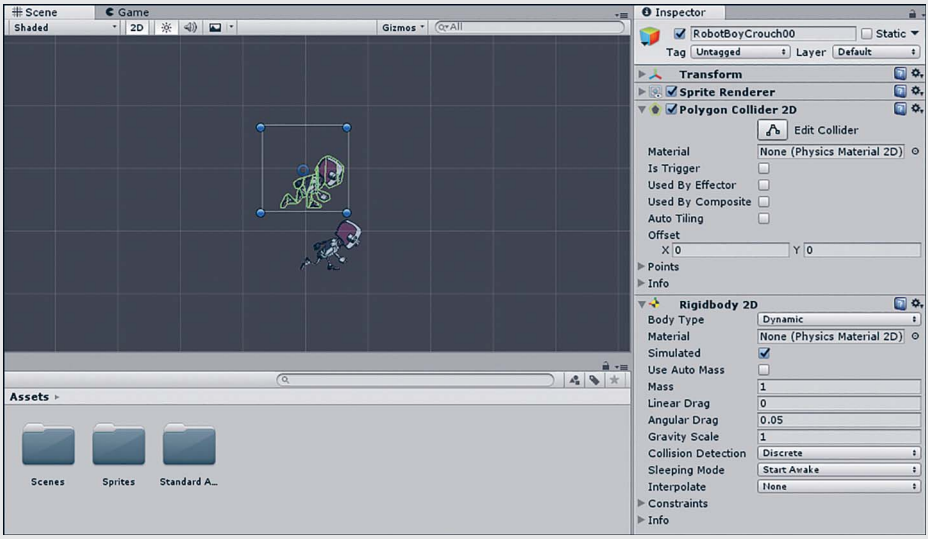


Рис. 12.12. Настройки верхнего робота

СОВЕТ

2D-коллайдеры и глубина

Кое-что в 2D-коллайдерах может показаться вам странным, если смотреть на них на 3D-сцене. Особенность в том, что им не обязательно быть на одной глубине *z*, чтобы столкнуться. Столкновение учитывает только координаты *x* и *y*.

Резюме

В этом часе вы узнали об основах создания 2D-игр на движке Unity. Сначала мы погрузились в изучение ортографической камеры и работу с глубиной в двумерном пространстве. Затем мы создали простой 2D-объект, способный двигаться и сталкиваться. Эти свойства лежат в основе многих 2D-игр.

Вопросы и ответы

Вопрос: **Подходит ли Unity для создания серьезных 2D-игр?**

Ответ: Да. В Unity есть невероятный набор инструментов для создания 2D-игр.

Вопрос: **Можно ли в Unity создавать 2D-игры для мобильных и других платформ?**

Ответ: Очень даже можно! Одно из основных преимуществ движка Unity заключается в его способности довольно легко разворачивать игры под многие платформы. 2D-игры не исключение, и многие весьма популярные 2D-игры сделаны именно в Unity.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Какой тип камеры отображает все объекты без искажений перспективы?
2. От чего зависит настройка размера ортографической камеры?
3. Должны ли два 2D-спрайта находиться на одной глубине, чтобы столкнуться?
4. Будут ли спрайты отрисовываться, находясь за камерой?

Ответы

1. Ортографическая.
2. Настройка размера определяет половину вертикальной высоты в мировых единицах, которую охватывает камера.
3. Нет. 2D-столкновения учитывают только координаты *x* и *y*.
4. Нет. Это частая причина потери спрайтов при создании 2D-игр.

Упражнение

В этом упражнении мы попробуем использовать некоторые инструменты Unity, чтобы посмотреть, чего можно достигнуть, добавив спрайтам щепотку анимации, капельку скриптов и парочку коллайдеров.

1. Создайте новый 2D-проект или новую сцену в существующем проекте.
2. Импортируйте пакет 2D-ассетов (выбрав команду **Assets** ⇒ **Import Package** ⇒ **2D**). Импортируйте весь пакет.
3. Найдите ассет **CharacterRobotBoy** в папке **Assets\Standard Assets\2D\Prefabs** и перетащите его на панель **Hierarchy**. Установите его в позицию с координатами (3, 1,8, 0). Обратите внимание, что у этого префаба на панели **Inspector** есть множество компонентов.

4. Найдите спрайт **PlatformWhiteSprite** в папке **Assets\Standard Assets\2D\Sprites** и перетащите его на панель **Hierarchy**. Установите его в позицию с координатами (0, 0, 0) и задайте масштаб (3, 1, 1). Добавьте компонент **Box Collider 2D**, чтобы игрок не проваливался сквозь платформу!

5. Продублируйте платформу. Поместите дубликат в позицию с координатами (7,5, 0, 0), задайте вращение (0, 0, 30), чтобы сделать наклонную платформу, и задайте масштаб (3, 1, 1).

6. Переместите компонент **Main Camera** в позицию с координатами (11, 4, −10) и настройте ее размер, сделав равным 7. На рис. 12.13 приведены финальные настройки.

7. Нажмите кнопку **Play**. С помощью клавиш со стрелками и **Пробел** вы можете перемещаться и прыгать.

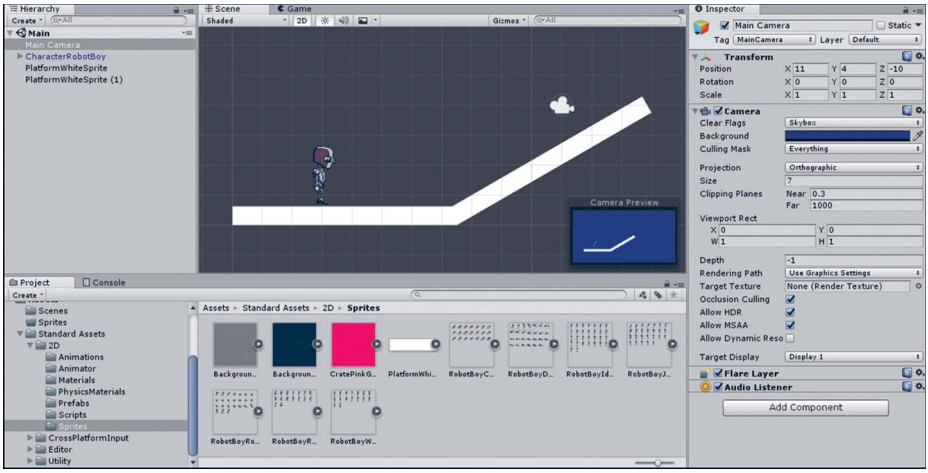


Рис. 12.13. Окно программы с законченным упражнением

13-Й ЧАС
Тайлмапы в 2D-играх

Что вы узнаете в этом часе

- Что такое тайлмап
- Как создавать палитры
- Как создавать и размещать тайлы
- Как добавить физику тайлмапам

Ранее (в часе 4) вы научились создавать 3D-мир с помощью системы ландшафта в Unity. Теперь пришло время применить эти знания для 2D-игр с помощью новой системы 2D-тайлмапов Unity. Благодаря ей вы сможете быстро и легко создавать миры, в которые игроки с удовольствием будут погружаться. В этом часе мы начнем с тайлмапов, затем изучим палитры, в которых будут храниться ваши тайлы. Наконец, мы создадим тайлы и нанесем их на тайлмап, а затем добавим тайлмапам компоненты столкновения для интерактивности.

Введение в тайлмапы

Как следует из названия, *тайлмап* — это «карта» тайлов (так же, как битмапы — это карты битов). Тайлмапы устанавливаются на сетки, которые определяют размеры и отступы тайлов для всего тайлмапа. Тайлы — это отдельные спрайты, предназначенные для рисования мира. Они располагаются на палитре, которая затем используется для нанесения тайлов на тайлмап. Поначалу новые слова могут сбить с толку, но это, в целом, сводится к простым понятиям, вроде «краски, палитра, кисти, холст», и как только мы начнем работу, все окажется гораздо проще.

Если это не так уж очевидно, следует отметить, что тайлмапы обычно используются в 2D-играх. В принципе, технически их можно применять в 3D-пространствах, но это неэффективно. Можно также использовать листы спрайтов в сочетании с тайлмапами. Вы можете создать лист с разными частями среды и превращать их в тайлы.

Создание тайлмапа

Вы можете разместить на сцене столько тайлмапов, сколько душе угодно, и наверняка вы скоро будете создавать их по нескольку штук и наслаивать один на другой. Этот метод позволяет настроить задний, средний и передний план для появления эффекта параллакса. Чтобы создать на сцене тайлмап, выберите команду **GameObject ⇒ 2D Object ⇒ Tilemap**. Программа Unity добавит на сцену два игровых объекта с именами **Grid** и **Tilemap** (рис. 13.1).

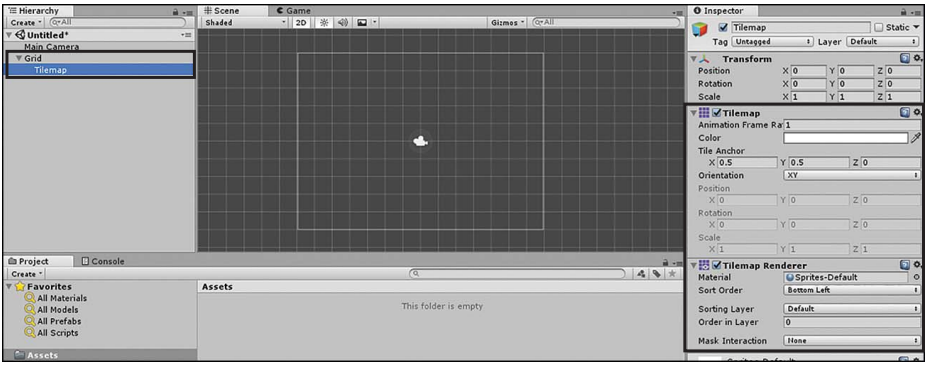


Рис. 13.1. Добавление тайлмапа

У игрового объекта **Tilemap** есть два компонента, о которых стоит поговорить отдельно: **Tilemap** и **Tilemap Renderer**. Компонент **Tilemap** отвечает за размещение тайлов, расположение якорей (точек привязки) и общую цветовую тонировку. Компонент **Tilemap Renderer** позволяет задать порядок сортировки, чтобы ваши тайлмапы отрисовывались в правильном порядке.

ПРАКТИКУМ

Добавление тайлмапа на сцену

В этом упражнении мы добавим на сцену тайлмап. Не забудьте сохранить сцену, так как она понадобится нам позднее в этом часе. Выполните следующие действия.

1. Создайте новый 2D-проект. Создайте новую папку с именем **Scenes** и сохраните сцену в нее.
2. Добавьте на сцену тайлмап, выбрав команду **GameObject ⇒ 2D Object ⇒ Tilemap**.
3. Этот тайлмап станет фоном вашей сцены, так что переименуйте его в **Background**.
4. Чтобы создать еще один тайлмап, не создавая при этом еще одну сетку, щелкните правой кнопкой мыши по игровому объекту **Grid** на панели **Hierarchy**

и выберите команду **2D Object** ⇒ **Tilemap** (рис. 13.2). Присвойте новому тайлмапу имя **Platforms**.

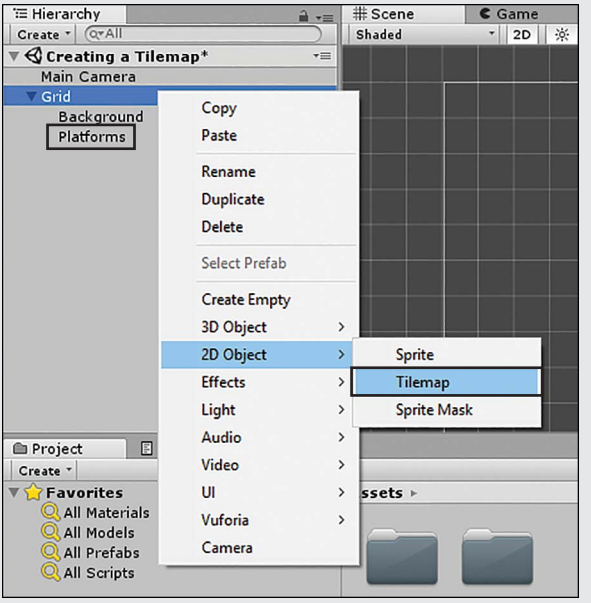


Рис. 13.2. Добавление нового тайлмапа на панели **Hierarchy**

5. Добавьте в проект два новых слоя сортировки и назовите их **Background** и **Foreground**. (Повторите материалы часа 12, если не помните, как добавлять слои сортировки.)
6. Выберите тайлмап **Background** и задайте свойству **Sorting Layer** компонента **Tilemap Renderer** значение **Background**. Выберите тайлмап **Platforms** и задайте свойству **Sorting Layer** компонента **Tilemap Renderer** значение **Foreground**.

Сетка

Как вы видели ранее, при добавлении на сцену тайлмапа вы также добавляете игровой объект **Grid** (рис. 13.3). Этот объект сетки задает параметры, которые применяются ко всем аналогичным тайлмапам. В частности, сетка задает размер ячейки и разрывы между ячейками ваших тайлмапов. Поэтому, если все тайлмапы должны быть одинакового размера, вам достаточно сделать одну сетку на всех. Если же нет, то вам понадобится несколько сеток для тайлмапов разных размеров. Обратите внимание, что размер ячейки по умолчанию равен 1. Эта информация нам еще потребуется, запомните ее.

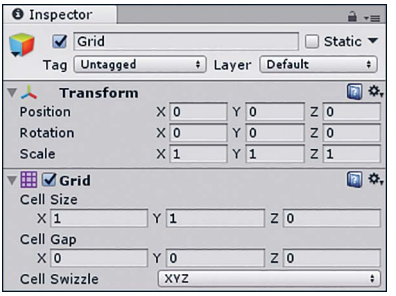


Рис. 13.3. Объект **Grid**

СОВЕТ

Наклонные тайлмапы

Обычно тайлмапы выравниваются относительно друг друга. Но не всегда. Если вы хотите, чтобы один из фоновых тайлмапов располагался под углом, вы можете повернуть его на панели **Scene**. Вы даже можете перемещать тайлмапы, чтобы располагать их в шахматном порядке или отдалять их по оси *z* для создания параллактического эффекта.

Палитры

Чтобы нанести тайлы на тайлмап, необходимо сначала собрать их на палитре. Это примерно то же самое, что палитра художника, на которой располагаются все имеющиеся краски. У палитры есть большое количество инструментов, которые помогут вам создавать любые миры, доступные вашей фантазии. Чтобы открыть палитру тайлов, выберите команду меню **Window** ⇒ **Tile Palette**. Откроется панель **Tile Palette**, показанная на рис. 13.4.

Рис. 13.4. Панель **Tile Palette**

Панель Tile Palette

На панели Tile Palette есть ряд инструментов для рисования, а также центральная область, где находятся все ваши тайлы. По умолчанию в проекте нет палитры, но вы можете создать ее, открыв раскрывающийся список **Create New Palette**.

ПРАКТИКУМ

Создание палитры

Теперь настало время добавить в проект парочку палитр. Используйте проект, созданный в практикуме «Добавление тайлмапа на сцену», а если вы еще не выполнили его — следует сделать это сейчас. Обязательно сохраните сцену, так как она понадобится нам позднее в этом часе. Когда вы будете готовы создать палитру, выполните следующие действия.

1. Откройте сцену, созданную в практикуме «Добавление тайлмапа на сцену». Откройте панель **Tile Palette** (выбрав пункт меню **Window** ⇒ **Tile Palette**) и закрепите ее рядом с панелью **Inspector** (рис. 13.4).

2. Добавьте палитру, нажав кнопку **Create New Palette**, и присвойте ей имя **Jungle Tiles**. Оставьте другие настройки палитры по умолчанию (рис. 13.5). Нажмите кнопку **Create**.

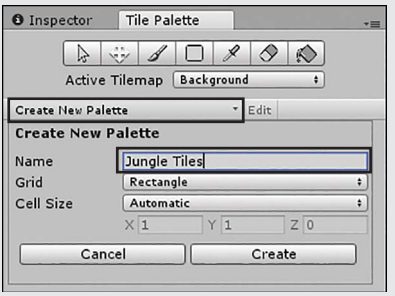


Рис. 13.5. Создание новой палитры

3. В появившемся окне **Create Palette** создайте новую папку, назовите ее **Palettes** и нажмите кнопку **Select Folder**.
4. Повторите шаги 2 и 3, чтобы создать еще одну палитру под названием **Grass Tiles**. После этого у вас должно получиться две палитры в раскрывающемся списке палитр, а также два тайлмапа в раскрывающемся списке **Active Tilemap** (рис. 13.6).

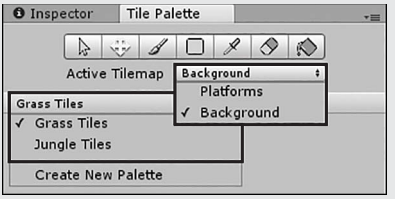


Рис. 13.6. Созданные палитры и тайлмапы

Тайлы

Ранее в этом часе мы проделали много подготовительной работы, которая позволит нам использовать тайлы. Теперь настало время взяться непосредственно за рисование. В сущности, тайлы — это спрайты, которые настроены специально для работы с тайлмапами. Спрайты, которые используются в качестве тайлов, можно применять и как обычные спрайты, если вам будет нужно. После того как спрайты импортированы и настроены, их можно превратить в тайл, добавить на палитру, а затем наносить на тайлмап.

ПРИМЕЧАНИЕ

Ох уж эти пользовательские тайлы...

В этом часе мы будем работать с базовыми тайлами. Во встроенных тайлах заложена огромная функциональность, но вы можете применить к 2D-тайлмапам кучу своих настроек. Если вы хотите пойти еще дальше и создать, например, анимированные тайлы, смарт-тайлы или даже пользовательские кисти с логикой игрового объекта, встроенные в тайлы, вам поможет пакет **Unity 2D Extras**. На момент публикации этой книги его можно скачать по ссылке **github.com/Unity-Technologies/2d-extras**. Но скоро он будет встроен в менеджер пакетов Unity и доступен так же, как и все прочие ассеты, которые вы использовали в этой книге. Так что у вас впереди много интересного!

Настройка спрайтов

Чтобы настроить спрайт для использования в качестве тайла, нужно приложить не так уж и много усилий. Вам нужно выполнить два основных задания.

1. Убедиться в том, что свойство **Pixels per Unit** на ваших спрайтах точно соответствует свойству **Cell Size** вашей сетки. (Мы в ближайшее время остановимся на этом подробнее.)

2. Нарезать спрайты (мы предполагаем, что они находятся на листе спрайтов) так, чтобы вокруг них было как можно меньше пустого пространства. Там, где это возможно, лучше вообще избавиться от него.

Первый шаг подготовки спрайта к использованию в качестве тайла может показаться сложным, но на деле все гораздо проще. К примеру, в этом часе мы воспользуемся листами спрайтов, на которых содержится несколько тайлов. Эти тайлы имеют размер 64 на 64 пикселя (так их нарисовали). Поскольку свойство сетки **Cell Size** задано 1 единица на 1 единицу, свойство **Pixels per Unit** должно равняться 64. Таким образом, каждые 64 пикселя в вашем спрайте будут равны 1 единице, что как раз соответствует размеру ячейки.

Создание тайла

После того как спрайты будут подготовлены, можно создавать тайлы. Для этого надо перетащить спрайты на нужную палитру на панели **Tile Palette** и выбрать, где вы хотите сохранить полученные тайлы. Исходные спрайты при этом остаются без изменений. Тайл — это создаваемый ассет, который ссылается на исходные спрайты.

ПРАКТИКУМ

Настройка спрайтов и создание тайлов

В этом упражнении мы покажем, как настраивать спрайты и использовать их для создания тайлов. Работайте с проектом, открытым ранее в этом часе, а если вы пропустили практикум «Создание палитры» — сделайте его прямо сейчас. Если вы забыли, как выполнить то или иное действие, можете вернуться к часу 12 и повторить его. Не забудьте сохранить эту сцену, так как она потребуется нам в дальнейшем. Выполните следующие действия.

1. Откройте сцену, созданную в практикуме «Создание палитры». Создайте новую папку и назовите ее **Sprites**. Найдите два спрайта **GrassPlatform_TileSet** и **Jungle_Tileset** в приложенных к книге файлах примеров к этому часу. Перетащите их во вновь созданную папку.
2. Выберите спрайт **GrassPlatform_Tileset** на панели **Project** и посмотрите на его свойства на панели **Inspector**. Задайте свойству **Sprite Mode** значение **Multiple**, а свойству **Pixerls per Unit** — значение **64**. Нажмите кнопку **Apply**.
3. Откройте панель **Sprite Editor** и нажмите кнопку **Slice** в верхнем левом углу. Задайте свойству **Type** значение **Grid By Cell Size**, а свойствам **Pixel Size X** и **Pixel Size Y** значение **64** (рис. 13.7).
4. Нажмите кнопки **Slice** и **Apply**. Затем закройте панель **Sprite Editor**.

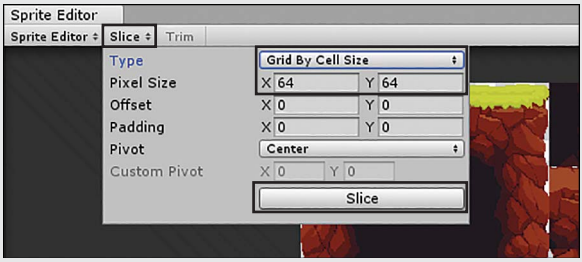


Рис. 13.7. Готовый лист спрайтов после нарезки

5. Повторите шаги со 2 по 4 для спрайта **Jungle_Tileset**. Он немного больше, поэтому задайте свойствам **Pixel Size X** и **Pixel Size Y** значение **128**.
6. Убедитесь в том, что панель **Tile Palette** открыта и прикреплена к панели **Inspector**. Также убедитесь, что выбрана палитра **Grass Tiles**. Перетащите спрайт **GrassPlatform_Tileset** в центр панели **Tile Palette** (см. рис. 13.8).

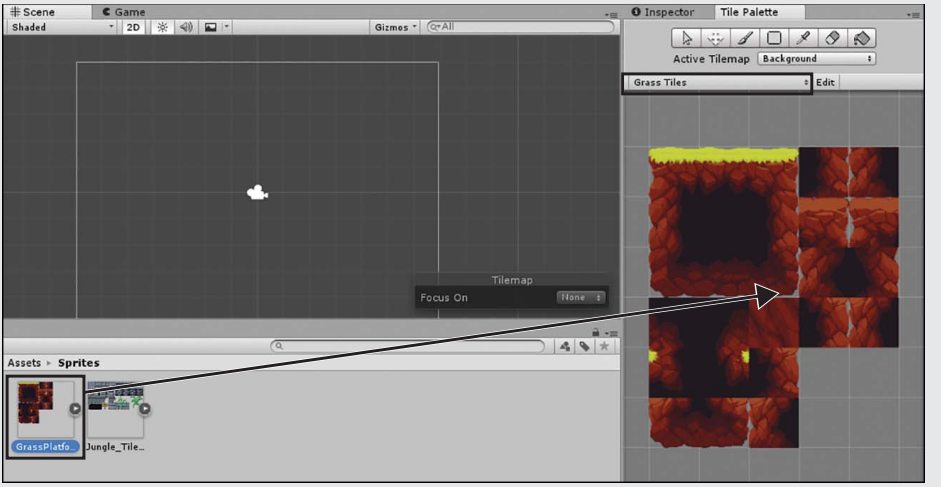


Рис. 13.8. Превращение спрайта в тайлы

7. В появившемся окне создайте новую папку, назовите ее **Tiles**, а затем нажмите кнопку **Select Tiles**.
8. На панели **Tile Palette** выберите палитру **Jungle Tiles**, а затем повторите шаги 6 и 7 для спрайта **Jungle_Tileset**.

Теперь, когда все спрайты настроены и тайлы созданы, настало время рисовать!

Нанесение тайлов

При нанесении тайлов на тайлмап вам нужно обратить внимание на три момента: выбранный тайл, активный тайлмап и выбранный инструмент (рис. 13.9). При выборе тайла, которым вы планируете рисовать, вы можете щелкнуть мышью по одному тайлу или взять сразу несколько. Это бывает полезно, когда вы планируете наносить тайлы, которые должны быть расположены вместе (например, сложные компоненты крыши).

Когда будете готовы, активируйте панель **Scene**, чтобы начать наносить тайл на тайлмап. В таблице 13.1 перечислены инструменты, которые доступны на панели **Tile Palette** (слева направо).

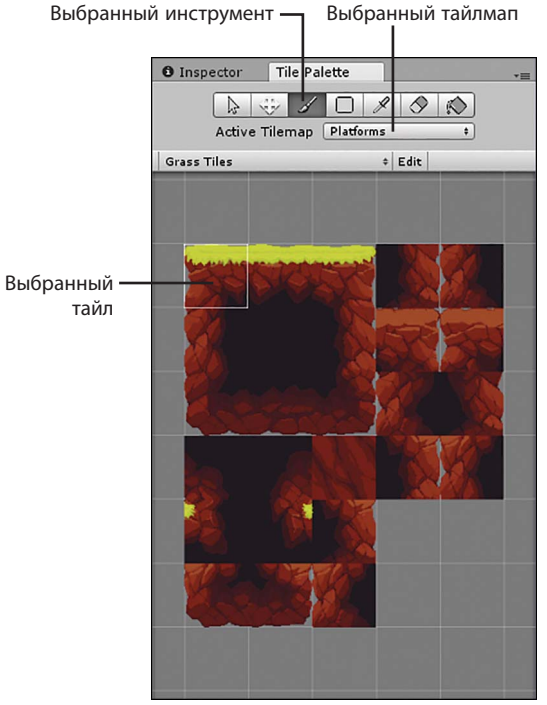


Рис. 13.9. Подготовка к рисованию

ТАБЛ. 13.1. Инструменты на панели Tile Palette

Инструмент	Описание
Select	Используется для выбора тайла или группы тайлов для нанесения
Move	Используется для перемещения выделенной области на тайлмпапе из одного места в другое
Paint	Наносит выбранный тайл (на палитре) на активный тайлмап. Путем перетаскивания можно наносить несколько тайлов одновременно. Удерживая клавишу **Shift** во время рисования, вы переключаетесь на инструмент **Erase**. Удерживая клавишу **Ctrl** (или клавишу ⌘ в macOS) во время рисования, вы можете переключиться на инструмент **Picker**
Rectangle	Используется для рисования прямоугольной формы на тайлмапе и заполняет ее выбранным в данный момент тайлом
Picker	Позволяет выбрать тайл на тайлмапе (вместо того чтобы выбирать его на палитре) для дальнейшего рисования и ускорить рисование при использовании повторяющихся групп тайлов

ТАБЛ. 13.1. Инструменты на панели Tile Palette. *Продолжение*

Инструмент	Описание
Erase	Позволяет стереть тайл или группу тайлов с тайлмапа
Fill	Заполняет область выбранным тайлом

ПРАКТИКУМ

Рисуем тайлами

Пора начать рисовать. Нам понадобится проект, созданный в этом часе ранее, поэтому, если вы еще не выполнили предыдущие практикумы, следует сделать их сейчас. Не забудьте сохранить сцену, так как она потребуется нам в дальнейшем. Выполните следующие действия.

1. Откройте сцену, созданную в практикуме «Настройка спрайтов и создание тайла».
2. Открыв панель **Tile Palette**, выберите палитру **Fungle Tiles** и убедитесь, что выбран тайлмап **Background**.
3. Выбирайте тайлы и наносите их на сцену (рис. 13.10). Выпустите на волю свою фантазию и сделайте такой рисунок, какой вам нравится.

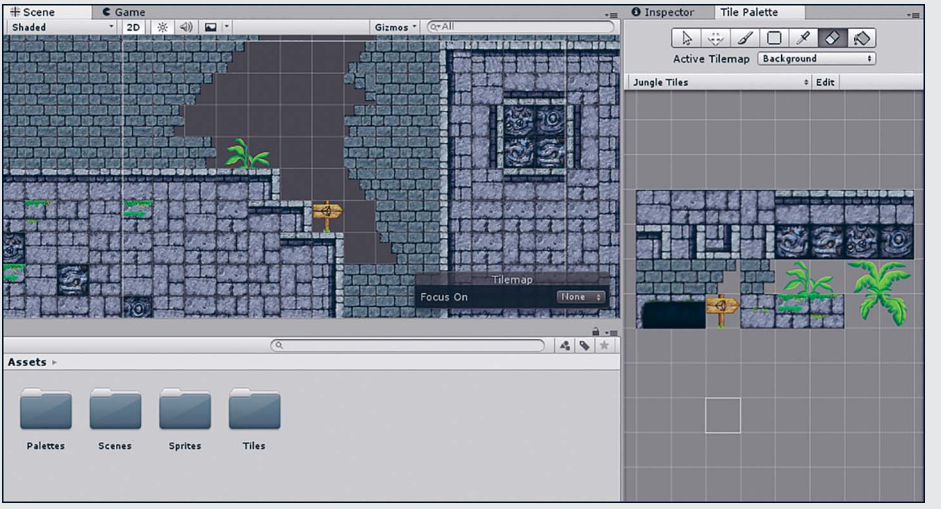


Рис. 13.10. Создание фона

4. Переключитесь на палитру **Grass Tiles** и измените активный тайлмап на **Platforms**.
5. Раскрасьте на уровне парочку платформ. Мы будем использовать их с помощью контроллера персонажа в ближайшее время, так что не забудьте сделать что-нибудь, на что игрок мог бы запрыгивать (рис. 13.11).

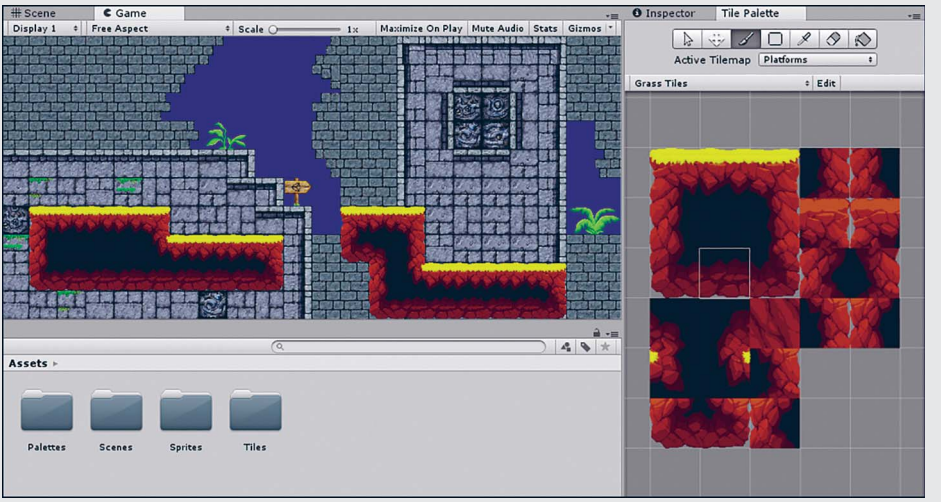


Рис. 13.11. Готовый уровень

СОВЕТ

Продвинутые элементы управления

В поиске и нанесении нужных тайлов вам поможет несколько горячих клавиш. Во-первых, навигация по панели **Tile Palette** осуществляется так же, как и по 2D-сцене. Это означает, что вы можете использовать колесо прокрутки для приближения, а правую кнопку мыши и перетаскивание для перемещения. Во-вторых, при нанесении тайлов вы можете вращать и переворачивать их для создания более интересного вида. Используйте клавиши, (запятая) и **.** (точка), чтобы повернуть тайл перед покраской. Аналогично, сочетанием клавиш **Shift+,** можно перевернуть тайл по горизонтали, а **Shift+.** — по вертикали.

Настройка палитр

Вы, возможно, заметили, что палитры расположены не совсем удобно. Когда вы создаете тайл, перетаскивая спрайт на палитру, Unity оставляет его прямо там, где вы отпустили кнопку мыши. К счастью, вы можете настроить палитру, как вам будет удобнее. Для этого нажмите кнопку **Edit** на панели **Tile Palette** (см. рис. 13.12) и используйте инструменты палитры для рисования, перемещения или замены тайлов, как вам нравится. Вы даже можете создать несколько копий одного и того же элемента, повернуть их или зеркально отразить для удобства нанесения.

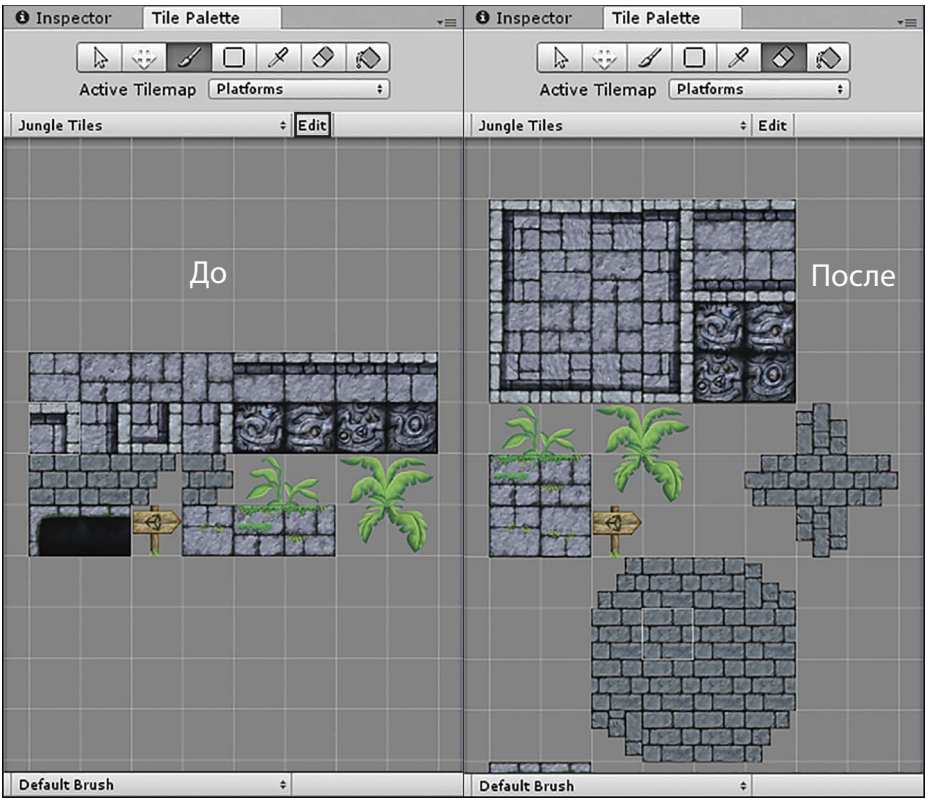


Рис. 13.12. Редактирование палитры

Тайлмапы и физика

Теперь вы научились рисовать на тайлмапах и создавать совершенно новые конструкции на 2D-уровнях. Однако, если вы попробуете поиграть на таком уровне, то обнаружите, что ваши персонажи проваливаются сквозь пол. Пора узнать, как использовать коллайдеры в сочетании с тайлмапами.

Коллайдеры тайлмапов

Вы можете добавить на уровень столкновение, вручную разместив коллайдеры вокруг тайлов, но кому нужные такие трудности? Нам — точно нет! Поэтому лучше использовать компонент **Tilemap Collider 2D** для автоматической обработки столкновения. Эти коллайдеры функционируют так же, как и любые другие, которые вы использовали ранее, хотя и предназначены отдельно для тайлмапов. Все, что вам нужно сделать, это выделить тайлмап, который вы хотите наделить этим свойством, и выбрать команду **Component** ⇒ **Tilemap** ⇒ **Tilemap Collider 2D**, чтобы добавить коллайдер.

ПРАКТИКУМ

Добавление компонента Tilemap Collider 2D

Теперь мы завершим сцену, над которой работали на протяжении этого часа, добавив в платформы коллайдеры. Нам понадобится проект, созданный ранее, поэтому, если вы еще не выполнили предыдущие практикумы, следует сделать это сейчас. Выполните следующие действия.

1. Откройте сцену, созданную в практикуме «Рисуем тайлами». Импортируйте пакет стандартных 2D-ассетов (выбрав команду меню **Assets** ⇒ **Import Package** ⇒ **2D**).
2. Найдите префаб **CharacterRobotBoy** (в папке **Assets\Standard Assets\2D\Prefabs**) и перетащите его на сцену, поместив над одной из ваших платформ. Вам может понадобиться изменить масштаб префаба на (1, 1, 1).
3. Запустите сцену и обратите внимание, что робот падает сквозь платформу. Выключите сцену.
4. Выберите тайлмап **Platforms** и добавьте коллайдер (командой **Add Component** ⇒ **Tilemap** ⇒ **Tilemap Collider 2D**). Обратите внимание, что коллайдер помещается вокруг каждого отдельно взятого тайла (рис. 13.13).

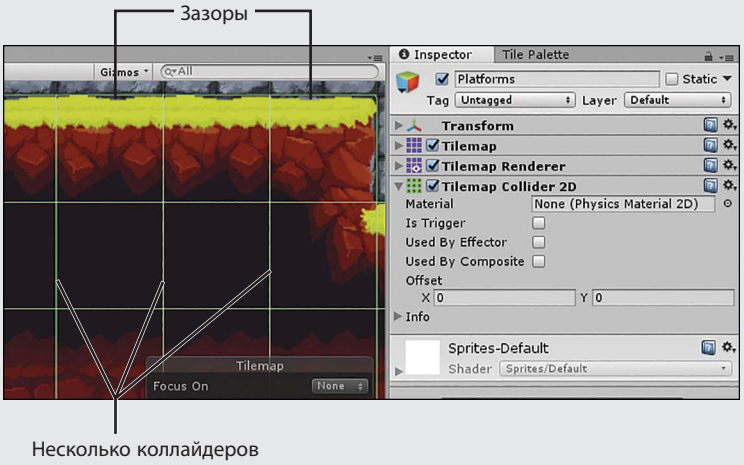


Рис. 13.13. Компонент **Tilemap Collider 2D**

5. Перейдите в режим воспроизведения и обратите внимание, что теперь робот приземляется на платформу. Если увеличить масштаб, вы заметите, что появились зазоры между участками травы и коллайдером (рис. 13.13).
6. Для компонента **Tilemap Collider 2D** присвойте параметру **Y Offset** значение **–0,1**, чтобы слегка опустить коллайдер. Запустив сцену снова, вы увидите, что робот стоит на траве.

Коллайдеры на тайлах делают уровень завершенным и играбельным. Однако стоит отметить, что размещение коллайдера вокруг каждого тайла крайне

неэффективно и может снизить производительность. Вы можете решить эту проблему, используя компонент **Composite Collider 2D**.

СОВЕТ

Точность столкновения

Когда вы начнете использовать коллайдеры с тайлмапами, вам, возможно, потребуется внести некоторые изменения в компоненты **Rigidbody** ваших подвижных объектов. Поскольку края компонентов **Tilemap Collider 2D** и даже **Composite Collider 2D** очень тонкие, вы можете заметить, что объекты, которые имеют небольшие коллайдеры или быстро движутся, могут застревать или даже проваливаться сквозь них. Если вы обнаружили такое поведение, нужно присвоить свойству **Collision Detection** компонента **RigidBody** значение **Continuous**. Это должно предотвратить провалы коллайдера в тайлмап.

Использование компонента Composite Collider 2D

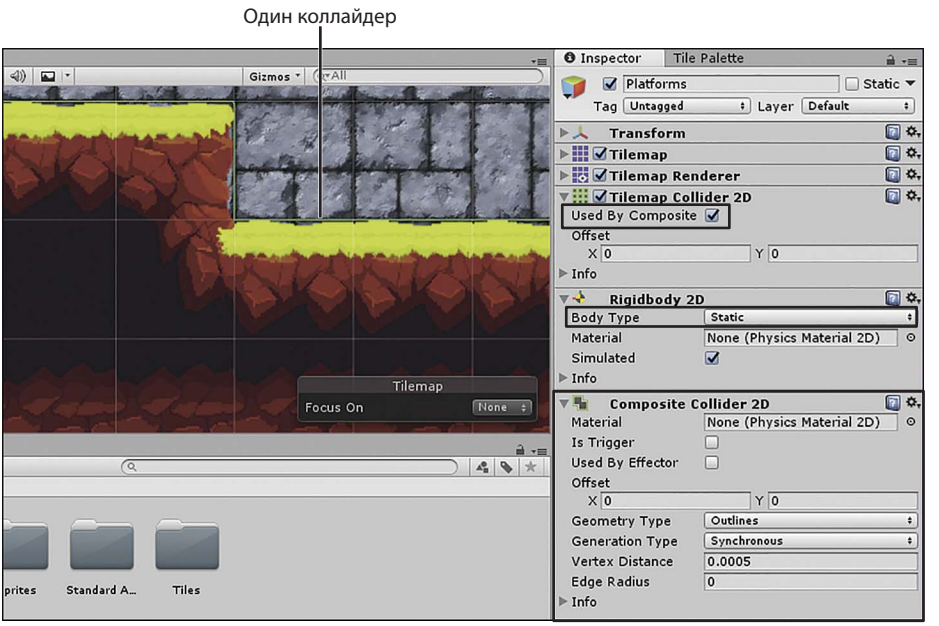


Рис. 13.14. Компонент **Composite Collider 2D**

Комбинированным считается коллайдер, который состоит из нескольких коллайдеров. Он позволяет объединить все отдельно взятые коллайдеры тайлов в один большой. Круче всего то, что всякий раз, когда вы добавляете или изменяете тайлы, коллайдер подстраивается автоматически. Вы можете добавить компонент

Composite Collider 2D, выбрав команду **Add Component** ⇒ **Physics 2D** ⇒ **Composite Collider 2D**. При этом также добавляется компонент **Rigidbody 2D**, так как он требуется для работы компонента **Composite Collider 2D** (рис. 13.14). Очевидно, что, если вы не хотите, чтобы все ваши тайлы падали под воздействием силы тяжести, нужно присвоить свойству **Body Type** компонента **Rigidbody 2D** значение **Static**.

После добавления комбинированного коллайдера в тайлмапе, по сути, ничего не меняется. У каждого тайла остается свой отдельный коллайдер. Поэтому необходимо сообщить данному коллайдеру, что он должен быть использован в комбинированном варианте. Чтобы сделать это, установите флажок **Used By Composite**. После этого все коллайдеры объединятся в большой (и более эффективный) комбинированный коллайдер.

Резюме

В этом часе вы научились создавать 2D-миры с помощью системы 2D-тайлмапов в Unity. Мы начали с понятия о том, что такое тайлмапы. Затем мы создали палитры и настроили спрайты, которые будут добавлены в виде тайлов. Затем мы нарисовали пару тайлмапов и создали уровень. И наконец, вы научились добавлять тайлмапам коллайдеры, чтобы уровень стал играбельным.

Вопросы и ответы

Вопрос: **Может ли тайлмап сочетаться с обычными спрайтами при построении 2D-миров?**

Ответ: Да, конечно. Тайл — это лишь особый вид спрайта.

Вопрос: **Существуют ли какие-либо виды уровней, на которых использовать тайлмапы неуместно?**

Ответ: Тайлмапы идеально подходят для повторяющихся и модульных уровней. А вот сцены, в которых содержится много различных форм или уникальных и неповторяющихся спрайтов, создать с помощью тайлмапов будет трудно.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Какой компонент определяет свойства (например, размер ячейки), которые применяются ко всем тайлмапам?
2. Где размещаются тайлы перед нанесением на тайлмап?
3. Какой тип коллайдера позволяет объединить несколько коллайдеров в один?

Ответы

1. Компонент **Grid**.
2. На палитре.
3. Компонент **Composite Collider 2D**.

Упражнение

В этом упражнении мы поэкспериментируем с созданными тайлмапами, чтобы улучшить их внешний вид и повысить удобство использования. Можно сделать следующее.

1. Попробуйте полностью раскрасить оба ваших тайлмапа, чтобы получить такой вид, какой вы хотите.
2. Попробуйте добавить тайлмап **Foreground**, чтобы на сцене появилось больше растений или скал.
3. Протестируйте готовый уровень с помощью 2D-персонажа и настройте камеру, чтобы следить за персонажем. В файлах примеров к книге есть скрипт *CameraFollow.cs*, настраивающий камеру на слежку за игроком.
4. Попробуйте отдалить тайлмап **Background** от камеры по оси *z*. Таким образом вы сможете создать естественный эффект параллакса. Помните, что он будет заметен только при использовании перспективной камеры.
5. Попробуйте изменить цвет фонового тайлмапа, чтобы фоновые изображения выглядели более бледными и далекими.

14-Й ЧАС

Пользовательские интерфейсы

Что вы узнаете в этом часе

- Какие существуют элементы пользовательского интерфейса (UI)
- Как описываются элементы пользовательского интерфейса
- Различные режимы визуализации пользовательского интерфейса
- Как создать простое меню

Пользовательский интерфейс (User Interface, сокращенно — UI) — это специальный набор компонентов, используемых для вывода информации пользователю и для получения информации от него. В этом часе вы узнаете все об использовании системы пользовательского интерфейса, встроенной в движок Unity. Мы начнем с изучения базовой информации о пользовательских интерфейсах. Затем мы научимся пользоваться различными их элементами, такими как текст, изображения, кнопки и многими другими. В конце урока мы создадим простое, но полноценное меню для ваших игр.

Основные принципы работы с пользовательскими интерфейсами

Пользовательский интерфейс (обычно его называют кратко UI) — это специальный слой, предназначенный, чтобы выводить пользователю информацию и принимать какую-либо информацию от него. Эта информация и входные данные могут быть реализованы в виде HUD-элементов (отображающихся в процессе игры), нарисованных поверх вашей игры, или в виде каких-либо объектов в созданном вами 3D-мире.

В редакторе Unity пользовательский интерфейс реализуется в виде холста, на который наносятся все элементы. Холст должен быть родительским компонентом

по отношению ко всем объектам пользовательского интерфейса, чтобы все работало как надо, и именно холст является основным объектом всего UI.

СОВЕТ

Разработка пользовательского интерфейса

Как правило, эскиз вашего UI нужно продумывать заранее. Надо как следует спланировать, что именно должно отображаться на экране, что где будет находиться и как выглядеть в целом. Избыток информации создает впечатление захламленности на экране, а недостаток вселяет в игрока неуверенность. Всегда нужно искать способы грамотно расположить информацию и сделать ее более значимой. Ваши игроки будут вам благодарны.

СОВЕТ

Новый пользовательский интерфейс

Начиная с версии 4.6 у программы Unity имеется новый пользовательский интерфейс. В старых версиях для создания пользовательских интерфейсов нужно было писать много строк сложного кода, но теперь все стало гораздо проще. Тем не менее старая система UI сохранена. Если вы умеете работать с традиционной системой, вы, возможно, захотите использовать именно ее. Однако не стоит этого делать. Старая система пока присутствует исключительно для отладки, обратной совместимости со старыми проектами, а также расширений редактора. Но она и близко не такая эффективная и мощная, как новая система!

Холст

Холст — основной элемент UI, и все элементы пользовательского интерфейса находятся на нем. Все элементы интерфейса, добавляемые на сцену, будут дочерними объектами холста на панели **Hierarchy** и должны оставаться таковыми — в противном случае они исчезнут со сцены.

Добавить на сцену холст очень легко, выбрав команду **GameObject** ⇒ **UI** ⇒ **Canvas**. После этого можно создавать остальную часть пользовательского интерфейса.

ПРИМЕЧАНИЕ

Объект EventSystem

Вы наверняка заметили, что, когда добавили на сцену холст, появился игровой объект **EventSystem** (система событий). Он всегда добавляется при создании холста. Система событий предназначена, чтобы игрок мог взаимодействовать с пользовательским интерфейсом при помощи кнопок или путем перетаскивания элементов. Без нее интерфейс не будет знать, что он вообще используется, поэтому не удаляйте объект **EventSystem**!

ПРАКТИКУМ

Добавление холста

Выполните следующие действия, чтобы добавить на сцену холст и изучить его уникальные особенности.

1. Создайте новый проект (2D или 3D).
2. Добавьте холст UI на сцену (выбрав команду **GameObject** ⇒ **UI** ⇒ **Canvas**).
3. Отдалите камеру, чтобы увидеть весь холст (дважды щелкнув по нему на панели **Hierarchy**). Обратите внимание, насколько он велик!
4. Обратите внимание на странный компонент преобразования на панели **Inspector**. Это **Rect Transform**, и мы поговорим о нем в ближайшее время.

ВНИМАНИЕ!

Проблемы с производительностью

Холсты очень эффективны, поскольку они превращают элементы пользовательского интерфейса, вложенные в них, в один статичный объект. Это позволяет серьезно ускорить их обработку. Недостаток заключается в том, что, когда изменяется одна часть UI, перестраивается весь интерфейс. Этот процесс может быть очень медленным и неэффективным и вызывать серьезные «тормоза» в игре. Использование холста весьма полезно для отделения объектов, движущихся с высокой скоростью, от остальных. Тогда при их движении будет перестраиваться меньшая часть пользовательского интерфейса, и процесс станет выполняться намного быстрее.

Компонент Rect Transform

Вы уже увидели, что у холста (и любого другого элемента UI) есть компонент **Rect Transform**, а не обычный компонент 3D-преобразования, к которому вы привыкли. Компонент **Rect** (сокращение от *rectangle* — прямоугольник) предоставляет фантастические возможности управлять расположением и изменять масштаб элементов пользовательского интерфейса, при этом не в ущерб гибкости. Он помогает создать пользовательский интерфейс, который будет хорошо работать на самых разных устройствах.

У холста, который вы создали ранее в этом часе, компонент **Rect Transform** полностью неактивен (см. рис. 14.1). Сейчас он принимает значения с панели **Game** (и, соответственно, разрешение и соотношение сторон каких-либо устройств, на которых работает игра). Это означает, что холст всегда будет занимать весь экран. При работе обязательно нужно выбрать соотношение сторон, соответствующее экрану, на котором планируется использовать ваш интерфейс. Вы можете сделать это с помощью раскрывающегося списка **Aspect Ratio** на панели **Game** (см. рис. 14.2).

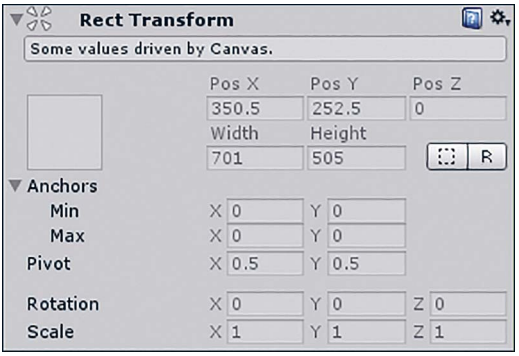


Рис. 14.1. Компонент холста **Rect Transform**

Рис. 14.2. Настройка соотношения сторон для игры

Компонент **Rect Transform** работает немного не так, как привычные нам преобразования. У обычных 2D- и 3D-объектов преобразование определяет, как далеко находится (и как повернут) игровой объект от глобального начала координат. UI вообще не зависит от начала координат. Он должен знать, как он располагается по отношению к его *якорю*. (Вы узнаете больше о **Rect Transform** и якорях позже, когда у вас появится первый элемент пользовательского интерфейса, для которого это будет актуально).

Якорь

Главное в создании элементов пользовательского интерфейса — это *якоря* (якорные точки, точки привязки). Каждый элемент имеет якорь, определяющий его глобальное положение по отношению к **Rect Transform** родительского объекта. Якоря определяют, каким образом изменяются размеры и положение элементов при изменении размера и формы игрового окна. Кроме того, у якоря есть два «режима»: совместный и раздельный. Когда якорь находится в совместном режиме в одной точке, объект узнает его положение за счет определения расстояния (в пикселях) от его оси вращения до якоря. Когда якорь разделен, элемент пользовательского интерфейса устанавливает свою ограничительную рамку

в зависимости от того, как далеко (опять же, в пикселях), каждый из его углов находится от углов разделенного якоря. Трудновато для осознания, да? Давайте попробуем на практике!

ПРАКТИКУМ

Использование компонента Rect Transform

Усвоить принципы работы компонентов **Rect Transform** и якорей сразу может быть сложновато, поэтому выполните следующие шаги, чтобы лучше понять их.

1. Создайте новую сцену или проект.
2. Добавьте в интерфейс изображение (выбрав команду **GameObject** ⇒ **UI** ⇒ **Image**). Обратите внимание, что, если добавить изображение на сцену без холста, Unity автоматически поместит на сцену холст, а затем разместит на нем изображение.
3. Отдалите камеру, чтобы видеть все изображение и холст. Обратите внимание, что работать с интерфейсом гораздо проще, используя инструмент **Rect** (клавиша **T**), и если ваша сцена находится в 2D-режиме (в который вы можете перейти, нажав кнопку **2D** в верхней части панели **Scene**).
4. Попробуйте подвигать изображение по холсту. Также попробуйте подвигать якорь. Обратите внимание, что появляются линии, показывающие, как далеко осевая точка изображения находится от якоря. Также обратите внимание на свойства компонента **Rect Transform** на панели **Inspector** и на то, как они изменяются (рис. 14.3).

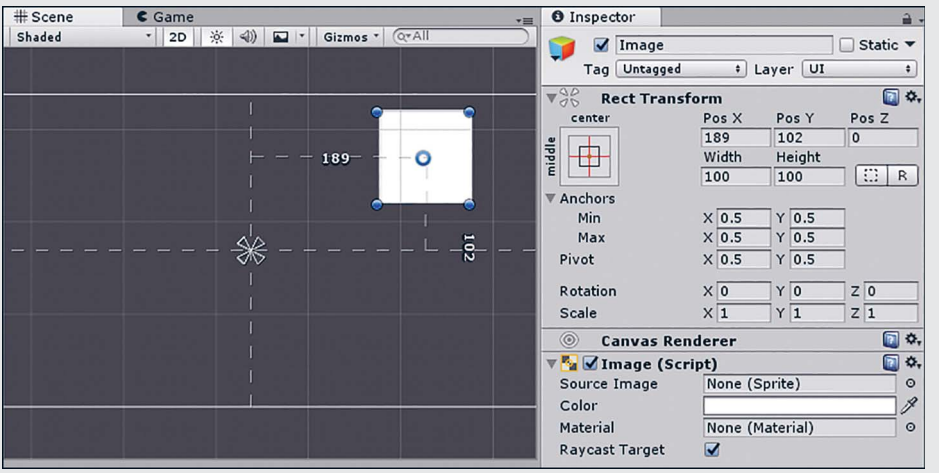


Рис. 14.3. Якорь в одной позиции

5. Теперь разделите ваш якорь. Вы можете сделать это путем перетаскивания любого из углов якоря в сторону от остальных углов. Разделив якорь, снова подвигайте

изображение. Обратите внимание, как изменяются свойства компонента **Rect Transform** (рис. 14.4). (Подсказка: куда делись свойства **Pos X**, **Pos Y**, **Width** и **Height**?)

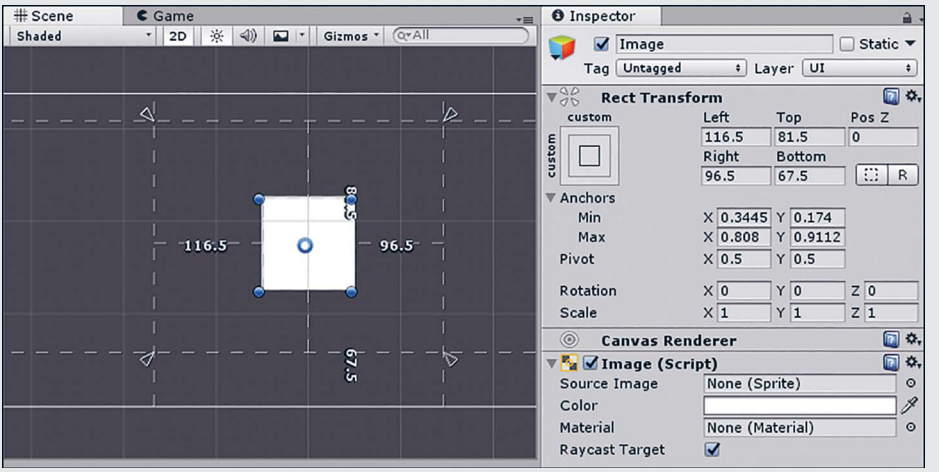


Рис. 14.4. Якорь после разделения

Так в чем же преимущество разделения (или объединения) якоря? Говоря простым языком, расположенный в одной позиции якорь фиксирует элемент пользовательского интерфейса в некотором месте относительно этой позиции. Таким образом, если холст изменяет размер, то элемент не меняется. Разделенный якорь фиксирует углы элемента относительно своих углов. Если холст изменяет размер, с элементом происходит то же самое. Вы можете легко проверить это в редакторе Unity. Используя предыдущий пример, выберите изображение, а затем перетаскиванием отодвиньте границу холста (или любого другого родительского объекта, если у вас их несколько). Появится слово **Preview**, и вы увидите, что будет происходить при использовании различных разрешений (рис. 14.5). Попробуйте сделать это с целым и разделенным якорями и обратите внимание на разницу в поведении.

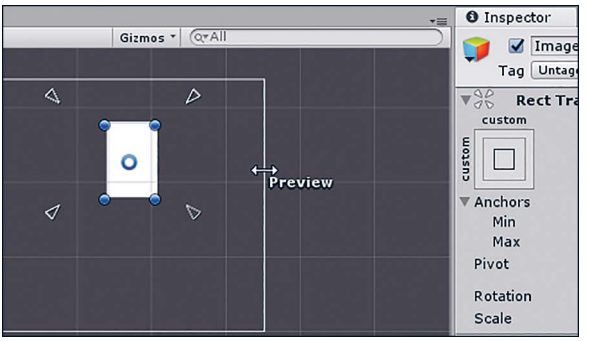


Рис. 14.5. Предварительный просмотр изменений холста

СОВЕТ

Опыт работы с якорями

Поначалу работа с якорями может показаться немного странной, но изучение ее принципов — ключ к пониманию того, как работает пользовательский интерфейс. Когда вы освоите якоря, все остальное встанет на свои места. При работе с элементами пользовательского интерфейса нужно всегда сначала размещать якорь, а затем объект, поскольку именно объекты привязываются к якорям, а не наоборот.

Когда это войдет у вас в привычку (поместить якорь, поместить объект, поместить якорь, поместить объект и так далее), все станет намного проще. Уделите некоторое время якорям, пока не научитесь с ними работать без запинки.

СОВЕТ

Кнопка якоря

Не всегда требуется вручную перетаскивать якорь по сцене, чтобы разместить его. Можно ввести нужные значения в группе элементов управления **Anchor** на панели **Inspector** (значение 1 соответствует 100%, 0,5–50%, и так далее). Если даже это вам делать лень, вы можете использовать удобную кнопку, которая позволяет поместить якорь (а также задать точку поворота) в одно из 24 предустановленных положений (рис. 14.6). Иногда быть ленивым выгодно!

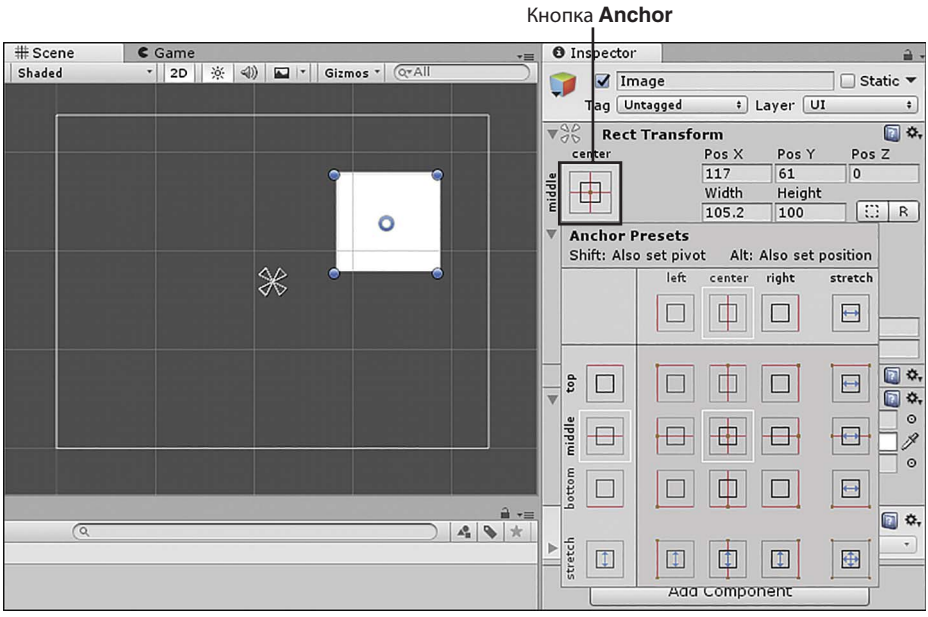


Рис. 14.6. Кнопка настройки якоря

Дополнительные компоненты холста

Мы уже немного поговорили о холсте, но до сих пор даже не упоминали сам компонент **Canvas**. По правде говоря, он не такой уж сложный, и переживать тут не о чем. Вам достаточно знать режимы рендеринга, и мы рассмотрим их подробнее позже в этом часе.

В зависимости от того, какую версию Unity вы используете, у вас может быть некоторое количество дополнительных компонентов. Опять же, они весьма просты в использовании, поэтому мы не будем говорить о них подробно (у нас слишком много других хороших тем для обсуждения). Компонент **Canvas Scaler** (Масштабирование холста) позволяет определить, как ваши элементы пользовательского интерфейса должны (и должны ли вообще) изменять размер и вид в случае изменений целевого устройства (например, отображение одного и того же интерфейса на веб-странице и на устройстве iPad с дисплеем Retina с высокой плотностью пикселей). Компонент **GraphicalRaycaster**, который работает с объектом **EventSystem**, позволяет вашему пользовательскому интерфейсу воспринимать нажатия кнопок и касания экрана. Так вы можете использовать рейкастинг без необходимости подключать к этой задаче целый физический движок.

Элементы пользовательского интерфейса

Вам, наверное, уже надоели разговоры о холсте, поэтому давайте поработаем с некоторыми элементами пользовательского интерфейса (иначе называемыми элементами управления пользовательского интерфейса). В Unity есть несколько встроенных элементов управления, но не беспокойтесь, если не удается найти нужный вам. Библиотека Unity UI имеет открытый исходный код, и сообщество наших пользователей постоянно придумывает и создает новые элементы управления. А вообще, если не боитесь трудностей, вы можете тоже создавать свои элементы управления и делиться ими со всеми.

В Unity есть много элементов управления, которые можно добавлять на сцену. Большинство из них представляют собой простые сочетания и вариации двух основных элементов: изображения и текста. Если подумать, то все логично: панель — это всего лишь полноразмерное изображение, кнопка — изображение с текстом, а полоса прокрутки — три изображения, соединенных вместе. По сути, весь пользовательский интерфейс представляет собой систему базовых элементов, сочетание которых позволяет сформировать нужный функционал.

Изображения

Изображения — основной структурный компонент пользовательского интерфейса. Это могут быть фоновые изображения, кнопки, логотипы, шкалы здоровья, все что угодно. Если вы выполнили приведенные ранее в этом часе практикумы, то вы уже немного знакомы с изображениями, но теперь пора взглянуть на них поближе. Как вы видели ранее, вы можете добавить изображение на холст, выбрав команду **GameObject ⇒ UI ⇒ Image**. В таблице 14.1 перечислены свойства изображений, которые представляют собой игровой объект с компонентом **Image**.

ТАБЛ. 14.1. Свойства компонента Image

Свойство	Описание
Source Image	Определяет изображение, которое будет воспроизведено. Оно должно быть спрайтом. (Более подробную информацию о спрайтах можно получить в часе 12.)
Color	Позволяет задать любой цвет тонировки и непрозрачность для изображения
Material	Указывает материал (если таковые имеются), который будет применен к изображению
Raycast Target	Определяет, кликабельно ли изображение
Image Type	Указывает тип спрайтов, которые будут использоваться. Это свойство определяет масштаб и тайл изображения
Preserve Aspect	Определяет, будет ли изображение сохранять свое исходное соотношение сторон при масштабировании
Set Native Size	Устанавливает размер объекта изображения равным исходному размеру в файле

Помимо основных свойств, больше вам ничего особенного знать не нужно. Проработайте приведенный далее практикум, чтобы увидеть, как легко можно использовать изображения.

ПРАКТИКУМ

Использование изображений

Давайте попробуем создать фоновое изображение. Для этого упражнения понадобится файл *BackgroundSpace.png* из файлов примеров к часу 14. Выполните следующие действия.

1. Создайте новую сцену или проект.
2. Импортируйте файл *BackgroundSpace.png* в свой проект как спрайт. (Вернитесь к часу 12, если не помните, как это делать.)

3. Добавьте на сцену изображение (выбрав команду **GameObject** ⇒ **UI** ⇒ **Image**).
4. Укажите спрайт **BackgroundSpace** в качестве значения свойства **Source Sprite** объекта **Image**.
5. Измените размер изображения, чтобы заполнить весь холст. Переключитесь на панель **Game** и посмотрите, что происходит, если изменить соотношение сторон. Обратите внимание, что изображение либо получается обрезанным, либо не заполняет экран.
6. Разделите якорь изображения так, чтобы четыре угла якоря совпали с четырьмя углами холста (рис. 14.7). Теперь переключитесь обратно в режим игры и посмотрите, что происходит при изменении соотношения сторон. Обратите внимание, что изображение всегда заполняет экран и не обрезается, хотя его размер искажается.

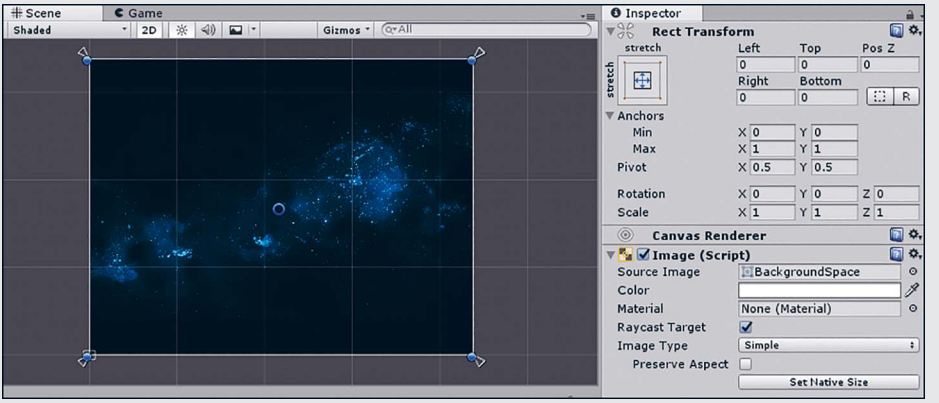


Рис. 14.7. Расширение изображения и якорей

ПРИМЕЧАНИЕ

Материалы пользовательского интерфейса

Как видно на рис. 14.7 и из практикума «Использование изображений», у компонента изображения есть свойство **Material**. Стоит отметить, что свойство **Material** не является обязательным и не требуется для данного интерфейса. Кроме того, в текущем режиме рендеринга холста (более подробно о нем мы поговорим далее в этом часе) свойство **Material** не особенно значимо. Но в других режимах оно позволяет применять эффекты освещения и шейдеров к элементам пользовательского интерфейса.

Текст

Текстовые объекты (на самом деле это обычные текстовые компоненты) используются для вывода текста пользователю. Если вы использовали элементы форматирования текста раньше (например, в ПО для ведения блогов, текстовых

редакторах вроде Word или WordPad и тому подобное), работа с этим компонентом не составит труда. Вы можете добавить компонент **Text** на холст, выбрав команду **GameObject** ⇒ **UI** ⇒ **Text**. В таблице 14.2 перечислены свойства компонента **Text**. Поскольку большинство из них интуитивно понятны, перечислены только новые или специфические свойства.

ТАБЛ. 14.2. Свойства компонента Text

Свойство	Описание
Text	Определяет отображаемый текст
Rich text	Указывает, будут ли использоваться в тексте метки форматирования
Horizontal Overflow и **Vertical Overflow**	Определяет, как редактор выходит из ситуации, когда текст не помещается в пределах ограничительной рамки элемента пользовательского интерфейса, который его содержит. Значение **Wrap** означает, что текст будет переноситься на следующую строку. **Round** означает, что не вмещающийся текст будет удален. **Overflow** означает, что текст может выходить за пределы рамки. Если текст не помещается в рамку и это свойство не установлено, то текст может исчезнуть
Best Fit	В качестве альтернативы свойству **Overflow** (которая, впрочем, не будет работать, если свойство **Overflow** используется) можно изменять размер текста, чтобы он поместился в рамку объекта. Используя свойство **Best Fit**, вы можете выбрать минимальный и максимальный размер. Шрифт будет увеличиваться или уменьшаться в диапазоне между этими двумя пределами, чтобы всегда вписываться в текстовое поле

Обязательно стоит уделить время и опробовать различные настройки, особенно для свойств **Overflow** и **Best Fit**. Если вы не знаете, как использовать эти свойства, вы удивитесь, если текст вдруг неожиданно исчезнет (так как не влез в рамку), и вам потребуется некоторое время, чтобы выяснить почему.

Кнопки

Кнопки — это элементы, которые пользователь нажимает для ввода информации. Работа с ними может показаться на первый взгляд сложной, но помните, что кнопка, как уже упоминалось ранее, это всего лишь изображение с текстом, имеющее чуть больше функциональных возможностей. Вы можете добавить на сцену кнопку, выбрав команду **GameObject** ⇒ **UI** ⇒ **Button**.

Кнопка отличается от других элементов управления только тем, что она интерактивна и поэтому имеет некоторые интересные свойства и особенности. Например, у кнопок могут быть переходы, с помощью кнопок может осуществляться навигация, и у них есть обработчик событий `OnClick()`. В таблице 14.3 перечислены свойства компонента **Button**.

ТАБЛ. 14.3. Свойства компонента Button

Свойство	Описание
Interactable	Указывает, может ли пользователь нажимать на кнопку
Transition	Определяет, как кнопка должна реагировать на действия пользователя (рис. 14.8). Действия, на которые может быть ответ: **Normal** (ничего не происходит), **Highlighted** (когда мышь наведена на кнопку), **Pressed** (нажата) и **Disabled** (неактивна). По умолчанию кнопка меняет цвет (**Colot Tint**). Вы можете также удалить любые переходы (**None**) или изменять изображение кнопки (**Sprite Swap**). Наконец, вы можете выбрать вариант **Animation** и использовать полноценную анимацию, тогда ваши кнопки будут выглядеть очень эффектно
Navigation	Определяет, как пользователь будет перемещаться между кнопками в случае использования контроллера или джойстика (то есть без мыши и сенсорного экрана). Нажатие кнопки **Visualize** (Визуализация) позволяет увидеть, как будет происходить навигация. Это работает только в случае, если на холсте расположено не менее двух кнопок
`OnClick()`	Определяет, что произойдет после нажатия кнопки (обсуждается далее в этом часе

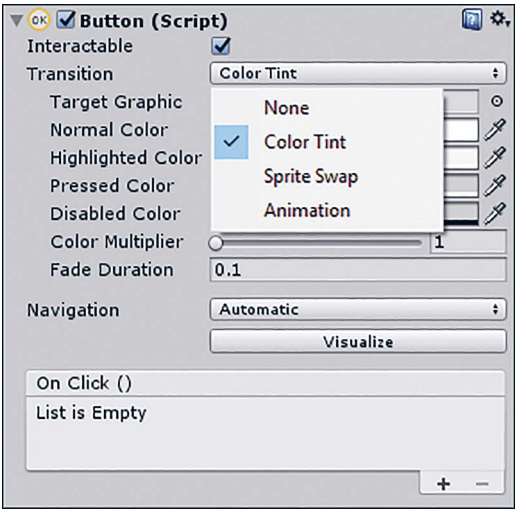


Рис. 14.8. Выбор типа перехода

Событие `OnClick()`

Когда пользователи вдоволь насладятся созданными вами различными эффектами кнопок, они захотят что-нибудь понажимать. Настройте свойство `OnClick()` в нижней части панели **Inspector**, чтобы вызвать функцию из скрипта и получить доступ ко многим другим компонентам. Вы можете передать параметры в любой вызываемый метод, а это означает, что разработчик имеет возможность управлять поведением игры, не вводя код. Более глубокое использование этих функций позволит вызывать методы на игровом объекте или поворачивать камеру на объекты.

ПРАКТИКУМ

Использование кнопок

Выполните следующие действия, чтобы применить ваши новые знания на практике и создать кнопку, добавить ей цветовые переходы и настроить ее так, чтобы текст кнопки изменялся при нажатии.

1. Создайте новую сцену или проект.
2. Добавьте на сцену кнопку (выбрав команду **GameObject** ⇒ **UI** ⇒ **Button**).
3. В области компонента **Button (Script)** на панели **Inspector** задайте свойству **Highlighted Color** красный цвет, а свойству **Pressed Color** — зеленый (см. рис. 14.9).

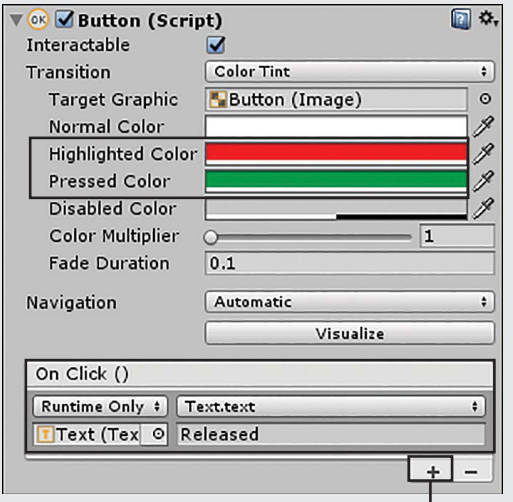


Рис. 14.9. Настройки кнопки

4. Добавьте новый обработчик события `OnClick()` нажатием кнопки **+** в нижней части панели **Inspector** (рис. 14.9). В дефолтном состоянии, когда обработчик ожидает объекта для манипулирования, на нем написано **None (Object)**.
5. Разверните игровой объект кнопки на панели **Hierarchy**, и вы увидите дочерний объект **Text**. Перетащите объект **Text** на свойство **Object** обработчика событий.
6. В раскрывающемся списке функций (на котором написано **No Function**), укажите, что кнопка должна сделать для выбранного объекта. Для этого щелкните по раскрывающемуся списку (❶ на рис. 14.10), а затем выберите команду **Text** (❷) ⇒ **string text** (❸).

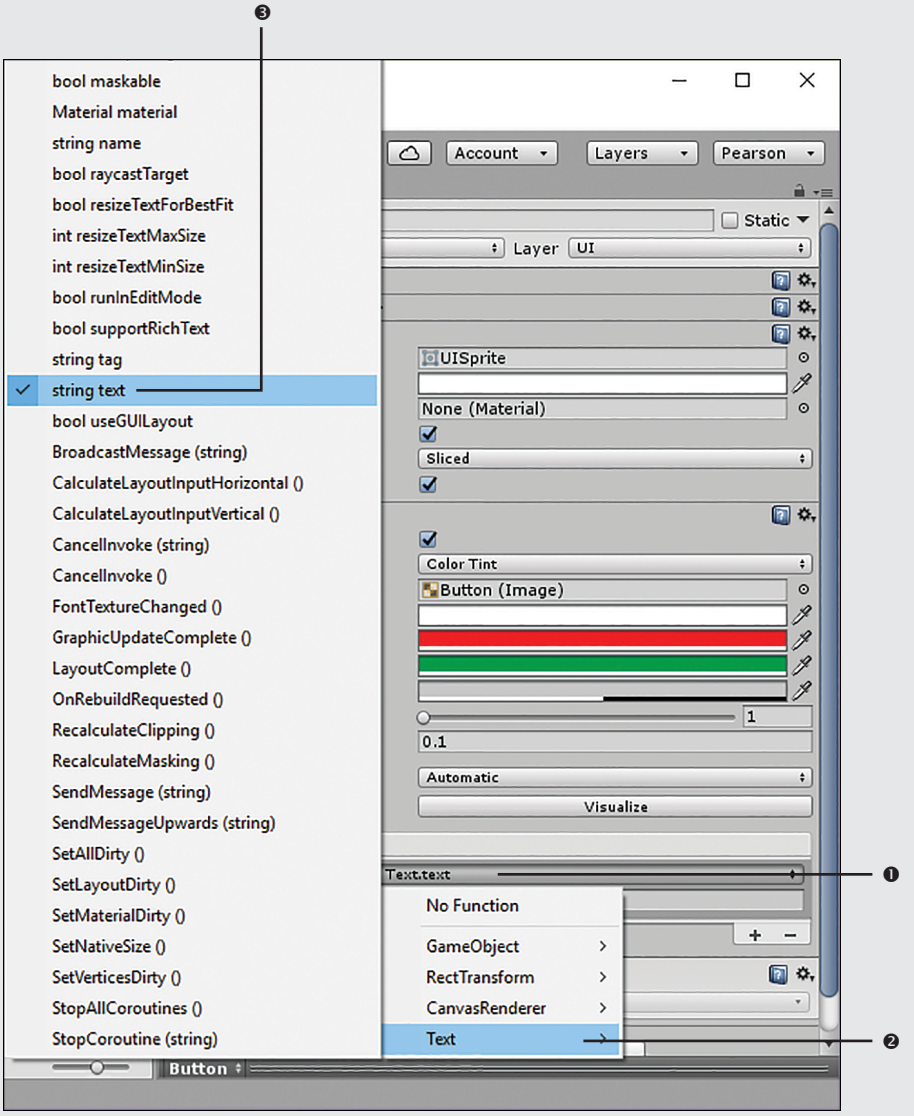


Рис. 14.10. Настройка события `OnClick()` на изменение текста кнопки

7. В появившемся поле введите **Released**.
8. Запустите игру и поводите мышью над кнопкой и понажимайте ее. Обратите внимание на изменение цвета и текста при нажатии кнопки.

СОВЕТ

Сортировка элементов

Теперь, когда мы познакомились с кое-какими элементами, нужно разобраться, как они отрисовываются. Вы, возможно, заметили, что компонент **Canvas**, который мы видели ранее в этом часе, имеет свойство **Sorting Layer** (как 2D-изображения). Это свойство используется только для сортировки нескольких холстов в пределах одной и той же сцены. Для сортировки элементов пользовательского интерфейса на одном холсте используется порядок объектов на панели **Hierarchy**. Поэтому, если вам нужно, чтобы один объект отрисовывался поверх другого, переместите его ниже на панели **Hierarchy**.

СОВЕТ

Пресеты

В Unity 2018.1 были добавлены пресеты компонентов. *Пресеты* — это сохраненные свойства компонентов (например, **UI Text**), которые можно быстро применить к новым компонентам. Меню пресетов находится в правом верхнем углу компонента, рядом с шестеренкой настроек на панели **Inspector**. Пресеты работают для любых компонентов, но мы не говорили о них ранее, так как они особенно хорошо сочетаются с пользовательским интерфейсом. Очень часто они применяются для того, чтобы весь текст в игре был одинаковым. Делать текст префабом не обязательно, а вот использовать пресет текста — вполне можно.

Режимы рендеринга холста

В Unity есть три крутых способа вывести пользовательский интерфейс на экран. Вы можете указать режим на панели **Inspector**, выбрав компонент **Canvas** и свойство **Render Mode**. После этого вы увидите режимы, представленные на рис. 14.11. Использовать все режимы холста сложно, поэтому разбираться с ними всеми прямо сейчас необязательно. Вместо этого прочитайте описание трех режимов (**Screen Space — Overlay**, **Screen Space — Camera** и **World Space**), и вы сможете выбрать наиболее подходящий для вас.

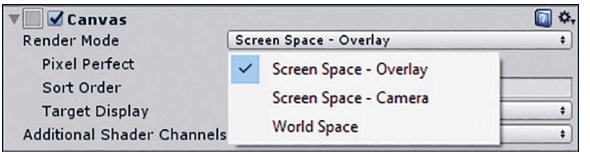


Рис. 14.11. Три различных режима рендеринга холста

Режим Screen Space — Overlay

Режим **Screen Space — Overlay**, который используется по умолчанию, — самый простой, но от этого не менее крутой. Пользовательский интерфейс в режиме **Screen Space — Overlay** отображается поверх всего на экране, независимо от настроек камеры или ее положения в мире. По сути, интерфейс на панели **Scene** не имеет никакого отношения к объектам в мире, так как на самом деле он отрисовывается камерой.

Пользовательский интерфейс отображается на сцене в фиксированном положении, и его левая нижняя точка имеет глобальную координату (0, 0, 0). Масштаб интерфейса отличается от мирового, и то, что вы видите на холсте, имеет масштаб в 1 мировую единицу для каждого пикселя на панели **Game**. Если вы используете этот тип интерфейса в игре, но его положение в мире мешает вам во время работы, вы можете скрыть его. Для этого в раскрывающемся списке **Layers** в редакторе скройте значок глаза рядом со слоем UI (рис. 14.12). Интерфейс исчезнет с панели **Scene** (но появится, когда вы запустите игру). Не забудьте вернуть его, чтобы случайно не потерять!

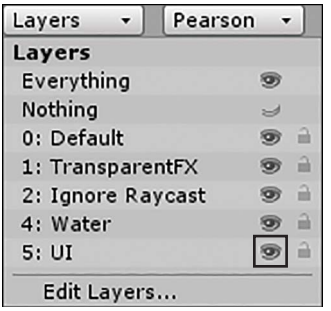


Рис. 14.12. Сокрытие интерфейса

Режим Screen Space — Camera

Режим **Screen Space — Camera** похож на **Screen Space — Overlay**, но интерфейс здесь отрисовывается камерой на ваш выбор. Вы можете вращать и масштабировать элементы, чтобы создавать гораздо более динамичные 3D-интерфейсы.

В отличие от **Screen Space — Overlay**, этот режим использует для рендеринга камеру. Это означает, что на пользовательский интерфейс влияют такие эффекты, как, например, освещение, а между камерой и интерфейсом могут быть объекты. Придется проделать больше работы, но ваш интерфейс будет лучше смотреться в составе мира.

Обратите внимание, что в этом режиме интерфейс остается в фиксированном положении по отношению к камере, которую вы выбрали для отрисовки. При перемещении камеры перемещается и холст. Довольно удобно использовать отдельную камеру исключительно для рендеринга холста (чтобы все происходило в стороне от сцены).

Режим World Space

Последний режим, который следует рассмотреть, — **World Space**. Представьте себе виртуальный музей, где каждый объект, на который вы смотрите, содержит подробные сведения об объекте, находящемся в непосредственной близости от него. Вместе с информацией может отображаться кнопка, позволяющая читать дальше или перейти к другим залам музея. И если вы вообразите нечто подобное, то это будет лишь малая часть того, что вы можете сделать с режимом холста **World Space**.

Обратите внимание, что компонент **Rect Transform** в режиме **World Space** становится активным, а сам компонент **Canvas** можно отредактировать и изменить его размер (рис. 14.13). Поскольку в этом режиме холст является игровым объектом, он не отрисовывается поверх остальной игры, как HUD. Вместо этого он занимает определенное положение в мире и становится его частью.

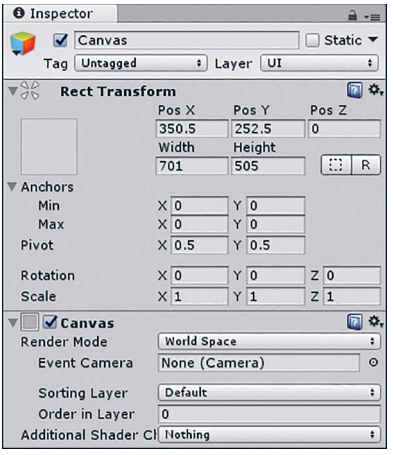


Рис. 14.13. Компонент **Rect Transform** в режиме **World Space**

ПРАКТИКУМ

Изучение режимов рендеринга

Выполните следующие действия, чтобы подробнее изучить три режима рендеринга интерфейса.

1. Создайте новую 3D-сцену или проект.
2. Добавьте на сцену холст UI (выбрав команду **GameObject** ⇒ **UI** ⇒ **Canvas**).
3. Обратите внимание, что компонент **Rect Transform** пока что неактивен. Подвигайтесь по сцене и посмотрите, где находится холст.
4. Переключитесь в режим рендеринга **Screen Space — Camera**. Для свойства **Render Camera** выделите объект **Main Camera**. Обратите внимание, как после этого изменится размер и положение холста.
5. Посмотрите, что произойдет, если вы переместите камеру и если измените свойство **Projection** камеры с **Perspective** на **Orthographic** и обратно.
6. Переключитесь в режим **World Space**. Теперь вы можете изменить преобразование холста, перемещать, вращать и масштабировать его.

Резюме

В начале этого часа мы изучили основные части любого интерфейса: холст и систему событий. Затем вы узнали о компоненте **Rect Transform** и о том, как якоря позволяют создавать универсальные интерфейсы, которые могут работать на многих устройствах. Затем мы изучили различные элементы пользовательского интерфейса, доступные для использования в Unity. Потом мы кратко рассмотрели суть различных режимов рендеринга пользовательского интерфейса: **Screen Space — Overlay**, **Screen Space — Camera** и **World Space**.

Вопросы и ответы

Вопрос: **Требуется ли пользовательский интерфейс в каждой игре?**

Ответ: Обычно хорошо продуманный интерфейс делает игру только лучше. Редки случаи, когда у игры вообще нет интерфейса. Тем не менее интерфейс не должен быть перегружен, его задача — предоставлять игрокам только необходимую информацию и тогда, когда это нужно.

Вопрос: **Можно ли комбинировать разные режимы рендеринга на одной сцене?**

Ответ: Да. На сцене может быть более одного холста с разными режимами рендеринга.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Что означает аббревиатура UI?
2. Какие два игровых объекта в Unity всегда появляются вместе с интерфейсом?
3. Какой режим рендеринга интерфейса вы будете использовать, чтобы разместить знак вопроса над головой игрока в 3D-игре?
4. Какой режим рендеринга лучше всего подойдет для простого HUD-элемента?

Ответы

1. User Interface — пользовательский интерфейс (простенький вопрос для разминки!).
2. Компоненты **Canvas** и **EventSystem**.
3. Следует использовать режим **World Space**, поскольку элемент интерфейса позиционируется в мировом пространстве, а не относительно поля зрения игрока.
4. Режим **Screen Space — Overlay**.

Упражнение

В этом упражнении мы создадим простую, но полноценную систему меню, которую вы можете адаптировать к любой игре. Мы используем и заставки, и затухание, и фоновую музыку, и многое другое.

1. Создайте новый 2D-проект. Добавьте панель UI (выбрав команду **GameObject** ⇒ **UI** ⇒ **Panel**). Обратите внимание, что программа Unity сама добавила необходимые компоненты **Canvas** и **EventSystem** на панель **Hierarchy**.
2. Импортируйте файл *Hour14Package.unitypackage* из приложенных к книге файлов примеров. Выберите файл **clouds.jpg** в папке **Assets** и убедитесь, что свойству **Texture Type** присвоено значение **Sprite (2D and UI)**.
3. Установите это изображение в качестве значения свойства **Source Image** панели. Обратите внимание, что изображение по умолчанию имеет

некоторую прозрачность, чтобы фоновый цвет тоже отображался. Вы можете управлять прозрачностью, открыв окно выбора цвета с ползунковым регулятором **A** (где **A** означает «альфа-канал»).

4. Добавьте заголовок и подзаголовок (выбрав команду **GameObject** ⇒ **UI** ⇒ **Text**). Переместите их так, чтобы они красиво смотрелись на панели (рис. 14.14).

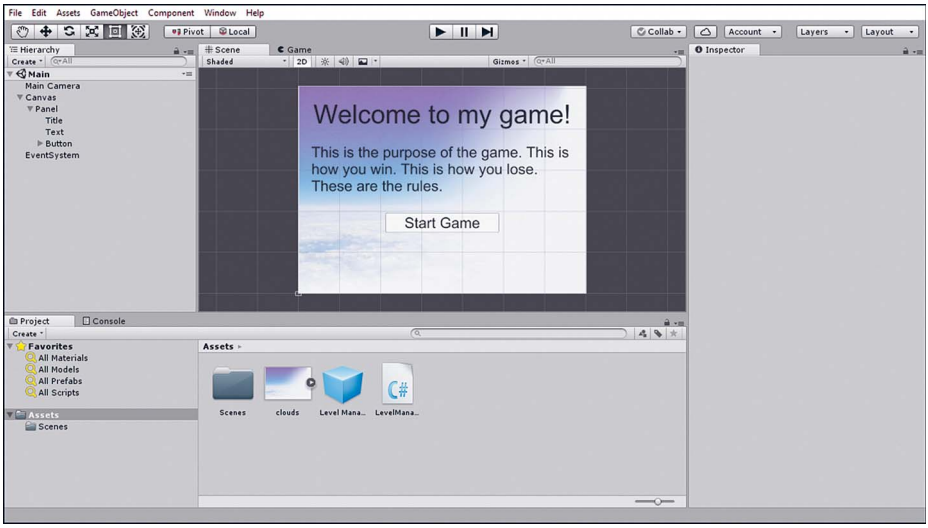


Рис. 14.14. Готовый интерфейс

5. Добавьте кнопку (выбрав команду **GameObject** ⇒ **UI** ⇒ **Button**). Присвойте ей имя **Start**, а свойству **Text** дочернего объекта — значение **Start Game**. Разместите кнопку там, где хотите, не забывая, что для перетаскивания нужно выбрать именно кнопку, а не текст.

6. Сохраните вашу сцену как **Menu** (выбрав команду **File** ⇒ **Save Scene**). Теперь создайте новую сцену, выступающую в качестве заполнителя для вашей игры, и сохраните ее. Наконец, добавьте обе сцены в сборку, открыв параметры сборки (команда **File** ⇒ **Build Settings**) и перетащив обе сцены в раздел **Scenes in Build**. Убедитесь, что сцена с меню находится сверху.

7. Переключитесь обратно на сцену меню. Перетащите импортированный префаб **LevelManager** на панель **Hierarchy**.

8. Найдите кнопку **Start** и присвойте ее свойству `OnClick()` значение в виде метода **`LoadGame()`** менеджера уровня (рис. 14.15).

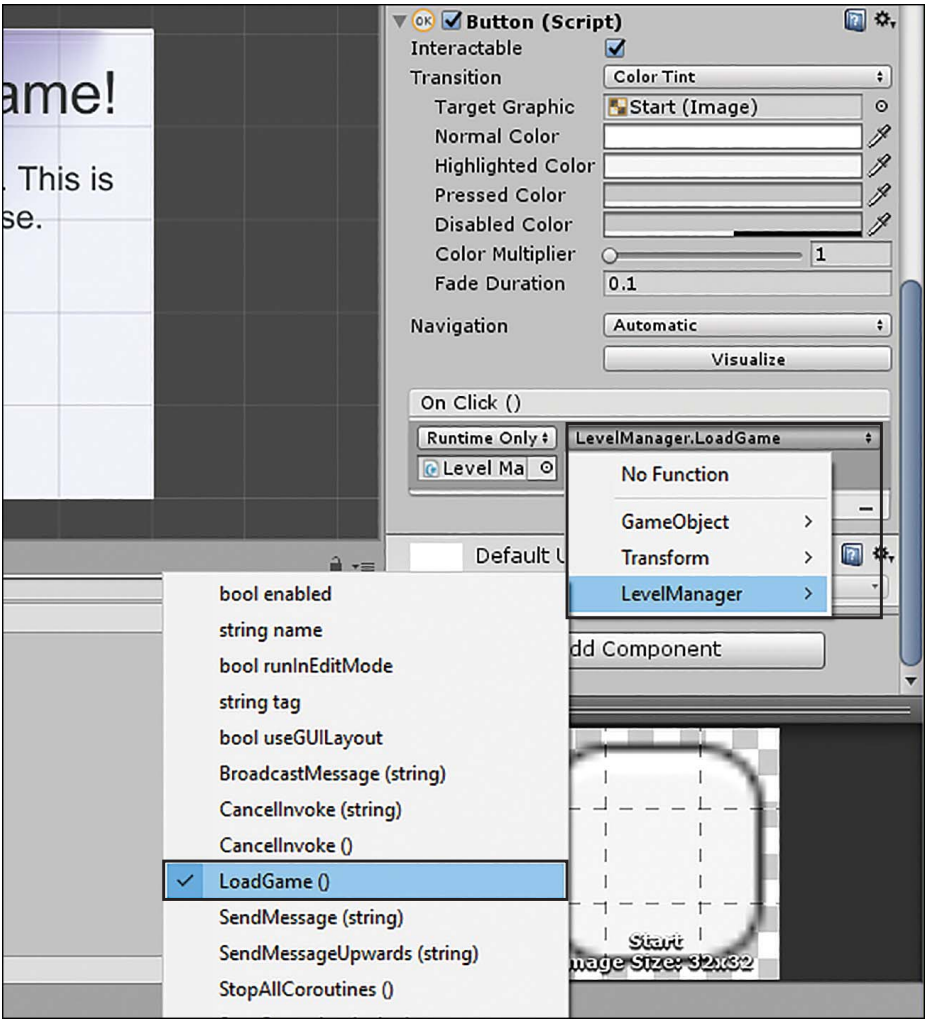


Рис. 14.15. Настройка обработчика `OnClick()` кнопки **Start Game**

9. Запустите сцену. Нажмите кнопку **Start Game**, после чего должна отобразиться пустая сцена с игрой. Поздравляем! Теперь у вас есть система меню, готовая для использования в будущих играх!

15-Й ЧАС

Игра третья: «Капитан Бластер»

Что вы узнаете в этом часе

- Как создать игру «Капитан Бластер»
- Как построить мир для игры «Капитан Бластер»
- Как создать объекты для игры «Капитан Бластер»
- Как создать элементы управления для игры «Капитан Бластер»
- Как еще можно улучшить игру «Капитан Бластер»

Пора бы нам создать игру! В этом часе мы сделаем 2D-скролл-шутер под названием «Капитан Бластер». Мы начнем с разработки различных элементов игры. Затем перейдем к созданию прокручивающегося фона. Когда с движением все будет ясно, мы начнем добавлять игровые объекты. После объектов займемся настройкой управления и игрофикацией проекта. В конце часа мы проанализируем игру и определимся, как можно улучшить ее.

СОВЕТ

Завершенный проект

Обязательно выполните все задания из этого часа, чтобы создать полноценный игровой проект. В случае затруднений вы можете посмотреть копию игры в приложенных к книге файлах примеров для часа 15. Обратитесь к ним, если вам потребуется вдохновение.

Проектирование

Вы изучили основные элементы проекта в часе 6, посвященном игре «Великолепный гонщик». Здесь мы займемся примерно тем же.

Концепция

Как уже упоминалось ранее, «Капитан Бластер» — это 2D-скролл-шутер. Идея заключается в том, что игрок будет летать по уровню, уничтожая метеоры и стараясь не погибнуть. Главная «фишка» 2D-скроллеров в том, что на самом деле игрок никуда не двигается, ощущение движения создает подвижный фон. Это снижает требования к мастерству игрока и позволяет устанавливать более сложные препятствия — например, врагов.

Правила

Правила игры определяют, как в нее играть, а также намекают на некоторые свойства объектов. Правила игры «Капитан Бластер» следующие.

- Игрок играет до столкновения с метеором. Условия победы не предусмотрены.
- Игрок может стрелять снарядами, тем самым уничтожая метеоры. Игрок получает одно очко за каждый уничтоженный метеор.
- Игрок может выстреливать до двух снарядов в секунду.
- Движение игрока ограничено экраном.
- Метеоры появляются непрерывно, пока игрок не проиграет.

Требования

Требования, предъявляемые к этой игре, весьма просты.

- Фоновый спрайт, который изображает космическое пространство.
- Спрайт корабля.
- Спрайт метеора.
- Менеджер игры, которого мы создадим в Unity.
- Интерактивные скрипты. Их мы напишем в Visual Studio, как обычно.

Создание игрового мира

Поскольку действие происходит в космосе, создать мир довольно просто. Игра будет двумерной, а фон станет двигаться в вертикальном направлении позади игрока, чтобы казалось, что игрок движется вперед. На самом-то деле игрок будет неподвижен. Но перед тем как реализовать скроллинг на месте, нужно настроить ваш проект. Начнем вот с чего.

1. Создайте новый 2D-проект под названием «Капитан Бластер».

2. Создайте папку с именем **Scenes**, в которую сохраните сцену под названием **Main**.

3. На панели **Game** задайте соотношение сторон **5:4** (рис. **15.1**).

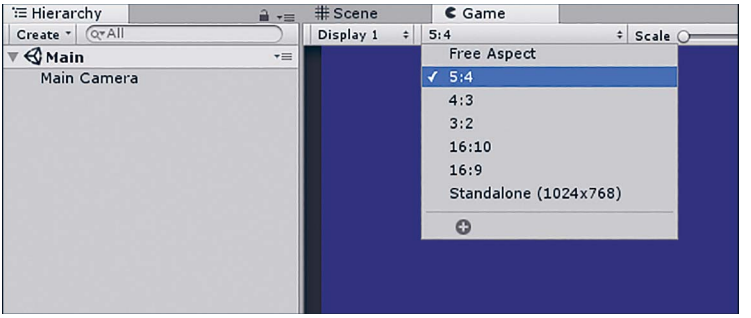


Рис. 15.1. Настройка соотношения сторон для игры

Камера

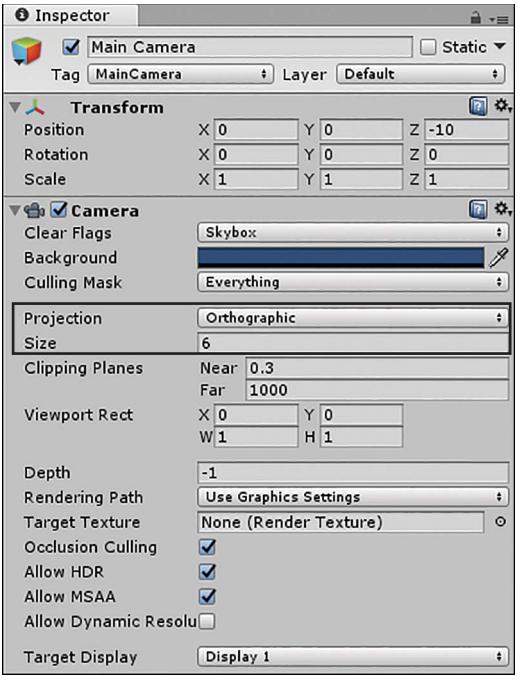


Рис. 15.2. Свойства объекта **Main Camera**

Теперь, когда сцена настроена, настало время поработать с камерой. Поскольку это 2D-проект, мы возьмем ортографическую камеру, у которой отсутствует

глубина перспективы. Эта камера прекрасно подходит для создания 2D-игр. Для настройки объекта **Main Camera** присвойте свойству **Size** значение **6**. (На рис. 15.2 показаны свойства камеры.)

Фон

Настроить правильную прокрутку фона сложновато. Идея такая: нам нужно два фоновых объекта, движущихся вниз по экрану. Как только нижний объект выходит за кадр, мы помещаем его над экраном. Они так и будут чередоваться, не оповещая игрока. Чтобы реализовать прокрутку фона, выполните следующие действия.

1. Создайте новую папку с именем **Background**. Найдите изображение *Star_Sky.png* в прилагаемых файлах примеров и импортируйте его в Unity, перетащив в папку **Background**. Помните, что мы создавали 2D-проект, так что изображение автоматически импортируется как спрайт.

2. Выберите импортированный спрайт на панели **Project** и задайте его свойству **Pixels per Unit** значение **50**. Перетащите на сцену спрайт **Star_Sky** и расположите его в позиции с координатами (0, 0, 0).

3. Создайте в папке **Background** новый скрипт под названием **ScrollBackground** и перетащите его на фоновый спрайт на сцене. Поместите в скрипт следующий код:

```
    public float speed = -2f;
    public float lowerYValue = -20f;
    public float upperYValue = 40;

    void Update()
    {
        transform.Translate(0f, speed * Time.deltaTime, 0f);
        if (transform.position.y <= lowerYValue)
        {
            transform.Translate(0f, upperYValue, 0f);
        }
}
```

4. Продублируйте спрайт **Background** и поместите его в позицию с координатами (0, 20, 0). Запустите сцену. Фон будет плавно двигаться мимо вас.

ПРИМЕЧАНИЕ

Варианты реализации

Ранее мы использовали достаточно простую систему организации. Ассеты у нас лежали в соответствующих им папках: спрайты в папке спрайтов, скрипты в папке

скриптов, и так далее. Но в этом часе мы попробуем кое-что новенькое. Мы будем собирать ассеты в группы в зависимости от функций: в одной папке ассеты для фона, в другой — для корабля, и так далее. Эта система удобна, чтобы находить все ассеты, связанные с определенной частью игры. Но вы можете применять и другую систему, а именно сортировать ассеты по типам и именам, используя строку поиска и фильтры, расположенные в верхней части панели **Project**. Как недавно сказал мне Бен Тристем: «У каждой задачи есть множество решений!»

ПРИМЕЧАНИЕ

Бесшовная прокрутка

Вы, возможно, заметили небольшую линию в прокручивающемся фоне, который только что создали. Она появляется потому, что изображение, используемое для фона, не подогнано само к себе верхним и нижним краем. Как правило, это не очень заметно, происходящее на экране отвлекает внимание. Но если вам потребуется бесшовный фон в будущем, используйте изображение, подогнанное для склейки.

Игровые сущности

В этой игре нам нужны три основные сущности: игрок, метеор и снаряд. Взаимодействие между этими объектами довольно тривиально. Игрок стреляет снарядами. Снаряды уничтожают метеоры. Метеоры уничтожают игрока. Поскольку игрок может стрелять довольно часто и метеоров порождается тоже много, нужен способ избавляться от лишних объектов. Таким образом, нужно создать триггеры, которые будут уничтожать сталкивающиеся с ними снаряды и метеоры.

Игрок

В роли игрока выступает космический корабль. Спрайты для него и для метеора можно найти в файлах примеров для часа 15. (Спасибо Краси Василев, **freegameassets.blogspot.com**.) Чтобы создать игрока, выполните следующие действия.

1. Создайте на панели **Project** новую папку с именем **Spaceship** и импортируйте в нее спрайт *spaceship.png*. Обратите внимание, что космический корабль «смотрит» вниз. Это нормально.

2. Выберите спрайт корабля, на панели **Inspector** задайте свойству **Sprite Mode** значение **Multiple** и нажмите кнопку **Apply**. Затем нажмите кнопку **Sprite Editor**, чтобы начать нарезку листа спрайтов. (Если вы забыли, как делать нарезку, вернитесь к часу 12.).

3. Нажмите кнопку **Slice** в верхнем левом углу окна **Sprite Editor** и задайте свойству **Type** значение **Grid By Cell Size**. Свойству **X** задайте значение **116**,

а свойству **Y −140** (рис. 15.3). Нажмите кнопку **Slice** и обратите внимание на контуры вокруг кораблей. Нажмите кнопку **Apply** и закройте окно **Sprite Editor**.

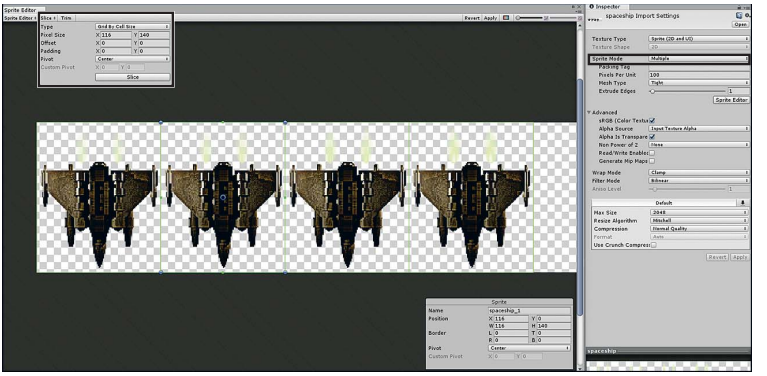


Рис. 15.3. Нарезка спрайта космического корабля

4. Откройте содержимое спрайта корабля и выберите все кадры. Вы можете сделать это, щелкнув мышью по первому кадру и, удерживая нажатой клавишу **Shift**, затем щелкнуть по последнему кадру.

5. Перетащите кадры на панель **Hierarchy** или **Scene**. Появится диалоговое окно **Create New Animation**, и ваша новая анимация будет сохранена в файл с расширением *.anim*. Присвойте файлу имя **Ship**. Когда вы закончите, на сцене появится анимационный спрайт, а на панели **Project** — ассет контроллера анимации и клип, как показано на рис. 15.4. (Вы узнаете больше об анимации в часах 17 и 18).

6. Установите корабль в позицию с координатами (0, −5, 0), задайте масштаб (1, −1, 1). Обратите внимание, что масштаб −1 по оси *y* поворачивает корабль в другую сторону, носом вверх.

7. Для компонента **Sprite Renderer** корабля задайте свойству **Order in Layer** значение **1**. Таким образом корабль всегда будет отображаться на переднем плане.

8. Добавьте кораблю компонент **Polygon Collider** (выбрав команду **Add Component ⇒ Physics2D ⇒ Polygon Collider**). Этот коллайдер будет автоматически окружать ваш корабль, давая неплохую точность обнаружения столкновения. Обязательно установите флажок **Is Trigger**, чтобы создать триггер-коллайдер.

9. Запустите игру и обратите внимание на легкую анимацию двигателей корабля.

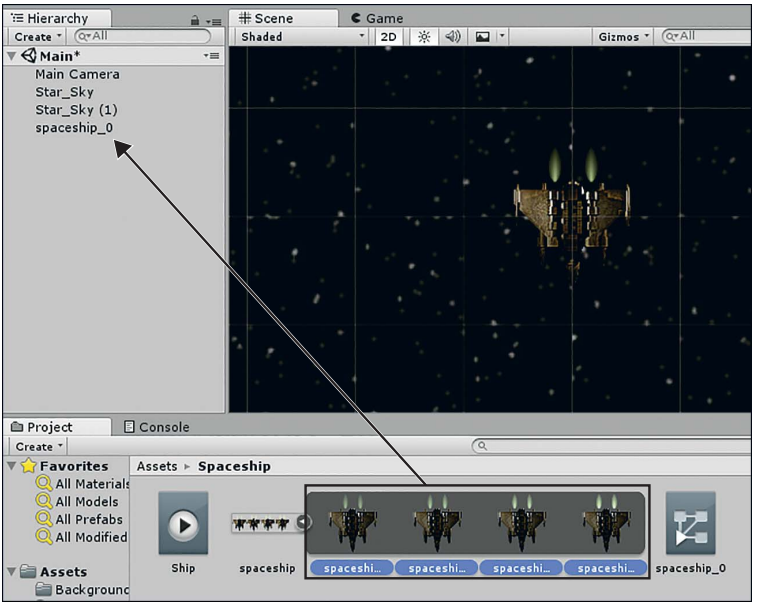


Рис. 15.4. Готовый спрайт космического корабля

Теперь у вас есть симпатичный анимированный летящий вверх космический корабль, готовый уничтожить несколько метеоров.

Метеоры

Порядок создания метеоров такой же, как и для космического корабля. Единственное отличие состоит в том, что метеоры мы превратим в префабы для дальнейшего использования. Выполните следующие действия.

1. Создайте новую папку с именем **Meteor** и импортируйте в нее файл *meteor.png*. Это лист спрайтов, в котором находится 19 кадров анимации.

2. Задайте свойству **Sprite Mode** значение **Multiple**, а затем откройте редактор **Sprite Editor**, как в предыдущем разделе.

3. Задайте свойству **Slice Type** значение **Automatic**, а другие настройки оставьте по умолчанию. Нажмите кнопку **Apply**, чтобы применить изменения, и закройте окно **Sprite Editor**.

4. Разверните ассет спрайта метеора и выберите все 19 кадров на листе спрайтов. Перетащите их на панель **Hierarchy** и в появившемся окне сохраните анимацию под именем **Meteor**. Unity создаст для вас еще один анимационный спрайт с необходимыми компонентами. Круто!

5. Выберите игровой объект **meteor_0** на панели **Hierarchy** и добавьте в него компонент **Circle Collider 2D** (команда **Add Component** ⇒ **Physics2D** ⇒ **Circle Collider 2D**). Обратите внимание, что зеленый контур примерно соответствует контуру спрайта. Для игры «Капитан Бластер» такой точности вполне достаточно. Полигональный коллайдер будет менее эффективным и не даст существенного улучшения точности.

6. Для компонента **Sprite Renderer** метеора задайте свойству **Order in Layer** значение **1**, чтобы метеор всегда отрисовывался перед фоном.

7. Добавьте к метеору компонент **Rigidbody2D** (выбрав команду **Add Component** ⇒ **Physics2D** ⇒ **Rigidbody2D**). Задайте свойству **Gravity Scale** значение **0**.

8. Переименуйте игровой объект **meteor_0** в **Meteor**, а затем перетащите его из панели **Hierarchy** в папку **Meteor** на панели **Project** (рис. 15.5). Unity создаст префаб метеора, которым мы воспользуемся позже.

9. Теперь, когда наш настроенный метеор сохранен в префаб, удалите его экземпляр из панели **Hierarchy**. И вот у вас есть «многоразовый» метеор, готовый разломать пару посудин.

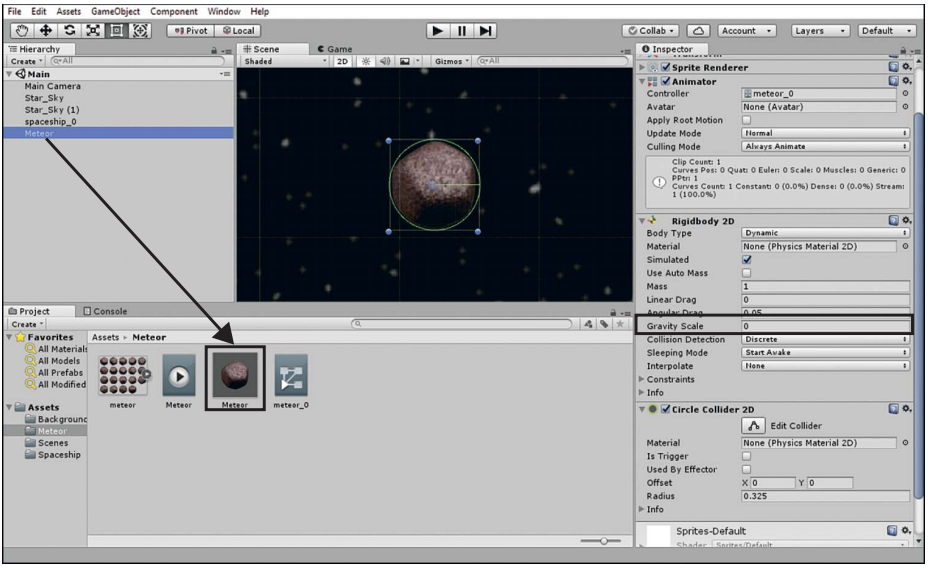


Рис. 15.5. Создание префаба метеора

Снаряды

Настроить снаряды для этой игры весьма просто. Поскольку они движутся очень быстро, тщательной детализации им не потребуется. Чтобы создать снаряд, выполните следующие действия.

1. Создайте папку с именем **Bullet** и импортируйте в нее файл *bullet.png*. Выбрав спрайт снаряда на панели **Project**, задайте свойству **Pixels per Unit** значение **400**.
2. Перетащите спрайт снаряда на сцену. Настроив свойство **Color** компонента **Sprite Renderer**, окрасьте снаряд в зеленый цвет.
3. В компоненте **Sprite Renderer** снаряда задайте свойству **Order in Layer** значение **1**, чтобы он отрисовывался на переднем плане.
4. Добавьте к снаряду компонент **Circle Collider 2D**, компонент **Rigidbody2D** (выбрав команду **Add Component** ⇒ **Physics2D** ⇒ **Rigidbody2D**) и задайте свойству **Gravity Scale** значение **0**.
5. Присвойте игровому объекту имя **Bullet**, чтобы соблюдался порядок в именах. Перетащите снаряд с панели **Hierarchy** в папку **Bullet**, чтобы превратить его в префаб. Удалите снаряд со сцены.

На этом с первичной подготовкой всё. Единственное, что осталось сделать, — настроить триггеры, которые не дадут снарядам и метеорам улетать в бесконечность космоса.

Триггеры

Триггеры (которые в этой игре называются «дробилками») будут реализованы как два коллайдера — один над экраном, а другой под ним. Их задача состоит в том, чтобы уничтожать вылетающие за пределы экрана снаряды и метеоры. Выполните следующие действия, чтобы создать дробилки.

1. Добавьте на сцену пустой игровой объект (выбрав команду **GameObject** ⇒ **Create Empty**) и присвойте ему имя **Shredder**. Поместите созданный объект в позицию с координатами (0, −10, 0).
2. Добавьте к дробилке компонент **Box Collider 2D** (выбрав команду **Add Component** ⇒ **Physics2D** ⇒ **Box Collider 2D**). Не забудьте на панели **Inspector** для компонента **Box Collider 2D** установить флажок **Is Trigger** и настроить его размер, равный (16, 1).
3. Продублируйте дробилку и поместите копию в позицию с координатами (0, 10, 0).

Позже эти два триггера уничтожат любые предметы, которые будут сталкиваться с ними — а именно метеоры и снаряды.

Пользовательский интерфейс

Наконец, нам нужно создать простой пользовательский интерфейс для отображения текущего счета игрока и вывода сообщения «Game Over!», когда игрок погибает. Выполните следующие действия.

1. Добавьте на сцену текстовый элемент пользовательского интерфейса (выбрав команду **GameObject** ⇒ **UI** ⇒ **Text**) и присвойте ему имя **Score**.

2. Поместите якорь в верхний левый угол холста и задайте ему положение (100, −30, 0) (рис. 15.6).

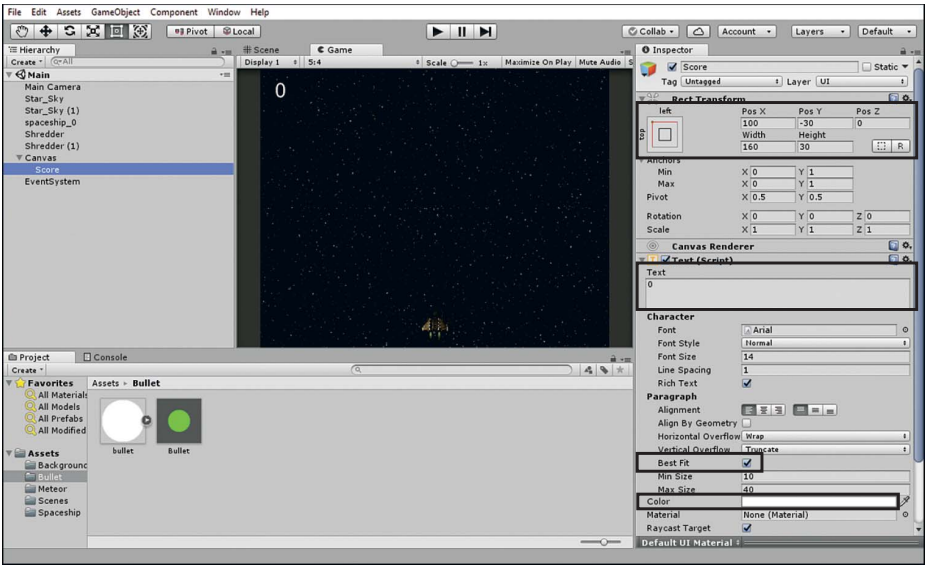


Рис. 15.6. Настройки счета игрока

3. Задайте свойству **Text** объекта Score значение **0** (начальный счет), а свойству **Best Fit** назначьте белый цвет.

Теперь можно добавить сообщение «Game Over!», которое будет отображаться при поражении игрока.

1. Добавьте еще один текстовый элемент интерфейса на сцену и присвойте ему имя **Game Over**. Оставьте его якорь в центре и задайте тексту позицию (0, 0, 0).

2. Установите ширину элемента **Game Over** равной **200**, а высоту — равной **100**.

3. Задайте свойству **Text** значение **Game Over!**, установите флажок **Best Fit**, установите выравнивание (**Alignment**) по центру и измените цвет на красный.

4. Наконец, сбросьте флажок **Text (Script)**, чтобы скрыть текст до тех пор, пока он нам не понадобится (рис. 15.7). Обратите внимание, что на рис. 15.7 показан текст до отключения, чтобы было понятно, как он выглядит.

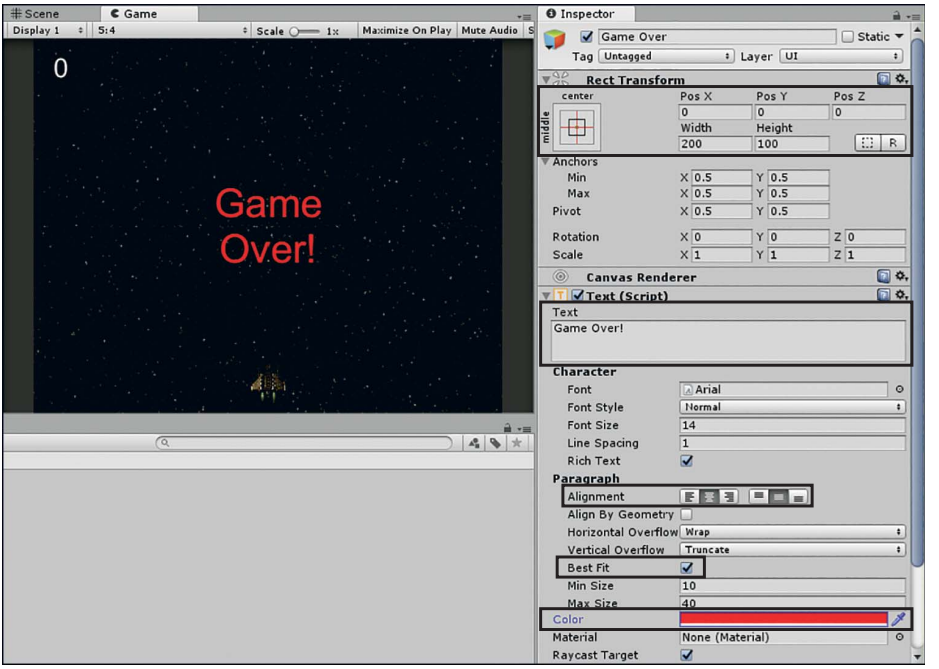


Рис. 15.7. Настройка сообщения «Game Over!»

Позже мы подключим отображение очков к скрипту **GameManager**, чтобы счет обновлялся. Теперь все нужные объекты готовы, и пора превратить сцену в игру.

Управление

Чтобы игра заработала, нужно много скриптов. Игрок должен иметь возможность двигать корабль и стрелять снарядами. Снаряды и метеоры должны перемещаться автоматически. Нужен объект-спаунер для метеоров, чтобы они создавались непрерывно. Дробилки должны удалять лишние объекты, а менеджер игры — отслеживать все происходящее.

Менеджер игры

Менеджер правит бал в этой игре, поэтому добавим его. Чтобы его создать, выполните следующие действия.

1. Создайте пустой игровой объект и присвойте ему имя **GameManager**.
2. Поскольку единственный ассет для менеджера игры — это скрипт, создайте папку с именем **Scripts**, чтобы вам было куда сохранять ваши скрипты.
3. Создайте скрипт под названием **GameManager** в папке **Scripts** и прикрепите его к объекту **GameManager**. Поместите в скрипт следующий код:

```
using UnityEngine;
using UnityEngine.UI; //Обратите внимание на важность этой строки
                      //для интерфейса

public class GameManager: MonoBehaviour
{
    public Text scoreText;
    public Text gameOverText;

    int playerScore = 0;

    public void AddScore()
    {
        playerScore++;
        // превращает счет (число) в строку
        scoreText.text = playerScore.ToString();
    }

    public void PlayerDied()
    {
        gameOverText.enabled = true;
        // приостанавливает игру
        Time.timeScale = 0;
    }
}
```

Из кода видно, что менеджер управляет подсчетом очков и отслеживает запуск игры. Менеджер создает два открытых метода: `PlayerDied()` и `AddScore()`. Метод `PlayerDied()` будет вызываться игроком при столкновении с метеором. Метод `AddScore()` вызывается снарядом при разрушении метеора.

Не забудьте перетащить объекты **Score** и **Game Over** на скрипт **GameManager** (рис. 15.8).

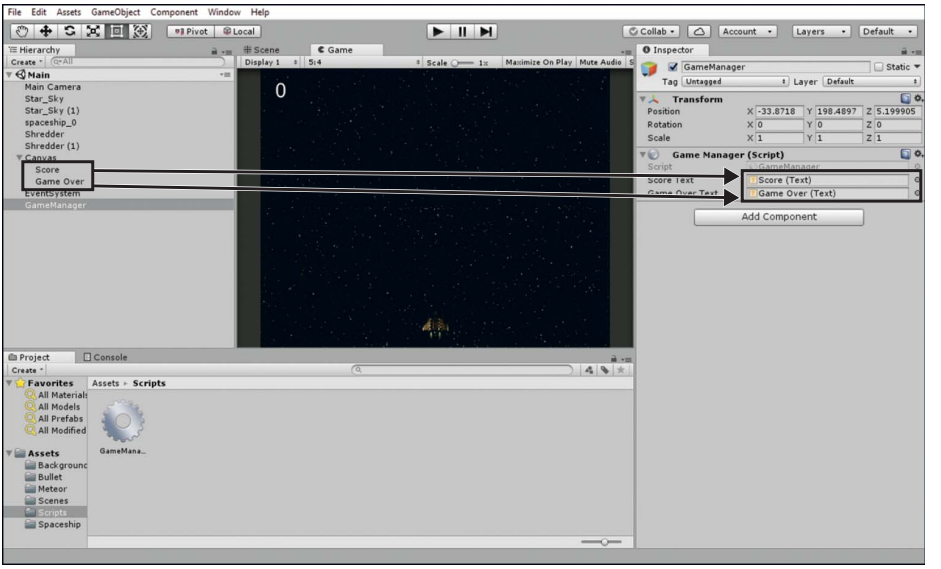


Рис. 15.8. Прикрепление текстовых элементов к менеджеру игры

Скрипт для метеора

Метеор будет падать из верхней части экрана и мешать игроку на пути. Чтобы создать скрипт для метеора, выполните следующие действия.

1. Создайте новый скрипт в папке **Meteor** и присвойте ему имя **MeteorMover**.
2. Выберите префаб **Meteor**. На панели **Inspector** найдите кнопку **Add Component** (рис. 15.9) и выберите команду **Add Component** ⇒ **Scripts** ⇒ **MeteorMover**.
3. Добавьте в скрипт **MeteorMover** следующий код:

```
using UnityEngine;

public class MeteorMover: MonoBehaviour
{
    public float speed = -2f;

    Rigidbody2D rigidBody;

    void Start()
    {
        rigidBody = GetComponent<Rigidbody2D>();
        // задаем начальную скорость метеора
        rigidBody.velocity = new Vector2(0, speed);
    }
}
```

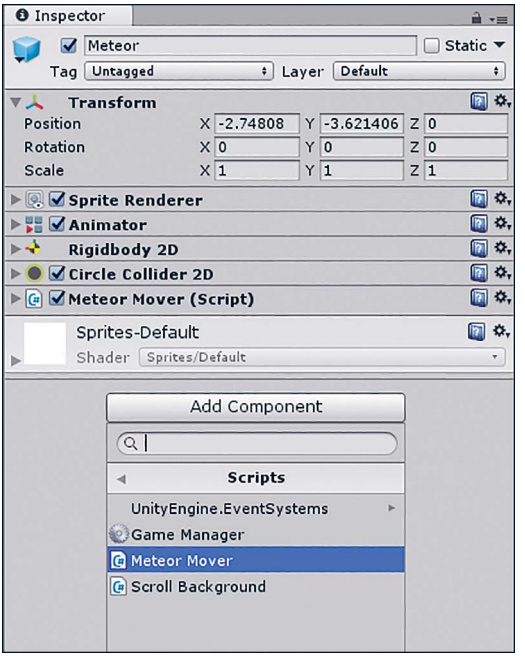


Рис. 15.9. Добавление скрипта **Meteor** в соответствующий префаб

Метеор устроен очень просто — в нем всего одна открытая переменная, учитывающая скорость его движения. В методе `Start()` мы создаем ссылку на компонент **Rigidbody 2D** метеора. Затем мы используем этот компонент, чтобы задать скорость метеора: он движется вниз, так как скорость имеет отрицательное значение. Обратите внимание, что метеор не выполняет функцию определения столкновений.

Вы можете перетащить префаб метеора на панель **Hierarchy** и запустить игру, чтобы проверить работоспособность кода, который вы только что написали. Метеор должен двигаться вниз и немного вращаться. Если вы сделаете это, не забудьте удалить экземпляр метеора с панели **Hierarchy**, когда насладитесь зрелищем.

Спаун метеоров

Пока наши метеоры — это лишь префабы, которые сами на сцену не попадают. Вам нужен объект, который будет спаунить метеоры через определенный интервал времени. Создайте новый пустой игровой объект, присвойте ему имя **Meteor Spawn** и поместите в позицию с координатами (0, 8, 0). Создайте новый скрипт под названием **MeteorSpawn** в папке **Meteor** и поместите его на объект **Meteor Spawn**. Добавьте в скрипт следующий код:

```
using UnityEngine;

public class MeteorSpawn: MonoBehaviour
{
    public GameObject meteorPrefab;
    public float minSpawnDelay = 1f;
    public float maxSpawnDelay = 3f;
    public float spawnXLimit = 6f;

    void Start()
    {
        Spawn();
    }

    void Spawn()
    {
        // Метеор создается в случайной позиции по оси x
        float random = Random.Range(-spawnXLimit, spawnXLimit);
        Vector3 spawnPos = transform.position + new Vector3(random, 0f, 0f);
        Instantiate(meteorPrefab, spawnPos, Quaternion.identity);

        Invoke("Spawn", Random.Range(minSpawnDelay, maxSpawnDelay));
    }
}
```

Этот скрипт решает несколько интересных задач. Во-первых, он создает две переменные для управления временем появления метеоров. Он также объявляет переменную **GameObject**, которая соответствует префабу метеора. В методе `Start()` вызывается функция `Spawn()`. Она отвечает за появление и размещение метеоров.

Легко заметить, что метеоры порождаются в тех же координатах *y* и *z*, что и точка спауна, но координата *x* вариабельна в диапазоне от –6 до 6. Это позволяет метеорам спауниться в разных местах экрана. Когда определяется положение нового метеора, функция `Spawn()` инициализирует (создает) метеор в новой точке, задавая вращение по умолчанию (`Quaternion.identity`). Последняя строка вызывает функцию спауна снова. Метод `Invoke()` вызывает указанную функцию (в данном случае `Spawn()`) через случайный промежуток времени. Величина промежутка зависит от значений двух переменных.

В редакторе Unity перетащите префаб метеора с панели **Project** на свойство **Meteor Prefab** компонента **Meteor Spawn Script** объекта **Meteor Spawn** (скороговорка, ей-богу!). Запустите сцену, чтобы увидеть, как метеоры спаунятся по всему экрану. (В атаку, приспешники!)

Скрипт DestroyOnTrigger

Теперь, когда с порождением метеоров все в порядке, надо заняться уборкой мусора. Создайте новый скрипт **DestroyOnTrigger** в папке **Scripts** (так как он ни с чем не связан) и прикрепите его к обеим (верхней и нижней) дробилкам, созданным ранее. Добавьте в скрипт следующий код, размещая его вне методов, но внутри класса:

```
void OnTriggerEnter2D(Collider2D other)
{
    Destroy(other.gameObject);
}
```

Этот простейший скрипт уничтожает любой объект, который соприкасается с дробилкой. Поскольку игрок не может двигаться по вертикали, корабль не может быть уничтожен. С дробилками сталкиваются только снаряды и метеоры.

Скрипт ShipControl

Метеоры-то у нас летят, но игрок не может увернуться от них. Для этого нужен скрипт управления кораблем. Создайте новый скрипт **ShipControl** в папке **Spaceship** и прикрепите ее к космическому кораблю. Добавьте в скрипт следующий код:

```
using UnityEngine;

public class ShipControl: MonoBehaviour
{
    public GameManager gameManager;
    public GameObject bulletPrefab;
    public float speed = 10f;
    public float xLimit = 7f;
    public float reloadTime = 0.5f;

    float elapsedTime = 0f;

    void Update()
    {
        // Отсчет времени после выстрела
        elapsedTime += Time.deltaTime;

        // Перемещение игрока влево и вправо
        float xInput = Input.GetAxis("Horizontal");
        transform.Translate(xInput * speed * Time.deltaTime, 0f, 0f);

        // Фиксируем положение корабля по оси x
        Vector3 position = transform.position;
        position.x = Mathf.Clamp(position.x, -xLimit, xLimit);
        transform.position = position;
```

```
        // Огонь клавишей "Пробел". Ввод по умолчанию в InputManager
        // называется "Jump"
        // Срабатывает только в том случае, если время перезарядки
        // уже прошло
        if (Input.GetButtonDown("Jump") && elapsedTime > reloadTime)
        {
            // Создание экземпляра снаряда на расстоянии 1,2 единицы
            // перед игроком
            Vector3 spawnPos = transform.position;
            spawnPos += new Vector3(0, 1.2f, 0);
            Instantiate(bulletPrefab, spawnPos, Quaternion.identity);

            elapsedTime = 0f; // Reset bullet firing timer
        }
    }

    // Столкновение метеора с игроком
    void OnTriggerEnter2D(Collider2D other)
    {
        gameManager.PlayerDied();
    }
}
```

Этот скрипт делает многое. Во-первых, он создает переменные для менеджера игры, префаба снаряда, скорости, ограничения движения и скорострельности.

В методе `Update()` скрипт сначала отслеживает, сколько прошло времени после последнего выстрела, чтобы определить, можно ли стрелять снова. Помните, что в соответствии с правилами, игрок может выстрелить только один раз в полсекунды. Игрок перемещается вдоль оси *x* в зависимости от управления со стороны пользователя. Положение оси *x* игрока ограничено, поэтому он не может выйти за пределы экрана влево или вправо. Далее скрипт определяет, нажимал ли игрок клавишу **Пробел**. Как правило, в проектах Unity клавиша **Пробел** используется для прыжков (действие можно переназначить в менеджере ввода, но мы этого не делали, чтобы избежать путаницы.) Если скрипт фиксирует, что игрок нажимает клавишу **Пробел**, он сравнивает прошедшее время с временем перезарядки `reloadTime` (в настоящее время полсекунды). Если прошло больше времени — скрипт создает снаряд. Обратите внимание, что снаряд находится в позиции чуть выше корабля. Это предотвращает столкновения снаряда и корабля. Прошедшее время становится равным 0 и запускается счетчик для следующего выстрела.

В последней части скрипта описан метод `OnTriggerEnter2D()`. Он вызывается всякий раз, когда метеор сталкивается с игроком. Когда это произойдет, скрипт **GameManager** будет знать, что игрок погиб.

Вернитесь в редактор Unity и перетащите префаб снаряда на компонент **Ship Control** корабля. Затем перетащите объект **Game Manager** на компонент **Ship Control**, чтобы дать ему доступ к скрипту **GameManager** (рис. 15.10). Запустите сцену и обратите внимание, что теперь вы можете перемещать игрока. Игрок может стрелять (хотя снаряды не двигаются). Также обратите внимание, что игрок теперь может погибнуть и закончить игру.

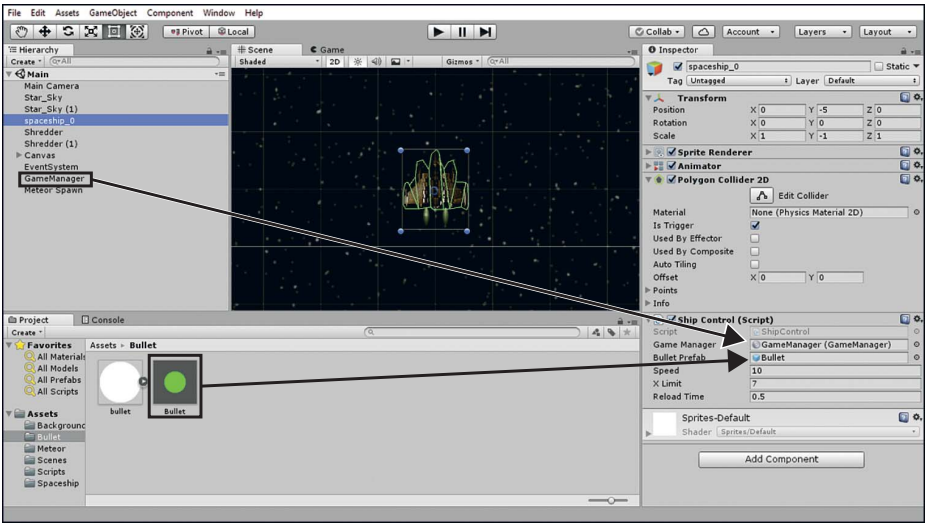


Рис. 15.10. Подключение скрипта **ShipControl**

Скрипт для снаряда

Последнее, что нам нужно из функционала, — заставить снаряды двигаться и сталкиваться с метеорами. Создайте новый скрипт под названием **Bullet** в папке **Bullet** и добавьте его к префабу снаряда. Поместите в скрипт следующий код:

```
using UnityEngine;

public class Bullet: MonoBehaviour
{
    public float speed = 10f;

    GameManager gameManager; // Note this is private this time

    void Start()
    {
        // Пока игра не запущена и снарядов на сцене нет, обращаться
        // к объекту Game Manager приходится по-другому.
        gameManager = GameObject.FindObjectOfType<GameManager>();
```

```
        Rigidbody2D rigidBody = GetComponent<Rigidbody2D>();
        rigidBody.velocity = new Vector2(0f, speed);
    }

    void OnCollisionEnter2D(Collision2D other)
    {
        Destroy(other.gameObject); // уничтожение метеора
        gameManager.AddScore(); // увеличение счета
        Destroy(gameObject); // уничтожение снаряда
    }
}
```

Основное различие между этим скриптом и скриптом метеора заключается в том, что здесь обрабатываются и столкновения, и очки игрока. Скрипт снаряда объявляет переменную, которой ссылается на скрипт **GameManager**, как делает и скрипт **ShipControl**. Поскольку снаряд на сцене отсутствует, ему приходится искать скрипт **GameManager** немного по-другому. В методе `Start()` скрипт ищет объект **GameControl** по типу с помощью метода `GameObject.FindObjectOfType<`*тип*`>()`. Ссылка на скрипт **GameManager** сохраняется в переменной `gameManager`. Стоит отметить, что метод `Fins()` в Unity (например, тот, который используется здесь) работает очень медленно, и его следует применять с осторожностью.

Поскольку ни у снаряда, ни у метеора нет триггер-коллайдера, метод `OnTriggerEnter2D()` не будет работать. Вместо этого скрипт использует метод `OnCollisionEnter2D()`, который считывает данные не в переменную `Collider2D`, а в переменную `Collision2D`. Различия между этими двумя методами в данном случае не имеют значения. Главное, что в результате разрушаются оба объекта, и **GameManager** получает информацию о том, что игроку надо прибавить одно очко.

Запустите игру. Теперь ваша игра полностью работоспособна. Вы не можете выиграть (так сделано специально), но при этом вы можете проиграть. Продолжайте играть и посмотрите, какой счет вам удастся заработать!

Веселья ради можете уменьшить значения переменных `minSpawnDelay` и `maxSpawnDelay`, чтобы метеоры спаунились чаще по ходу игры.

Улучшения

Теперь вы можете улучшить вашу игру. Как и в созданных ранее играх, некоторые части игры «Капитан Бластер» оставлены намеренно упрощенными. Поиграйте несколько раз, и вы наверняка заметите что-нибудь. Что в игре хорошо? Что не очень? Есть ли какие-либо очевидные способы сломать игру? Обратите

внимание, что в игре есть лазейка, позволяющая игроку заработать очень много очков. Вы нашли ее?

Вот что вы могли бы изменить.

- Попробуйте изменить скорость снаряда, скорострельность, траекторию движения.
- Попробуйте сделать так, чтобы игрок мог стрелять сразу двумя снарядами.
- Ускоряйте спаун метеоров со временем.
- Попробуйте добавить другие типы метеоров.
- Дайте игроку дополнительное здоровье или защитный щит.
- Разрешите игроку перемещаться и по вертикали, и по горизонтали.

Это довольно распространенный жанр, и есть много способов сделать вашу игру уникальной. Постарайтесь найти свой вариант. Стоит также отметить, что знания о системах частиц из часа 16 помогут вам сделать игру существенно круче.

Резюме

В этом часе мы создали игру «Капитан Бластер». Мы начали с проекта игровых элементов. Далее мы создали игровой мир. Мы добились вертикальной прокрутки фона. Затем мы создали различные игровые сущности. После этого мы добавили функционал с помощью скриптов и элементов управления. И наконец, мы изучили игру и подумали, как можно ее улучшить.

Вопросы и ответы

Вопрос: **Капитан Бластер и правда дослужился до капитана или это просто прозвище?**

Ответ: Кто знает, как оно было на самом деле. Одно можно сказать наверняка: простым лейтенантам космические корабли не дают!

Вопрос: **Почему задержка выстрела равна половине секунды?**

Ответ: В основном это вопрос баланса. Если игрок сможет стрелять быстрее, игра будет слишком легкой.

Вопрос: **Зачем использовать полигональный коллайдер на корабле?**

Ответ: Поскольку корабль имеет нестандартную форму, стандартный коллайдер ему точно не подойдет. К счастью, вы можете использовать полигональный коллайдер, который точно соответствует геометрии корабля.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Какое в этой игре условие победы?
2. Как работает прокрутка фона?
3. Какие объекты имеют компоненты **Rigidbody**? Какие объекты имеют коллайдеры?
4. Верно или нет: метеорит обнаруживает столкновения с игроком.

Ответы

1. Это вопрос с подвохом. Игрок не может выиграть. Но подсчет очков позволяет ему превзойти соперников.
2. Два идентичных спрайта, размещенные один над другим на оси *y*. Сменяя друг друга за кадром, они создают ощущение бесконечности.
3. Снаряды и метеоры имеют компоненты **Rigidbody**. Снаряды, метеоры, корабль и дробилки имеют коллайдеры.
4. Нет. Не метеоры, а скрипт **ShipControl** обнаруживает столкновение.

Упражнение

Это упражнение немного отличается от тех, которые вы делали ранее. Во время доведения игры до совершенства нужно предлагать ее людям, которые не занимались разработкой. Пользователи, совершенно не знакомые с игрой, могут предоставить честные отзывы, которые будут невероятно полезны. Для этого упражнения дайте другим людям поиграть в вашу игру. Постарайтесь найти разнообразную группу, и заядлых геймеров, и людей, которые не играют в игры. Найдите поклонников этого жанра и наоборот, не особых любителей. Отзывы от такой разнообразной компании позволят вам получить полную информацию о том, что в игре нужно улучшить. Кроме того, попытайтесь разобраться, нужно ли добавить какую-нибудь популярную функцию. Наконец, на основе отзывов вы можете реализовать или улучшить свою игру.

16-Й ЧАС
Системы частиц

Что вы узнаете в этом часе

- Основы систем частиц
- Как работать с модулями
- Как использовать редактор **Curves Editor**

В этом часе вы узнаете, как использовать системы частиц на движке Unity. Мы начнем с изучения систем частиц в целом и узнаем, как они работают. Также поэкспериментируем с различными модулями систем частиц. В конце часа мы поработаем с редактором Unity **Curves Editor**.

Системы частиц

Система частиц — это объект или компонент, который излучает другие объекты, обычно называемые *частицами*. Они могут быть быстрыми, медленными, плоскими, объемными, маленькими, большими и так далее. Определение частиц довольно обширно, так как эти системы при правильном использовании позволяют создать огромное разнообразие эффектов. Вы можете сделать струю огня, шлейфы развевающегося дыма, светлячков, дождь, туман и что угодно еще. Все это обычно называют эффектами частиц.

Частицы

Частица — это одиночная сущность, испускаемая системой частиц. Поскольку частицы, как правило, испускаются быстро, они должны быть как можно более эффективны с точки зрения производительности. Именно поэтому большинство частиц представляет собой 2D-билборды. Билборд — это плоское изображение, которое всегда повернуто к камере. Таким образом появляется иллюзия трехмерности и сохраняется производительность.

Системы частиц в Unity

Чтобы расположить на сцене систему частиц, вы можете создать объект системы частиц или добавить компонент системы частиц к существующему объекту. В первом случае выберите команду **GameObject** ⇒ **Effects** ⇒ **Particle System**. Чтобы добавить компонент системы частиц к существующему объекту, выделите объект и затем выберите команду **Add Component** ⇒ **Effects** ⇒ **Particle System**.

ПРАКТИКУМ

Создание системы частиц

Выполните следующие действия, чтобы создать на сцене объект системы частиц.

1. Создайте новый проект или сцену.
2. Добавьте на сцену систему частиц, выбрав команду **GameObject** ⇒ **Effects** ⇒ **Particle System**.
3. Обратите внимание, как система частиц испускает белые частицы на сцене (рис. 16.1). Это базовая система частиц. Попробуйте покрутить и изменить масштаб системы частиц и посмотреть, что получится.

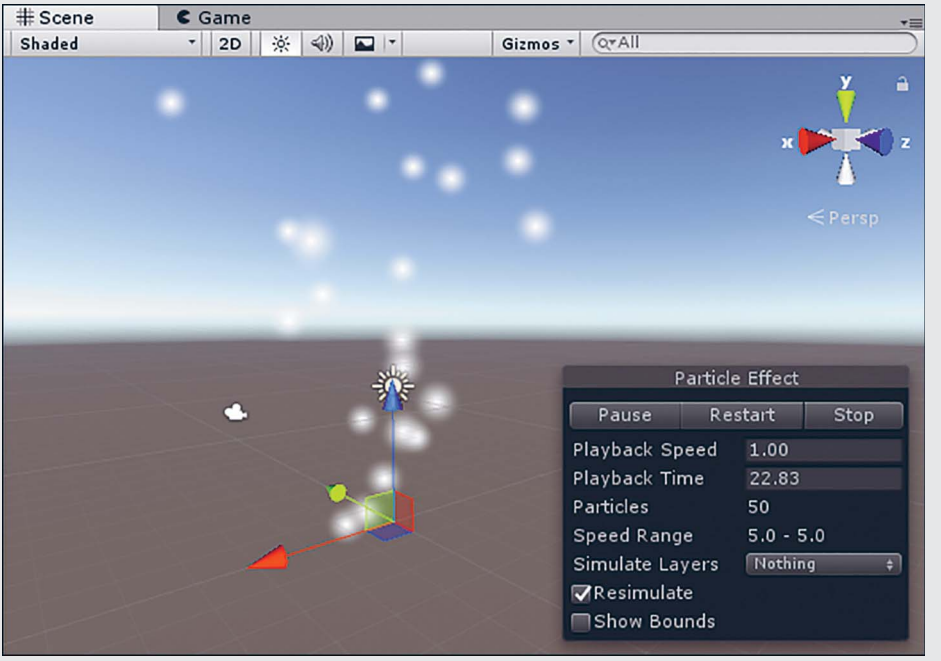


Рис. 16.1. Базовая система частиц

ПРИМЕЧАНИЕ

Пользовательские частицы

По умолчанию частицы в Unity представляют собой маленькие белые шары, со временем исчезающие. Это неплохой вариант для начала, но далеко на нем не уедешь. Иногда вам требуется что-то более конкретное (огонь, например). По желанию, вы можете создать свои частицы из любого 2D-изображения, чтобы воспроизвести эффекты, которые точно будут удовлетворять вашим потребностям.

Элементы управления системой частиц

Вы, возможно, заметили, что, когда добавили на сцену систему частиц, она начала испускать частицы. Вы также могли заметить, что появилась панель управления системой частиц (рис. 16.2). Элементы управления на ней позволяют приостанавливать, выключать и перезапускать анимацию частиц на сцене. Это крайне полезно при выполнении тонкой настройки поведения системы частиц.

Рис. 16.2. Система управления эффектами частиц

Элементы управления также позволяют ускорить воспроизведение и выводят информацию о том, как долго оно длится. Это может быть очень полезно при тестировании длительности эффектов. Обратите внимание, что элементы управления показывают скорость и время воспроизведения только в том случае, если игра остановлена.

ПРИМЕЧАНИЕ

Эффекты частиц

Чтобы создавать сложные и визуально привлекательные эффекты, можно одновременно использовать несколько систем частиц (например, дым и огонь). Совокупность систем частиц, работающих вместе, создает *эффект частиц*. В Unity такое достигается за счет совмещения систем частиц. Одна система может быть дочерней для другой, или обе они могут быть дочерними для какого-то объекта. В результате они рассматриваются как одна система, и вы можете использовать элементы управления для всей такой системы сразу.

Модули систем частиц

В принципе, система частиц — лишь точка в пространстве, которая испускает объекты частиц. Внешний вид и поведение частиц, а также их воздействие, задаются модулями. *Модули* — это различные свойства, которые определяют формы поведения. В системах частиц в Unity модули являются неотъемлемой и важной частью. В этом разделе мы перечислим модули и кратко опишем назначение каждого из них.

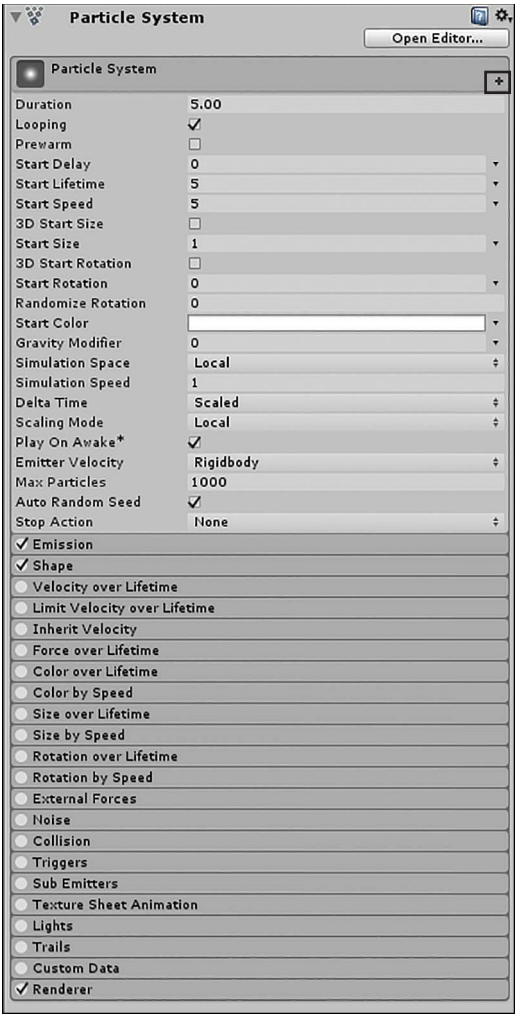


Рис. 16.3. Список модулей

Следует отметить, что за исключением модуля по умолчанию (его мы рассмотрим первым), все прочие можно включать и выключать. Для этого нужно установить

флажок рядом с именем модуля. Чтобы скрыть или отобразить модули, следует щелкнуть мышью по значку + рядом с системой частиц (см. рис. 16.3). Вы также можете щелкнуть мышью по имени системы частиц в списке, чтобы скрыть или отобразить ее. По умолчанию отображаются все модули, но включены из них только **Emission** (Излучение), **Shape** (Форма) и **Renderer** (Отрисовка). Чтобы развернуть модуль, необходимо щелкнуть мышью по его названию.

ПРИМЕЧАНИЕ

Краткий обзор свойств

У некоторых модулей есть свойства, которые либо очевидны (например, длина и ширина прямоугольника), либо были рассмотрены ранее. Ради упрощения (и чтобы этот час не растянулся на несколько десятков страниц) мы не будем описывать их. Поэтому, если у вас на экране окажется больше свойств, чем вы узнаете, не беспокойтесь: так задумано.

ПРИМЕЧАНИЕ

Параметры Constant, Curve, Random

Кривая значения позволяет изменять значение свойства на протяжении всего существования системы частиц или отдельной частицы. Свойство регулируется кривой, если рядом с его значением есть направленная вниз стрелка. Вам будут часто попадаться параметры **Constant**, **Curve**, **Random Between Two Constants** и **Random Between Two Curves**. В данном разделе всегда будет использоваться постоянное значение — **Constant**. Позже в этом часе мы поподрбнее рассмотрим редактор **Curves Editor**.

Модуль по умолчанию

Модуль по умолчанию называется **Particle System**. В нем содержится вся информация, необходимая для каждой системы частиц. В таблице 16.1 описаны его свойства.

ТАБЛ. 16.1. Свойства модуля по умолчанию

Свойство	Описание
Duration	Определяет, как долго (в секундах) работает система частиц
Looping	Определяет, перезапускается ли система частиц, когда воспроизведение закончено
Prewarm	Определяет, должна ли система частиц запускаться так, как если бы она уже излучала частицы в предыдущем цикле
Start Delay	Определяет, сколько времени в секундах система ожидает, прежде чем начать излучать частицы

Start Lifetime	Определяет, как долго существует каждая частица
Start Speed	Определяет начальную скорость частиц
Start Size	Определяет начальный размер частицы. Если выбрать вариант **3D Start Size**, можно указать различные значения размера вдоль трех осей. В противном случае ко всем осям будет применяться один размер
Start Rotation	Определяет начальное вращение частиц. Если выбрать вариант **3D Start Rotation**, можно указать различные значения вращения относительно трех осей. В противном случае ко всем осям будет применяться одно вращение
Randomize Rotation	Настраивает некоторые частицы вращаться в противоположном направлении
Start Color	Определяет цвет излучаемых частиц
Gravity Modifier	Определяет, насколько сильно на частицы действует гравитация
Simulation Space	Определяет, какая используется система координат — глобальная или локальная система координат родительского объекта
Simulation Speed	Позволяет выполнить тонкую настройку скорости для всей системы частиц
Delta Time	Определяет, используется ли масштабированное или немасштабированное время для системы частиц
Scaling Mode	Определяет, чем задается масштаб: игровым объектом, родительским объектом или формой излучателя
Play on Wake	Определяет, начинает ли система частиц воспроизведение сразу после создания
Emitter Velocity	Определяет, вычисляется ли скорость от преобразования объекта или от его **Rigidbody** (если он есть)
Max Particles	Определяет суммарное число частиц, которое может единовременно существовать для системы. Если это число будет достигнуто, то система перестанет излучать до тех пор, пока некоторые частицы не исчезнут
Auto Random Seed	Определяет, будет ли система частиц выглядеть по-разному при каждом воспроизведении
Stop Action	Позволяет указать, что происходит, когда система частиц заканчивает работу. Например, вы можете отключить или уничтожить игровой объект или запустить скрипт

Модуль Emission

Модуль **Emission** (Излучение) используется для определения скорости, с которой излучаются частицы. С его помощью вы можете настроить излучение с постоянной скоростью, волнами или что-то среднее. В таблице 16.2 описаны свойства модуля **Emission**.

ТАБЛ. 16.2. Свойства модуля Emission

Свойство	Описание
Rate over Time	Определяет число частиц, излучаемых с течением времени
Rate over Distance	Определяет число частиц, излучаемых в зависимости от расстояния
Bursts	Позволяет задать всплески частиц через определенные интервалы времени. Вы можете создать всплеск, щелкнув мышью по значку + и удалить его значком – (рис. 16.4)

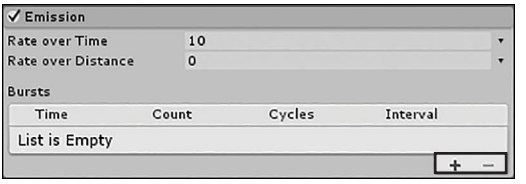


Рис. 16.4. Модуль **Emission**

Модуль Shape

Как следует из названия, модуль **Shape** (Форма) определяет форму, образуемую излучаемыми частицами. Это может быть сфера, полусфера, конус, пончик, куб, сетка, рендерер сетка, скин-сетка, круг и грани. Кроме того, у каждой формы есть набор свойств, например радиус для конусов и сфер. Они довольно очевидны и понятны, поэтому здесь мы их не рассматриваем.

Модуль Velocity over Lifetime

Модуль **Velocity over Lifetime** (Скорость в течение срока существования) непосредственно анимирует каждую частицу путем нанесения на нее осей *x*, *y*, и *z*. Обратите внимание, что речь идет о времени жизни конкретной частицы, а не всей системы частиц. В таблице 16.3 описаны свойства этого модуля.

ТАБЛ. 16.3. Свойства модуля Velocity over Lifetime

Свойство	Описание
XYZ	Определяет скорость, применяемую к каждой частице. Это может быть постоянная скорость, кривая скорости или случайное число между постоянной и кривой
Space	Определяет, измеряется ли скорость в локальных или глобальных координатах
Speed Modifier	Позволяет масштабировать все скорости частиц сразу

Модуль Limit Velocity over Lifetime

Этот модуль с длинным названием используется для снижения или ограничения скорости частицы. В основном он отключает движение или замедляет частицы, которые разгоняются сверх заданного значения вдоль одной или всех осей.

В таблице 16.4 описаны свойства модуля **Limit Velocity over Lifetime**.

ТАБЛ. 16.4. Свойства модуля Limit Velocity over Lifetime

Свойство	Описание
Separate Axis	Если это свойство отключено, для всех осей используется одно и то же значение. Если свойство включено, используются отдельные скорости для каждой оси, а также выбор локального или глобального пространства
Speed	Определяет порог скорости для каждой или всех осей
Dampen	Определяет коэффициент от 0 до 1, задающий замедление частицы, если ее скорость превышает пороговое значение, заданное свойством Speed. При значении 0 частица не замедляется, а при значении 1 — замедляется на 100%
Drag	Определяет линейное сопротивление, примяемое к частицам
Multiply by Size	Определяет, будут ли более крупные частицы замедляться сильнее
Multiply by Velocity	Определяет, будут ли более быстрые частицы замедляться сильнее

Модуль Inherit Velocity

Модуль **Inherit Velocity** очень прост — он определяет, какая часть скорости излучателя, если таковая имеется, передается частице. Первое свойство, **Mode**, определяет, применяется к частице только начальная скорость или частица также

получает скорость от излучателя. Наконец, свойство **Multiplier** определяет коэффициент передачи скорости.

Модуль Force over Lifetime

Модуль **Force over Lifetime** аналогичен модулю **Velocity over Lifetime**. Разница в том, что этот модуль применяет к каждой частице силу, а не скорость. То есть частица будет продолжать ускоряться в указанном направлении. Этот модуль также позволяет применять случайную силу в каждом кадре, в отличие от остальных.

Модуль Color over Lifetime

Модуль **Color over Lifetime** позволяет изменять цвет частицы с течением времени. Это полезно для создания таких эффектов, как искры, которые сначала имеют ярко-оранжевый цвет, а затем становятся темно-красными, прежде чем исчезнуть. Чтобы использовать этот модуль, необходимо задать градиент цвета. Можно также указать два градиента, чтобы программа Unity случайным образом выбрала один из них. Градиенты можно редактировать с помощью редактора **Gradient Editor** в Unity (рис. 16.5).

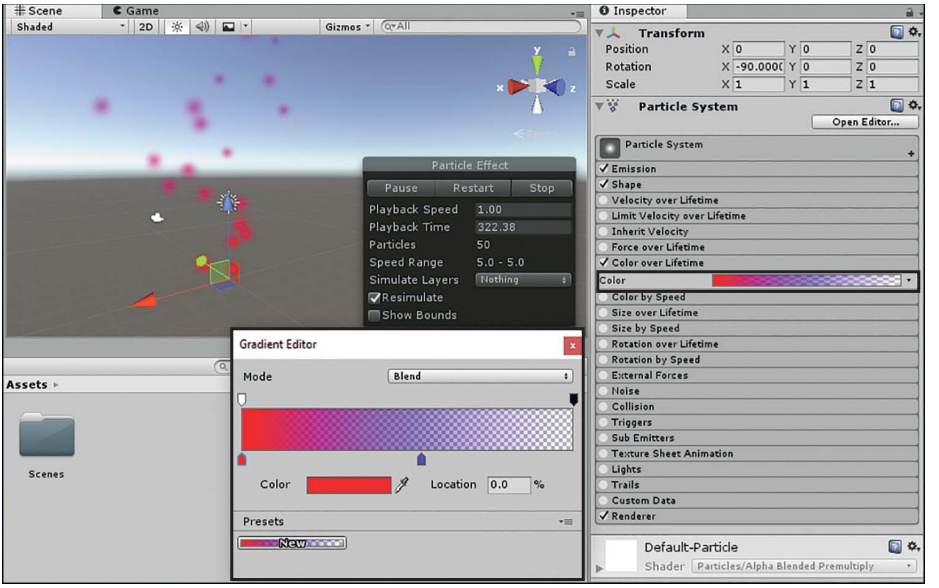


Рис. 16.5. Окно редактора **Gradient Editor**

Обратите внимание, что цвет градиента умножается на свойство **Start Color** модуля по умолчанию. Поэтому, если начальный цвет у нас черный, модуль **Color over Lifetime** не будет иметь никакого эффекта.

Модуль Color by Speed

Модуль **Color by Speed** позволяет изменять цвет частицы в зависимости от ее скорости. В таблице 16.5 описаны его свойства.

ТАБЛ. 16.5. Свойства модуля Color by Speed

Свойство	Описание
Color	Задает градиент (или два градиента для случайного цвета), используемый для определения цвета частицы
Speed Range	Определяет минимальные и максимальные значения скорости, которые подгоняются под градиент цвета. Частицы с минимальной скоростью отображаются в левой части градиента, а частицы с максимальной скоростью (или за ее пределами) — в правой части градиента

Модуль Size over Lifetime

Модуль **Size over Lifetime** позволяет задать изменение размера частицы. Значение размера должно быть представлено кривой, определяющей, как изменяется размер частицы с течением времени.

Модуль Size by Speed

Так же, как и **Color by Speed**, модуль **Size by Speed** задает изменение размера частицы в зависимости от ее скорости в диапазоне между минимальным и максимальным значениями.

Модуль Rotation over Lifetime

Модуль **Rotation over Lifetime** позволяет задать вращение в течение жизни частицы. Обратите внимание, что речь идет о вращении самой частицы, а не кривой в глобальной системе координат. Это означает, что если частица представляет собой простой круг, увидеть вращение вы не сможете. Но если у частицы есть несимметричные детали, вращение будет заметно. Значение вращения может быть задано в виде константы, кривой или случайного числа.

Модуль Rotation by Speed

Модуль **Rotation by Speed** работает так же, как **Rotation over Lifetime**, за исключением того, что значение вращения изменяется в зависимости от скорости частицы. Изменение вращения происходит в диапазоне от минимальных до максимальных значений скорости.

Модуль External Forces

Модуль **External Forces** позволяет применять множитель для любых сил, которые существуют вне системы частиц. Это может быть, например, ветер на сцене. Свойство **Multiplier** задает влияние этой силы через коэффициент.

Модуль Noise

Модуль **Noise** — это относительно новый модуль системы частиц в Unity. Он позволяет добавить случайности к движению частицы (как, например, у молнии). Это достигается путем создания изображения *шума Перлина*, используемого в качестве таблицы поиска. Используемый шум отображается в окне предварительного просмотра модуля. В таблице 16.6 перечислены свойства модуля **Noise**.

ТАБЛ. 16.6. Свойства модуля Noise

Свойство	Описание
Separate Axes	Определяет, будет ли шум применяться одинаково по всем осям или с разными значениями по каждой оси
Strength	Определяет, насколько сильный эффект оказывает шум на частицу в течение срока ее жизни. При более высоких значениях частицы двигаются быстрее и дальше
Frequency	Определяет, как часто частицы меняют направление движения. Более низкие значения создают мягкий, плавный шум, высокие значения — быстро меняющийся
Scroll Speed	Настраивает шумовое поле на перемещение и создание тем самым более непредсказуемого и беспорядочного движения частиц
Damping	Определяет, пропорциональна ли сила частоте
Octaves	Определяет, сколько применяется перекрывающихся слоев шума. Более высокие значения дают более богатый и интересный шум, но снижают производительность
Octave Multiplier	Снижает силу каждого дополнительного слоя шума
Octave Scale	Настраивает частоту каждого дополнительного слоя шума
Quality	Позволяет настроить качество шума, чтобы повысить производительность
Remap и Remap Curve	Позволяет переназначить конечное значение шума чему-то еще. Вы можете использовать кривую, чтобы определить, какие значения шума должны быть превращены в другие значения
Position, Rotation и Size Amount	Задает, как сильно шум влияет на положение, вращение и масштаб частицы

Модуль Collision

Модуль **Collision** позволяет настроить столкновения частиц. Это бывает полезно для всех видов эффектов столкновений, например таких как скатывание со стены или удары капель дождя о землю. Вы можете сделать так, чтобы столкновение работало с заранее заданными плоскостями (режим **Planes** наиболее эффективен) или с объектами на сцене (режим **World** негативно сказывается на производительности). У модуля **Collision** есть общие и уникальные свойства, зависящие от выбранного режима. В таблице 16.7 описаны общие свойства модуля **Collision**. В таблицах 16.8 и 16.9 описаны свойства режимов **Planes** и **World** соответственно.

ТАБЛ. 16.7. Общие свойства модуля Collision

Свойство	Описание
Planes и World	Определяет тип используемых столкновений. Planes — отталкивание от заранее заданных плоскостей. World — столкновение с любым объектом на сцене
Dampen	Определяет замедление частицы после столкновения. Диапазон значений от 0 до 1
Bounce	Определяет, какая часть скорости сохраняется после столкновения. В отличие от демпфирования, параметр влияет только на ось, в которой происходит отскок. Диапазон значений от 0 до 1
Lifetime Loss	Определяет, какая часть срока жизни частицы теряется при столкновении. Диапазон значений от 0 до 1
Min Kill Speed	Определяет минимальную скорость частицы, при которой она уничтожается при столкновении
Max Kill Speed	Определяет скорость частицы, при превышении которой она уничтожается при столкновении
Radius Scale	Настраивает радиус коллайдера частицы, чтобы он соответствовал видимому размеру частицы
Send Collision Messages	Определяет, будут ли объекты получать информацию о столкновении с частицей
Visualize Bounds	Отображает границы коллайдера каждой частицы на сцене в виде рамки

ТАБЛ. 16.8. Свойства режима Planes

Свойство	Описание
Planes	Определяет, с чем частицы могут сталкиваться. Ось преобразований *y* определяет вращение плоскости
Visualisation	Определяет, как плоскости отображаются на сцене. Они могут быть сплошными, либо отображаться в виде сетки
Scale Plane	Отвечает за изменение размера плоскостей

ТАБЛ. 16.9. Свойства режима World

Свойство	Описание
Collision Mode	Определяет режим: 2D или 3D
Collision Quality	Определяет качество мирового столкновения. Принимает значения **High**, **Medium** или **Low**. Очевидно, что значение **High** наиболее ресурсоемко и наиболее точно, а **Low** — наоборот
Collides With	Определяет, с какими слоями сталкиваются частицы. Значение по умолчанию — **Everything**
Max Collision Shapes	Определяет количество форм, которые могут быть рассмотрены на предмет столкновения. Лишние формы игнорируются. Приоритет отдается ландшафту
Enabled Dynamic Colliders	Определяет, могут ли частицы сталкиваться с нестатичными (некинематическими) коллайдерами
Collider Force и **Multiply Options**	Позволяет частицам оказывать силовое воздействие на объекты при столкновении, то есть давить на них. Дополнительные оции позволяют применять силу в зависимости от угла падения, скорости и размера частицы

ПРАКТИКУМ

Настройка столкновения частиц

Теперь мы настроим столкновения с системой частиц. В упражнении используются режимы **Plane** и **World**. Выполните следующие действия.

1. Создайте новый проект или сцену. Добавьте на сцену сферу и поместите ее в позицию с координатами (0, 5, 0) и масштабом (3, 3, 3). Добавьте к сфере компонент **Rigidbody**.
2. Добавьте на сцену систему частиц и поместите ее в позицию с координатами (0, 0, 0). В модуле **Emission** на панели **Inspector** присвойте параметру **Rate over Time** значение **100**.
3. Включите модуль **Collision**, щелкнув мышью по кругу рядом с названием модуля. Присвойте параметру **Type** значение **World**, а параметру **Collider Force** — значение **20** (рис. 16.6). Обратите внимание, что частицы уже начинают отскакивать от сферы.

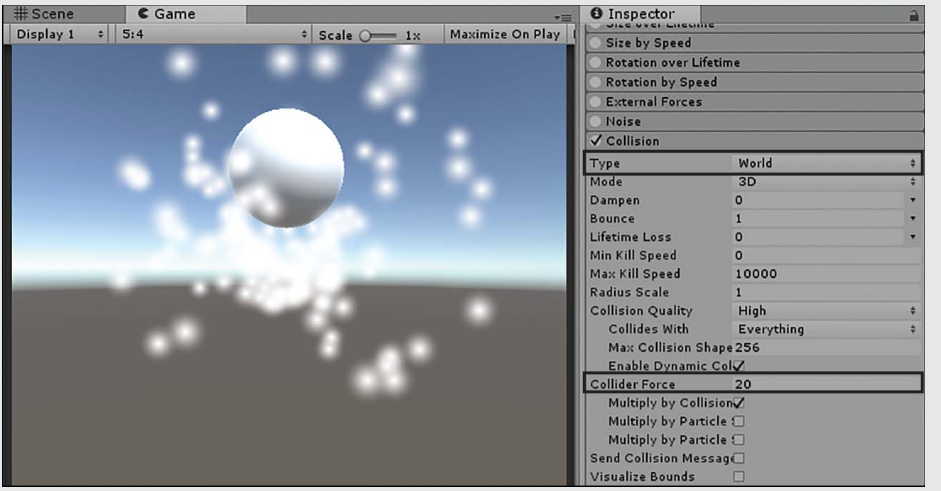


Рис. 16.6. Добавление преобразования плоскости

4. Перейдите в режим воспроизведения и обратите внимание, как частицы огибают сферу.
5. Поэкспериментируйте с различными параметрами излучения, формы и столкновения. Попробуйте как можно дольше удерживать сферу в воздухе.

СОВЕТ

Излучатель и настройки частиц

Некоторые модули изменяют излучатель, а другие изменяют частицы. Вы можете задаться вопросом — для чего предназначены свойства **Color** и **Color over Lifetime**. Одно из них управляет цветом в течение срока жизни излучателя, а второе меняет цвет частиц на протяжении их жизни.

Модуль Triggers

Модуль **Triggers** «включает» отклик на попадание частиц в коллайдер. Вы можете задать события на случай попадания частицы внутрь объема, выхода из объема, а также нахождения внутри и снаружи. В любом из этих случаев вы можете проигнорировать событие, уничтожить частицу или вызвать метод в каком-то коде и задать пользовательское поведение.

Модуль Sub Emitter

Модуль **Sub Emitter** невероятно мощный. Он позволяет породить новую систему частиц при наступлении определенных условий для каждой частицы имеющейся системы. Вы можете создавать новую систему частиц каждый раз, когда частица возникает, умирает или сталкивается с чем-нибудь. Благодаря этому вы можете воспроизводить сложные эффекты, такие как фейерверки. У модуля есть три свойства: **Birth**, **Death** и **Collision**. Каждое из них имеет системы частиц (или не имеет их), которые создаются при соответствующих событиях.

Модуль Texture Sheet

Модуль **Texture Sheet** позволяет изменять координаты текстуры, используемой для частицы, в течение ее срока жизни. По сути, это означает, что вы можете поместить на частицу несколько текстур в одном изображении, а затем переключаться между ними во время жизни частицы (мы делали подобное с анимацией спрайтов в часе 15). В таблице 16.10 описаны свойства модуля **Texture Sheet**.

ТАБЛ. 16.10. Свойства модуля Texture Sheet

Свойство	Описание
Mode	Определяет, будет ли использоваться стандартный метод листа текстуры или отдельный спрайт для прокрутки
Tiles	Определяет количество тайлов, на которые делится текстура в направлении оси *x* (по горизонтали) и *y* (по вертикали)
Animation	Определяет, содержит ли все изображение текстуры для частицы или только одну строку
Cycles	Определяет скорость анимации
Flip U и **Flip V**	Определяет, будут ли данные частицы отражены по горизонтали или по вертикали

Модуль Lights

Модуль **Lights** позволяет частицам иметь точечный источник света. Таким образом, системы частиц добавляют освещение на сцену (вспомним эффект факела, например). Большинство свойств этого модуля интуитивно понятны, но отдельно стоит рассмотреть свойство **Ratio**.

Значение 0 предполагает, что никакие частицы не будут «светиться», а значение 1 — что светиться будут все. Важно отметить, что добавление света к слишком большому количеству частиц значительно снизит производительность, так что используйте этот модуль аккуратно.

Модуль Trails

Модуль **Trails** позволяет частицам оставлять за собой след, что отлично подходит для вытянутых эффектов, таких как фейерверки или молнии. Почти все свойства этого модуля были уже рассмотрены или интуитивно понятны. Единственное, на что следует обратить внимание, — это свойство **Minimum Vertex Distance**. Оно определяет, как далеко частица должна пролететь, прежде чем у следа появится новая вершина. Более низкие значения позволяют получить более медленный, но плавный след.

Модуль Custom Data

Модуль **Custom Data** не описывается в этой книге, поскольку он выполняет сугубо техническую и очень мощную функцию. По существу, он позволяет передавать данные из системы частиц в пользовательский шейдер, который пользователь пишет сам.

Модуль Renderer

Модуль **Renderer** определяет, как частицы отрисовываются. Именно здесь вы можете указать текстуру, используемую для частиц, и другие свойства отрисовки. В таблице 16.11 описаны некоторые свойства модуля **Renderer**.

ТАБЛ. 16.11. Свойства модуля Renderer

Свойство	Описание
Render Mode	Определяет, как отрисовываются частицы. Доступные режимы: **Billboard**, **Stretched Billboard**, **Horizontal Billboard**, **Vertical Billboard** и **Mesh**. Все режимы-билборды настраивают так, чтобы частица поворачивалась к камере или по двум осям. Режим **Mesh** отрисовывает частицы в 3D, как задается мешем
Normal Direction	Определяет, насколько точно частицы «смотрят» в камеру. Значение 1 поворачивает частицы прямо в камеру
Material и **Trail Material**	Задает материал, используемый для отрисовки частицы и ее следа соответственно
Sort Mode	Определяет порядок, в котором отображаются частицы. Доступные значения: **None**, **By Distance**, **Youngest First** и **Oldest First**
Sorting Fudge	Определяет порядок, в котором отрисовываются системы частиц. Чем ниже значение, тем больше вероятность того, что система отрисуется поверх других частиц

ТАБЛ. 16.11. Свойства модуля Renderer. *Продолжение*

Свойство	Описание
Min Particle Size и Max Particle Size	Задает наименьший или наибольший размер частиц (независимо от других параметров), выраженный в виде доли от размера области просмотра. Обратите внимание, что эта настройка применяется только тогда, когда свойству **Rendering Mode** присвоено значение **Billboard**
Render Alignment	Определяет, выровнены ли частицы с камерой, миром, своими собственными преобразованиями или прямым положением камеры (полезно для VR)
Pivot	Определяет пользовательскую точку поворота для частиц
Masking	Определяет, взаимодействуют ли частицы с 2D-маской
Custom Vertex Stream	В сочетании с модулем **Custom Data** определяет, какие свойства системы частиц передаются в пользовательские шейдеры
Cast Shadows	Определяет, отбрасывают ли частицы тень
Receive Shadows	Определяет, падает ли тень других объектов на частицы
Motion Vectors	Определяет, будут ли частицы использовать векторы движения для отображения. Пока оставьте это свойство по умолчанию
Sorting Layer и Order in Layer	Разрешает частицам быть отсортированными с использованием системы сортировки спрайтов слоя
Light and Reflection Probes	Разрешает частицам работать с зондами света и отражениями (если они существуют)

Редактор Curves Editor

Несколько значений в перечисленных выше модулях могут задаваться константой или кривой. С константой все понятно: вы задаете конкретное значение. А если требуется, чтобы значение изменялось в течение определенного периода времени? И вот тут пригодятся кривые. Эта функция предоставляет очень точный уровень контроля над поведением значения. Редактор **Curves Editor** находится в нижней части панели **Inspector** (рис. 16.7). Возможно, вам придется прокрутить панель вниз.

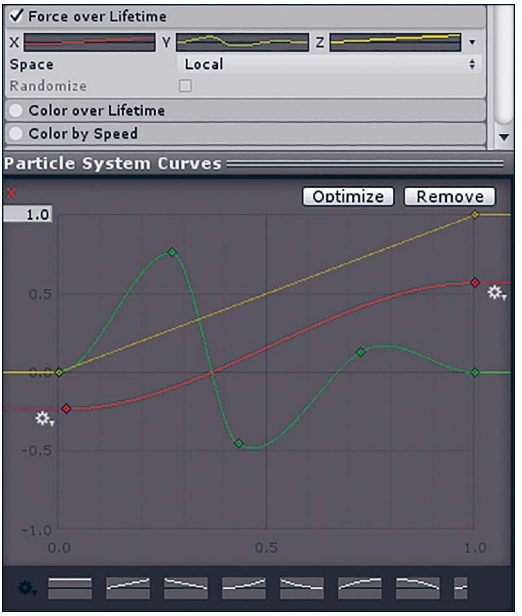


Рис. 16.7. Редактор **Curves Editor** на панели **Inspector**

Название кривой может быть любым. На рис. 16.7 показано значение силы, приложенной вдоль оси *x* в модуле **Force over Lifetime**. Диапазон задает минимальные и максимальные доступные значения. Он может быть изменен в большую или меньшую сторону. Кривая — это, по сути, запись значений в разные моменты времени, при этом можно настроить несколько основных форм кривой.

Кривая может перемещаться в любой из ключевых точек. По умолчанию их установлено две: одна в начале и одна в конце. Вы можете добавить новую ключевую точку в любом месте кривой, щелкнув правой кнопкой мыши и выбрав команду **Add Key Point** или дважды щелкнув мышью по кривой.

Вы можете развернуть окно редактора, нажав кнопку **Open Editor** в правом верхнем углу компонента **Particle System** или щелкнув правой кнопкой мыши по строке заголовка редактора **Curves Editor**.

ПРАКТИКУМ

Использование редактора Curves Editor

Чтобы познакомиться с редактором **Curves Editor**, в этом упражнении мы будем изменять размер испускаемых частиц в продолжение одного цикла системы частиц. Выполните следующие действия.

1. Создайте новый проект или сцену. Добавьте систему частиц и установите ее в позицию с координатами (0, 0, 0).
2. В раскрывающемся списке рядом со свойством **Start Size** выберите пункт **Curve**.
3. Смените диапазон кривой с 1,0 на **2,0**, изменив значение в верхнем левом углу редактора **Curves Editor**.
4. Щелкните правой кнопкой мыши по кривой приблизительно в середине и добавьте ключевую точку. Теперь перетащите начальную и конечную точки кривой в позицию 0 (рис. 16.8). Обратите внимание, что размер частиц изменяется в течение 5 секунд.

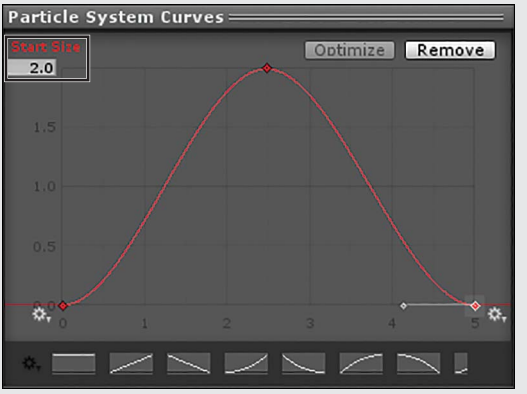


Рис. 16.8. Настройки кривой размера

Резюме

В этом часе мы изучили основы частиц и систем частиц в Unity. Мы также узнали о многих модулях, входящих в системы частиц в Unity. В конце часа мы ознакомились с работой редактора **Curves Editor**.

Вопросы и ответы

Вопрос: **Могут ли системы частиц негативно сказываться на производительности?**

Ответ: Зависит от настроек, которые вы зададите. По опыту, использовать системы частиц нужно только в случае, если они несут определенную ценность. Системы частиц могут быть очень красивыми, но не переусердствуйте.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Как называются 2D-изображения, которые всегда направлены в камеру?
2. Как развернуть окно редактора частиц?
3. Какой модуль определяет отрисовку частицы?
4. Верно или нет: редактор **Curves Editor** используется для создания кривых изменения значений во времени.

Ответы

1. Билборд.
2. Нажмите кнопку **Open Editor** в верхней части компонента **Particle System** на панели **Inspector**.
3. Модуль **Renderer**.
4. Верно.

Упражнение

В этом упражнении мы поэкспериментируем с некоторыми крутыми эффектами частиц, предоставляемыми в качестве стандартных пакетов в Unity. Вы сможете попрактиковаться с существующими эффектами и создать собственные. Правильного решения у упражнения нет; следуйте инструкциям и используйте свое воображение.

1. Импортируйте пакет эффектов частиц, выбрав команду **Assets** ⇒ **Import Package** ⇒ **ParticleSystems**. Не забудьте оставить все ассеты выбранными и нажмите кнопку **Import**.
2. Перейдите в папку *Assets\Standard Assets\ParticleSystems\Prefabs*. Перетащите префабы **FireComplex** и **Smoke** на панель **Hierarchy**. Поэкспериментируйте с позиционированием и настройкой этих эффектов. Нажмите кнопку **Play**, чтобы увидеть, что получится.
3. Продолжайте экспериментировать с остальными имеющимися эффектами частиц (точно стоит проверить взрыв и фейерверк).

4. Теперь, когда вы увидели, на что способны частицы, попробуйте создать что-то свое. Испытайте различные модули и придумайте собственные пользовательские эффекты.

17-Й ЧАС
Анимация

Что вы узнаете в этом часе

- Требования, предъявляемые к анимации
- Различные типы анимаций
- Как создавать анимацию в Unity

В этом часе вы узнаете об анимации в Unity. Мы начнем с изучения того, что такое анимация и что требуется для ее работы. Далее мы рассмотрим различные виды анимаций. Затем вы узнаете, как создавать собственные анимации с помощью инструментов Unity.

Основы анимации

Анимация — это готовый набор визуальных движений. В 2D-играх анимация представляет собой несколько последовательно показываемых изображений, которые быстро сменяются, создавая видимость движения (как в книжке, где на каждой страничке нарисован отдельный кадр). Анимация в 3D-мире сильно отличается. В 3D-играх можно использовать модели для представления игровых объектов, но вы не можете просто переключаться между ними, чтобы создать иллюзию движения. Вместо этого нужно перемещать части модели, а здесь необходимы скелет и анимация. Кроме того, анимацию можно также рассматривать как «автоматику», то есть вы можете использовать ее для автоматизации изменения свойств объектов, таких как размер коллайдеров, значения переменных в скриптах или даже цвет материалов.

Скелет

Создание сложных анимированных действий, таких как ходьба, невозможно (или невероятно сложно) без скелета. Если нет скелета, компьютер не будет знать, какие части модели должны двигаться и в какую сторону они могут, а в какую нет. *Скелет модели* — это примерно то же самое, что и человеческий скелет

(рис. 17.1), — он определяет жесткие части модели (*кости*), а также те, которые могут изгибаться — *сочленения*.

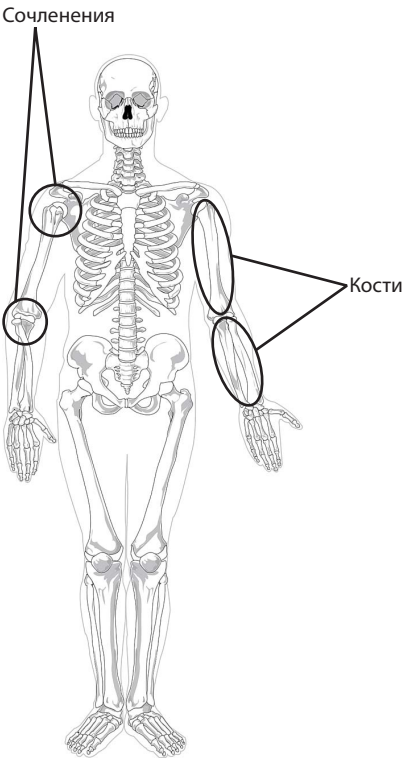


Рис. 17.1. Пример скелета

Кости и сочленения определяют физическую структуру модели. Именно она используется для анимирования модели. Стоит отметить, что для 2D-анимации, простой анимации и анимации простых объектов не требуется сложных скелетов.

Анимация

Если у модели есть скелет (или нет, в случае простой анимации), она может быть анимирована. На техническом уровне, анимация представляет собой всего лишь группу инструкций для свойства или скелета. Эти инструкции воспроизводятся как видеоролик. Их даже можно ставить на паузу, пропускать и воспроизводить в обратном порядке. Кроме того, имея подходящий скелет, вы сможете изменить действие модели путем изменения анимации. И самое крутое здесь то, что если у вас есть две совершенно разные модели с одинаковым скелетом (хотя подойдет и другой, но похожий, как вы узнаете в часе 18), вы можете применить одну

и ту же анимацию к обеим. Таким образом, орк, человек, гигант и оборотень могут, например, танцевать один и тот же танец.

ПРИМЕЧАНИЕ

Нужны ли 3D-художники

Страшная правда о 3D-анимации заключается в том, что бóльшая часть работы выполняется не в Unity. Строго говоря, моделирование, текстурирование, скелеты и анимация создаются отдельными специалистами, именуемыми *3D-художниками*, в таких программах, как Blender, Maya и 3ds Max. Такая работа требует серьезных навыков и практики и в этой книге не рассматривается. Зато вы узнаете, как делать интерактивные объекты в Unity, используя уже готовые ассеты. Помните, что создание игры — это нечто большее, чем просто соединение каких-то кусочков. Вы можете создать игру, но только художники придадут ей красивый вид!

Типы анимации

Вы уже немного узнали об анимации, скелетах и автоматизации. Эти термины пока мало о чем вам говорят, но скоро вы узнаете, как они реализуются вместе и что именно вам нужно для того, чтобы использовать анимацию в игре. В этом разделе мы рассмотрим различные виды анимации, вы разберетесь, как все работает, и сможете начать создавать что-то свое.

2D-анимация

В некотором смысле 2D-анимации — простейший тип. Как объяснялось ранее в этом часе, они очень похожи на перелистывание страниц блокнота с кадрами (или на мультфильм или даже кинофильм). Суть 2D-анимации в том, что изображения появляются в последовательном порядке в очень быстром темпе, создавая иллюзию движения.

Настроить 2D-анимацию в Unity очень легко, а вот модифицировать ее впоследствии гораздо сложнее. Причина заключается в том, что для работы 2D-анимации требуются графические ассеты (отображаемые изображения). Для внесения изменений в анимацию вам (или художнику) придется редактировать исходные изображения в соответствующих программах, например Photoshop или Gimp. Вносить изменения в сами изображения средствами Unity нельзя.

ПРАКТИКУМ

Нарезка листа спрайтов для анимации

В этом упражнении мы подготовим лист спрайтов для создания анимации. Новый проект будет использоваться далее, так что не забудьте сохранить его. Выполните следующие действия.

1. Создайте новый 2D-проект.
2. Импортируйте из пакета 2D-ассетов изображение **RobotBoyRunSprite.png** (изображение уже готовое — это анимированный персонаж из 2D-ассетов, и мы займемся его исследованием). Вы можете сделать это, выбрав команду **Assets** ⇒ **Import Package** ⇒ **2D** и импортировать *только* ассет **RobotBoyRunSprite.png** (рис. 17.2). Кроме того, вы можете найти ассет *RobotBoyRunSprite.png* среди файлов примеров для часа 17.

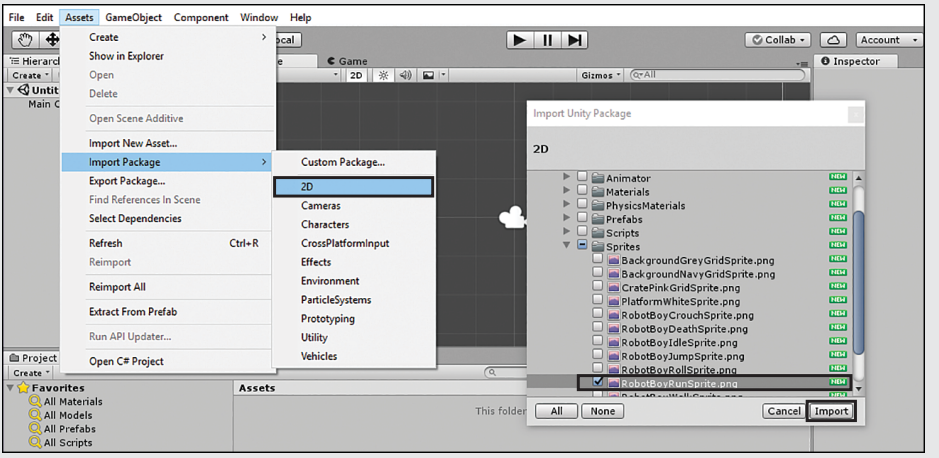


Рис. 17.2. Импорт листа спрайтов

3. Выберите импортированный файл **RobotBoyRunSprite.png** на панели **Project**.
4. На панели **Inspector** задайте свойству **Sprite Mode** значение **Multiple**, а затем нажмите кнопку **Sprite Editor**. В верхнем левом углу редактора нажмите кнопку **Slice**, а затем выберите тип нарезки **Grid by Cell Size**. Обратите внимание на размеры сетки (рис. 17.3).

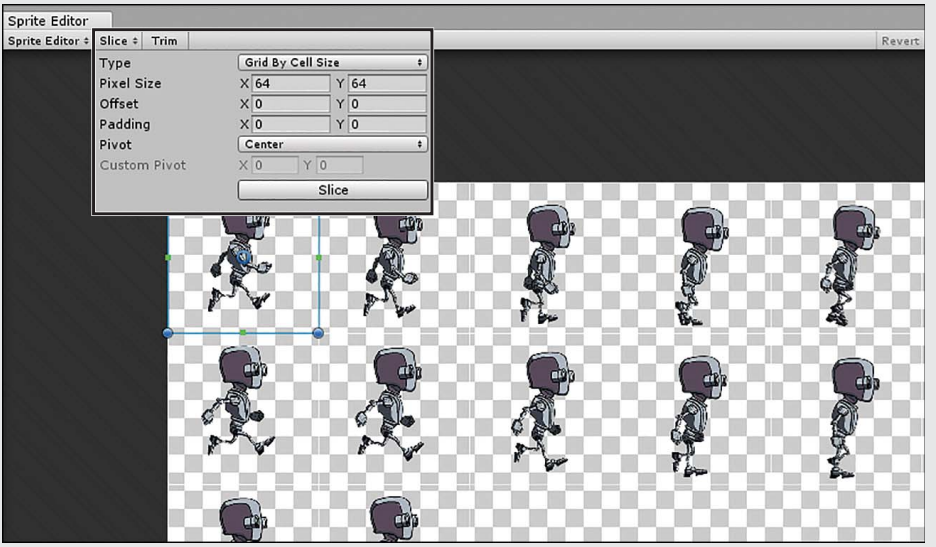


Рис. 17.3. Нарезка листа спрайтов

5. Закройте редактор **Sprite Editor**.

Теперь, когда у вас есть набор спрайтов, вы можете превратить их в анимацию.

ПРИМЕЧАНИЕ

Изобретаем колесо... еще раз

В этом часе мы будем использовать материалы из пакета 2D-ассетов. Вы, возможно, заметили, что эти ассеты уже анимированы, то есть вы повторяете работу, проделанную ранее. Благодаря этому вы можете увидеть результаты работы, которая потребовалась для создания анимации этих персонажей. Более того, вы можете исследовать готовые ассеты в пакете 2D-ассетов, чтобы выяснить, как они работают, и узнать, как на их основе можно решать более сложные задачи.

Создание анимации

Вы уже подготовили ассеты, и теперь пора превратить их в анимации. Существует два пути достижения этой цели — простой и сложный. Сложный подразумевает создание анимированного ассета с указанием свойств рендеринга спрайта, добавлением ключевых кадров, а также настройки значений. Поскольку вы пока не научились такому (хотя в следующем разделе дойдем и до этого), давайте выберем простой путь. В Unity реализован мощный автоматизированный процесс для создания анимаций, которым мы и воспользуемся.

ПРАКТИКУМ

Создание анимации

Выполните следующие действия, чтобы создать анимацию.

1. Откройте проект, созданный в практикуме «Нарезка листа спрайтов для анимации».
2. Найдите ассет **RobotBoyRunSprite** на панели **Project**. Разверните спрайт (щелкнув мышью по маленькой стрелке справа от спрайта), чтобы увидеть все кадры.
3. Выберите все спрайты из этого листа, щелкнув по первому, а затем по последнему спрайту, нажав и удерживая клавишу **Shift**. Затем перетащите все кадры на сцену (или на панель **Hierarchy**, неважно) (рис. 17.4).

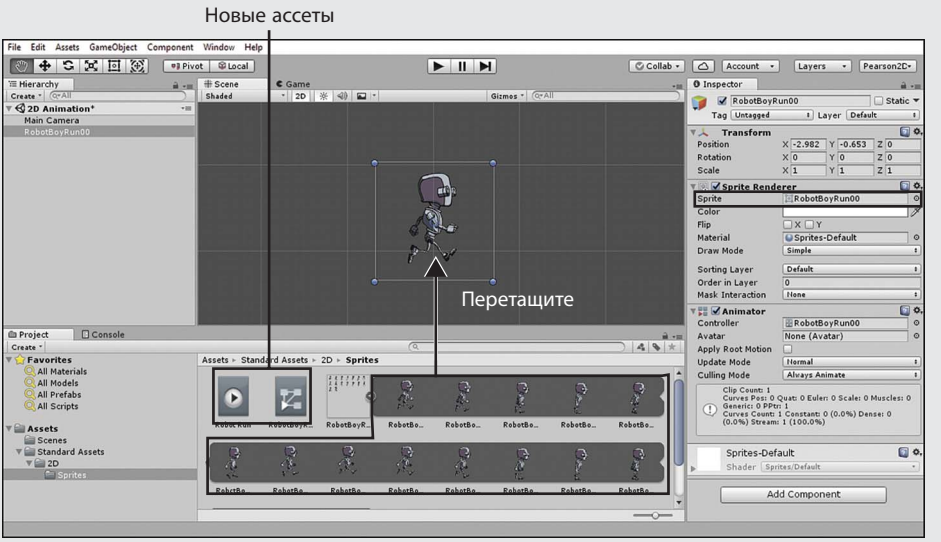


Рис. 17.4. Создание анимации

4. Если появится диалоговое окно **Save as**, укажите имя и расположение для новой анимации. Если этого не произойдет, в той же папке, где находился лист спрайтов, будет создано два новых ассета. В любом случае, программа Unity автоматически выполнит процесс создания анимированного персонажа на сцене. Два новых ассета (**animation** и **Animator controller**) будут рассмотрены более подробно в часе 18.
5. Запустите сцену, и вы увидите, как анимированный робот шагает в пространстве. На панели **Inspector** рассмотрите свойство **Sprite** компонента **Sprite Renderer**, в котором прокручиваются кадры анимации.

Вот и все! Создать 2D-анимацию, оказывается, проще простого.

ПРИМЕЧАНИЕ

Больше анимации

Теперь вы знаете, как создать одну 2D-анимацию. А как быть, если вам нужно несколько анимаций (например, для ходьбы, бега, покоя, прыжков и так далее), работающих вместе? К счастью, изученные вами основы применимы и для более сложных скриптов. Но чтобы анимации работали совместно, нужно более глубокое понимание системы анимации в Unity. Мы подробно изучим эту тему в часе 18, где будем использовать импортированную 3D-анимацию. Помните, что везде, где вы можете использовать 3D-анимацию, вы также можете использовать любой другой тип анимации. Таким образом, идеи, которые вы узнаете в часе 18, в той или иной степени применимы к 2D- и пользовательской анимациям.

Инструменты анимации

Программа Unity содержит множество инструментов, которые вы можете использовать для создания и изменения анимации, не покидая окно редактора. Вы уже применяли их, когда работали над 2D-анимацией, и теперь настало время более глубоко изучить эту тему.

Панель Animation

Чтобы использовать инструменты анимации в Unity, необходимо открыть панель **Animation**. Это можно сделать, выбрав команду **Window** ⇒ **Animation** (не путайте с компонентом **Animator**). Вы увидите новую панель: ее размер можно изменить, а ее саму — разместить в главном окне программы Unity. Как правило, удобнее закрепить эту панель так, чтобы можно было использовать ее вместе с остальными элементами управления программы. На рис. 17.5 показана панель **Animation** и ее элементы управления.

Рис. 17.5. Панель **Animation**

Обратите внимание, что на рисунке показана 2D-анимация из предыдущего упражнения. Попробуйте разобраться, как Unity использует спрайты, которые мы перетащили на сцену, чтобы создать анимацию. В таблице 17.1 перечислены некоторые из наиболее важных компонентов панели **Animation**.

ТАБЛ. 17.1. Важные элементы панели Animation

Компонент	Описание
Свойства	Список свойств компонента, изменяемых этой анимацией. Развернув свойство, вы увидите его значение в зависимости от положения ползунка на таймлайне
Частота кадров	Количество воспроизводимых кадров анимации в секунду
Таймлайн	Визуальное представление изменения свойств с течением времени. Таймлайн позволяет задать, когда будет изменяться то или иное свойство
Ключевые кадры	Особые точки на таймлайне. Ключевой кадр позволяет указать желаемое значение свойства в конкретный момент времени. Обычные кадры (которые не отображаются), также содержат значения, но вы не можете напрямую управлять ими
Добавить ключевой кадр	Кнопка, которая позволяет добавить ключевой кадр на таймлайн для выбранного свойства в положении ползунка
Ползунок	Красная линия, позволяет выбрать позицию на шкале времени, в которой нужно внести изменения. (Примечание: если режим записи выключен, ползунок становится белым)
Текущий кадр	Кадр, на котором находится ползунок
Предпросмотр	Переключатель, который позволяет просматривать анимацию. Он включается автоматически, если вы воспроизводите анимацию с использованием элементов воспроизведения на панели **Animation**
Режим записи	Переключатель, позволяющий включить/выключить режим записи. В режиме записи любые изменения выбранного объекта отражаются на анимации. Используйте его с осторожностью!
Раскрывающийся список анимаций	Меню, которое позволяет переключаться между различными анимациями, привязанными к выбранному объекту, а также создавать новые анимационные клипы

Надеюсь, эта панель дала вам более глубокое представление о том, как работает 2D-анимация. На рис. 17.5 видно, что была создана анимация под названием Robot Run. Она имеет скорость воспроизведения 12 кадров в секунду. Каждый кадр анимации содержит ключевой кадр, в котором меняется изображение для свойства **Sprite** компонента **Sprite Renderer**, тем самым изменяя внешний вид персонажа. Все это (и многое другое) делается автоматически.

Создание анимации

Теперь, когда вы познакомились с инструментами анимации, пора воспользоваться ими. Процесс создания анимации включает в себя размещение ключевых кадров, а затем определение значений для них. Значения кадров между всеми ключевыми кадрами рассчитываются автоматически из соображений плавного перехода между ними, например, чтобы объект подпрыгивал. Вы можете легко сделать это, изменяя преобразования объекта с помощью трех ключевых кадров. Первый будет иметь «низкое» значение по оси *y*, второй — «высокое», а третий — такое же «низкое», что и первый. В результате объект будет подпрыгивать вверх-вниз. Но это все слова — давайте выполним упражнение, чтобы получить лучшее представление о процессе.

ПРАКТИКУМ

Пусть объект вращается

В этом упражнении мы попробуем вращать объект. В качестве примера возьмем куб, хотя такую анимацию легко можно применить к любому объекту, результат будет таким же. Обязательно сохраните проект для дальнейшего использования.

1. Создайте новый 3D-проект.
2. Добавьте на сцену куб и расположите его в позиции с координатами (0, 0, 0).
3. Откройте панель **Animation** (выбрав команду **Window** ⇒ **Animation**) и закрепите ее в редакторе.
4. Выделив куб, вы увидите кнопку **Create** в середине панели **Animation**. Нажмите ее и, когда вам будет предложено сохранить анимацию, сохраните ее под именем **ObjectSpinAnim**.
5. Нажмите кнопку **Add property** на панели **Animation**, а затем щелкните мышью по значку **+** рядом с командой **Transform** ⇒ **Rotation** (рис. 17.6).

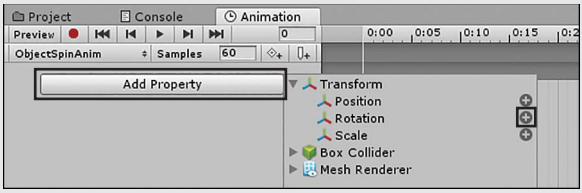


Рис. 17.6. Добавление свойства **Rotation**

Теперь вам нужно добавить к анимации два ключевых кадра: один в кадре 0, а другой в кадре 60 (или «спустя секунду» — для получения дополнительной информации о времени в анимациях смотрите следующий совет). Если вы развернете свойство **Rotation**, вы увидите свойства для отдельных осей. Кроме того, при выборе ключевого кадра вы можете просматривать и изменять его значения.

6. Переместите ползунок до последнего ключевого кадра (перетащив его на таймлайне), а затем задайте свойству **Rotation.y** значение **360** (рис. 17.7). Несмотря на то что начальное значение у вас равно 0, а конечное 360, — это один и тот же результат, поэтому куб начнет вращаться. Запустите сцену, чтобы увидеть анимацию.

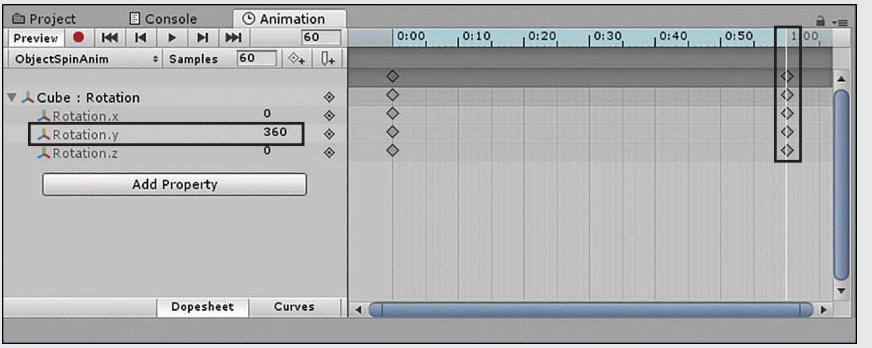


Рис. 17.7. Присвоение значения свойству **Rotation.y**

7. Сохраните сцену, так как она понадобится нам позже.

СОВЕТ

Разберемся со временем

Значения на таймлайне сначала могут показаться немного странными. На самом деле эти числа помогут вам понять скорость воспроизведения клипа. Если задана скорость воспроизведения по умолчанию — 60 кадров в секунду — таймлайн будет рассчитывать кадры от 0 до 59, а не 60, за время 1:00 (в течение 1 секунды). Таким образом, время 1:30 будет означать 1 секунду и 30 кадров. При скорости 60 кадров в секунду все просто, но что происходит при скорости, например, 12 кадров в секунду? Тогда подсчет будет таким: «1, 2, 3… 11, 12, одна секунда». Иными словами, считается «0:10, 0:11, 1:00, 1:01, 1: 02 … 1: 10, 1:11, 2:00…». Главное помнить, что перед двоеточием указана секунда, а после двоеточия — номер кадра.

СОВЕТ

Двигаем таймлайн

Вы можете увеличивать или уменьшать масштаб таймлайна, чтобы получше рассмотреть те или иные кадры. Система навигации тут такая же, как и на сцене в 2D-режиме. То есть с помощью колеса прокрутки мыши вы изменяете масштаб, а удерживая клавишу **Alt** (или **Option** в macOS), перемещаете таймлайн.

Режим записи

Инструменты, которые вы использовали до сих пор, были очень просты, но есть еще более легкие способы работы с анимацией. Еще один важный инструмент — это режим записи (на рис. 17.5 показано, где он расположен). Он записывает в анимацию любые изменения, внесенные в объект. Это очень крутой способ вносить быстрые мелкие поправки. В то же время он по-своему опасен. Подумайте, что произойдет, если вы забудете выключить режим записи и сделаете множество изменений объекта. Они запишутся в анимацию и будут постоянно воспроизводиться. Поэтому нужно всегда проверять, что режим записи выключен, если вы не хотите использовать его.

Звучит пугающе для инструмента, который вы еще даже не попробовали, но волноваться не о чем. Все на самом деле не так уж плохо. Не забывая его вовремя выключать, вы поймете, что это фантастический инструмент. В худшем случае, если Unity запишет что-то не то, вы можете вручную удалить все лишнее из анимации.

ПРАКТИКУМ

Использование режима записи

Выполните следующие действия, чтобы куб, созданный в предыдущем практикуме, начал изменять цвет по мере вращения.

1. Откройте сцену с крутящимся кубом. Создайте новый материал под названием **CubeColor** и примените его к кубу (чтобы можно было изменять цвет).
2. Выбрав куб, откройте панель **Animation**. Вы должны увидеть ранее созданную анимацию вращения. Если это не так, убедитесь, что куб выделен.
3. Перейдите в режим записи, нажав кнопку **Record Mode** на панели **Animation**. Свойство **Rotation** на панели **Inspector** станет красным: значения вращения изменяются в процессе анимации. Перетащите ползунок на таймлайне в положение 0:00 (см. рис. 17.8).

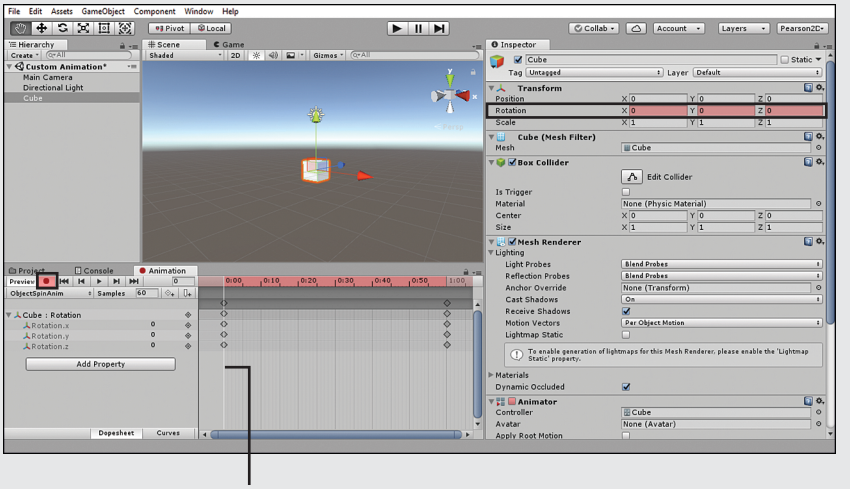


Рис. 17.8. Подготовка к записи

4. На панели **Inspector** найдите материал **CubeColor** куба и измените его цвет на красный. Обратите внимание, что на панели **Animation** появилось новое свойство и ключевой кадр. Если развернуть новое свойство, вы увидите значения цвета в формате RGBA в данном ключевом кадре (1, 0, 0, 1).
5. Переместите ползунок в дальнюю часть таймлайна и снова измените цвет. Вы можете выполнить этот шаг столько раз, сколько захотите. Если вам нужна плавная смена цвета, убедитесь, что последний ключевой кадр (в положении 1:00, для синхронизации с вращением) имеет тот же цвет, что и первый (рис. 17.9).

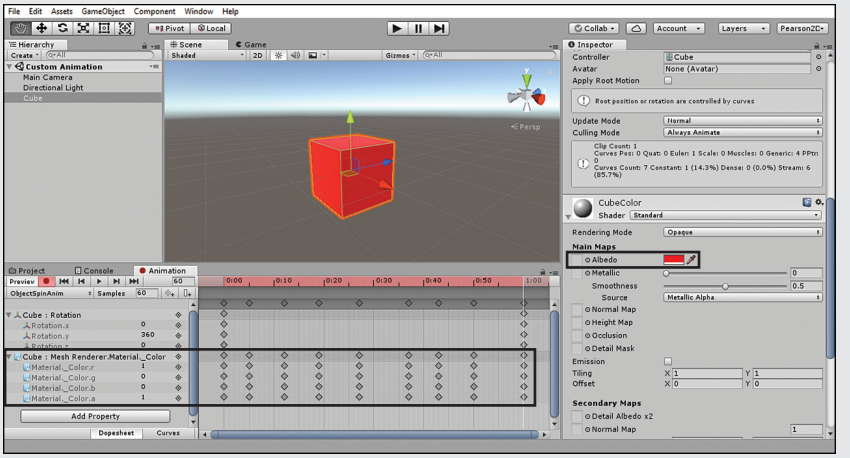


Рис. 17.9. Запись значений цвета

6. Запустите сцену, чтобы вопроизвести анимацию на панели **Game**. Обратите внимание, что запуск сцены отключает режим записи (что очень удобно, потому что мы закончили в нем работать). Теперь вы должны увидеть вращающийся и меняющий цвета куб.

Редактор Curves Editor

Последний инструмент, который мы рассмотрим в этом часе, — редактор **Curves Editor**. До сих пор мы использовали режим **Dopesheet** на панели **Animation**, представляющий собой список ключевых кадров, разбитых по категориям. Вы, возможно, заметили, что, указывая значения ключевых кадров, вы никак не управляете значениями свойств между ними. Если у вас возникал вопрос, как определяются эти значения, то вот и ответ: Unity автоматически рассчитывает значения между ключевыми кадрами (методом *интерполяции*) для создания плавного перехода. В редакторе **Curves Editor** такие переходы прекрасно видны. Чтобы открыть редактор **Curves Editor**, нажмите кнопку **Curves** в нижней части панели **Animation** (рис. 17.10).

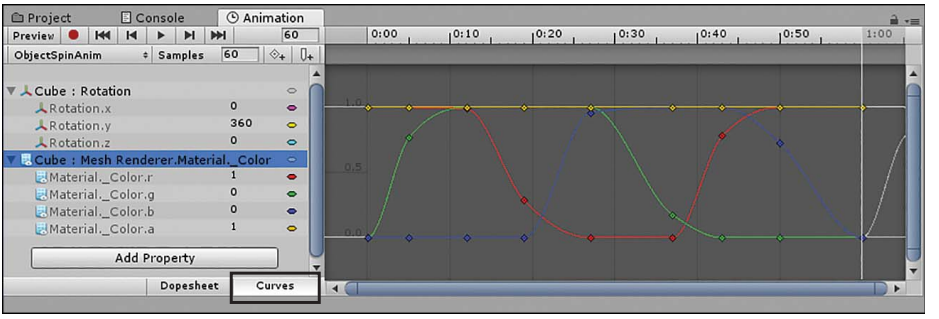


Рис. 17.10. Редактор **Curves Editor**

В этом режиме вы можете выбирать нужные вам значения, щелкая мышью по их свойствам слева. В редакторе **Curves Editor** отображается, как именно значения меняются между ключевыми кадрами. Вы можете перетащить ключевой кадр, чтобы изменить его значение, и даже дважды щелкнуть мышью по кривой, чтобы создать новый ключевой кадр в этой позиции. Если вам не нравится, как Unity генерирует значения между ключевыми кадрами, вы можете щелкнуть правой кнопкой мыши по ключевому кадру и выбрать команду **Free Smooth**. Появятся два маркера, которые можно перемещать, регулируя значения кривых. Попробуйте создать всякие безумства с помощью кривых.

ПРАКТИКУМ

Использование редактора Curves Editor

Вы наверняка заметили, что куб из предыдущего практикума вращается не плавно, а резко разгоняется и так же резко тормозит. В этом упражнении мы отредактируем анимацию, чтобы куб двигался более плавно. Выполните следующие действия.

1. Откройте сцену с вращающимся кубом из практикума «Использование режима записи». На панели **Animation** нажмите кнопку **Curves**, чтобы переключиться на редактор **Curves Editor** (см. рис. 17.10).
2. Щелкните мышью по свойству **Rotation.y**, чтобы вывести кривую на таймлайн. Если кривая слишком мала или, наоборот, не помещается на панели, установите указатель мыши на таймлайн и нажмите клавишу **F**.
3. Выпрямив кривую, вы получите красивую плавную анимацию куба (рис. 17.11). Для этого щелкните правой кнопкой мыши по первому ключевому кадру (в позиции 0:00) и выберите команду **Auto**. Сделайте то же самое для последнего ключевого кадра.

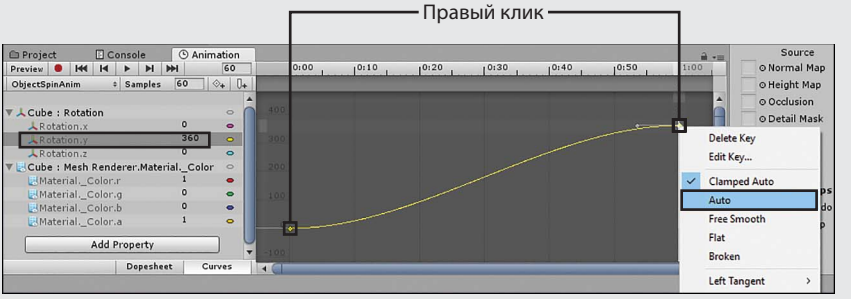


Рис. 17.11. Изменение кривой вращения

4. Теперь, когда кривая выпрямлена, перейдите в режим воспроизведения, чтобы увидеть новую анимацию.

Резюме

В этом часе мы изучили основы анимации в Unity. Мы разобрали основные понятия, включая скелеты. Также вы узнали о различных типах анимации в Unity. После этого мы создали несколько анимаций на основе 2D-анимации. Затем мы попробовали создать пользовательские анимации как вручную, так и в режиме записи.

Вопросы и ответы

Вопрос: **Можно ли создать смешанную анимацию?**

Ответ: Да, можно. Вы можете смешивать анимации с помощью системы Unity Mecanim, которую мы рассмотрим в часе 18.

Вопрос: **Может ли анимация быть применена к любой модели?**

Ответ: Только если у нее будет такой же скелет. Но, если анимация простая, скелет не требуется. В противном случае анимация может вести себя очень странно или попросту не работать.

Вопрос: **Можно ли задать для модели новый скелет в Unity?**

Ответ: В некотором смысле (см. час 18).

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Как называется «каркас» модели?
2. Какой тип анимации похож на быструю прокрутку изображений?
3. Как называется кадр анимации, для которого явно задано значение?

Ответы

1. Скелет.
2. 2D-анимация.
3. Ключевой кадр.

Упражнение

Это упражнение — что-то вроде «песочницы». Выделите время, чтобы привыкнуть к созданию анимации. В Unity есть очень мощный набор инструментов, и вам, конечно, стоит с ним ознакомиться. Попробуйте сделать следующее.

1. Сделайте так, чтобы объект летал по сцене по большой дуге.
2. Сделайте так, чтобы объект мерцал, включая или выключая его рендерер.
3. Попробуйте изменять свойства масштаба и материала объекта, чтобы он преобразовывался.

18-Й ЧАС
Аниматоры

Что вы узнаете в этом часе

- Основы аниматоров
- Как использовать конечные автоматы аниматоров
- Как управлять анимацией с помощью скриптов и параметров
- Основы деревьев смешивания

В этом часе мы применим уже имеющиеся знания об анимации и воспользуемся ими для работы с системой анимации Mecanim и аниматорами. Вы начнете с изучения аниматоров и того, как они работают. Затем вы узнаете, как задать или изменить скелет модели в Unity. После этого мы создадим и настроим аниматор. Наконец, мы посмотрим, как смешивать анимации для получения невероятно реалистичных результатов.

ПРИМЕЧАНИЕ

Предупреждение: придется пачкать руки!

Весь этот раздел представляет собой один большой практикум. Не забывайте сохранять свой проект в конце каждого упражнения, так как он вам понадобится далее. Разумеется, вам нужно находиться за компьютером. Лучший способ изучить материал — это практика!

Основы аниматоров

Все анимации в Unity создаются с помощью компонента **Animator**. В часе 17 мы уже создавали, изучали анимации и даже использовали аниматоры, сами того не зная. По сути, система анимации Unity (Mecanim) состоит из трех частей: анимационный клип, контроллер **Animator** и компонент **Animator**. Эти три компонента заставляют ваших персонажей оживать.

На рис. 18.1, позаимствованном из раздела документации Unity, касающегося компонента **Animator** (**docs.unity3d.com/ru/current/Manual/class-Animator.html**), показано, как эти части связаны друг с другом.

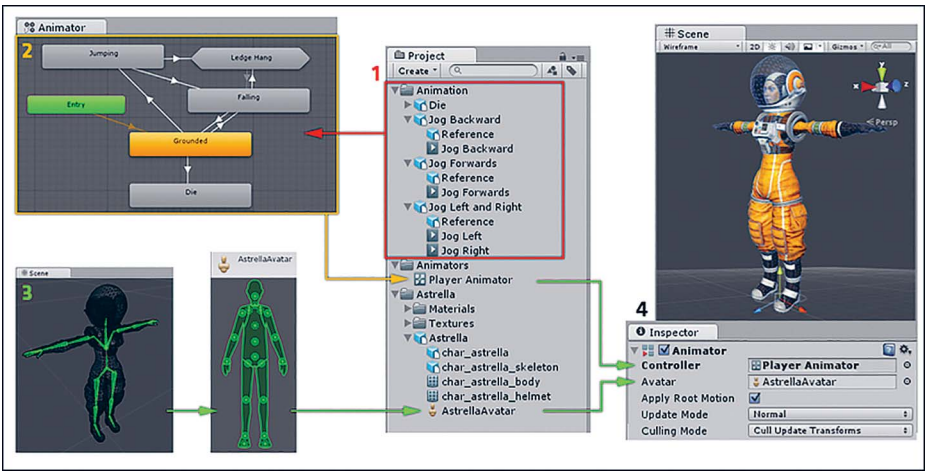


Рис. 18.1. Как части анимации гуманоида связаны друг с другом

Анимационные клипы (1 на рис. 18.1) представляют собой различные движения, которые вы можете создать в Unity или импортировать туда. Контроллер **Animator** (2) содержит анимацию и определяет, какие клипы должны воспроизводиться в данный момент. У моделей есть так называемый аватар (3), который играет роль «переводчика» между контроллером **Animator** и скелетом модели. Вы можете вообще не обращать внимания на аватар, поскольку он настраивается и используется автоматически. Наконец, контроллер **Animator** (лучше называть его просто «контроллер») и аватар совместно используются в модели с помощью компонента **Animator** (4). Многовато информации, да? Не волнуйтесь. Бóльшая ее часть интуитивно понятна либо настраивается автоматически.

ПРИМЕЧАНИЕ

Анализ конкретных ситуаций

Чтобы извлечь максимальную пользу из этого часа, мы поработаем с очень специфическим случаем: 3D-анимацией модели гуманоида (довольно распространенный пример для практики). Так вы узнаете много нового о 3D-анимациях, импорте моделей и их анимировании, о работе со скелетами и системой переназначения Unity. Не забывайте, что все, что мы рассмотрим в этом часе (за исключением применения анимаций к другим скелетам), полностью подходит и для других типов анимации. Так что, если вы захотите создать многослойную систему 2D-анимации, все полученные знания будут вам полезны.

Очень полезная особенность системы анимации Unity заключается в том, что вы можете применить одну и ту же анимацию к разным игровым объектам. Если вы анимируете куб, вы можете применить ту же анимацию к шару. Если вы анимируете персонажа, вы можете применить эту анимацию к другому персонажу с таким же скелетом (или даже с другим, как вы скоро увидите). Это означает, что вы можете, например, наделить орка и человека способностью танцевать один и тот же танец.

Немного о скелетах

Чтобы создать сложную систему анимации, сначала необходимо подготовить скелет модели. В часе 17 было сказано, что модели и анимации должны соответствовать друг другу, чтобы анимация работала. Это означает, что иногда очень трудно переложить анимацию, сделанную для одной модели, на другую. Таким образом, анимация и модель, как правило, создаются одновременно и друг для друга.

Если вы используете модель гуманоида (две руки, две ноги, голова и туловище), у вас есть возможность применить инструменты переназначения системы Mecanim. С помощью этой системы можно перенастроить скелет гуманоида в редакторе без каких-либо инструментов 3D-моделирования. Суть заключается в том, что любая анимация, созданная для гуманоидных моделей, может работать с любыми гуманоидами. Это означает, что аниматоры (люди, а не ассеты Unity) могут собирать крупные библиотеки анимации, которые можно применять к широкому диапазону моделей с множеством различных настроек.

Импорт модели

В этом часе мы будем работать с Итаном, моделью из пакета ассетов **Characters**. У этой модели есть множество различных элементов, и мы рассмотрим каждый из них, чтобы убедиться, что он настроен правильно. Чтобы импортировать модель, выберите команду **Assets** ⇒ **Import Package** ⇒ **Characters**. Оставьте все флажки установленными и нажмите кнопку **Import**.

Теперь найдите Итана на панели **Project** в папке **Assets\Standard Assets\Characters\ ThirdPersonCharacter\Models** (см. рис. 18.2).

Если вы щелкнете мышью по маленькой стрелке справа от Итана, вы можете развернуть модель и увидеть все ее составные части (см. рис. 18.2). Как структурированы эти части, зависит от того, как модель была экспортирована из приложения для 3D-моделирования, в котором ее создавали.

Компоненты слева направо: тело Итана с текстурой, текстурированные очки, определение скелета, исходный меш EthanBody, исходный меш EthanGlasses и, наконец, определение аватара Итана (используется для создания скелета).

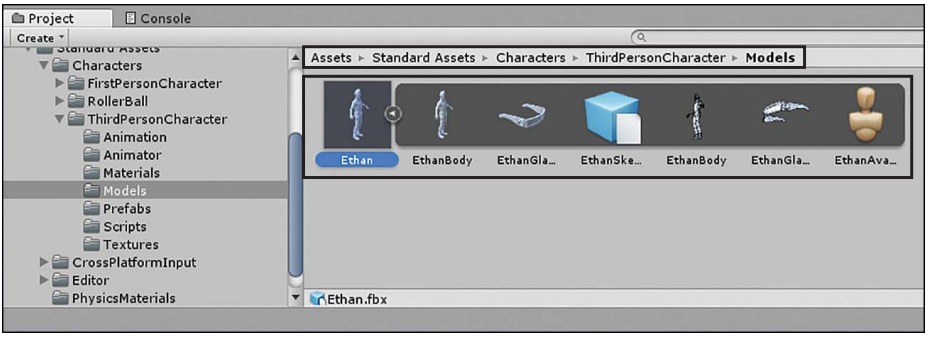


Рис. 18.2. Модель Итана

ПРИМЕЧАНИЕ

Предварительный просмотр мешей

Щелкнув мышью по модели Итана или очков, вы увидите окошко предпросмотра в нижней части панели **Inspector**. (Если нет, измените положение, чтобы оно появилось.) Здесь вы можете покрутить модель, чтобы посмотреть на нее со всех сторон (см. в нижней части рис. 18.3).

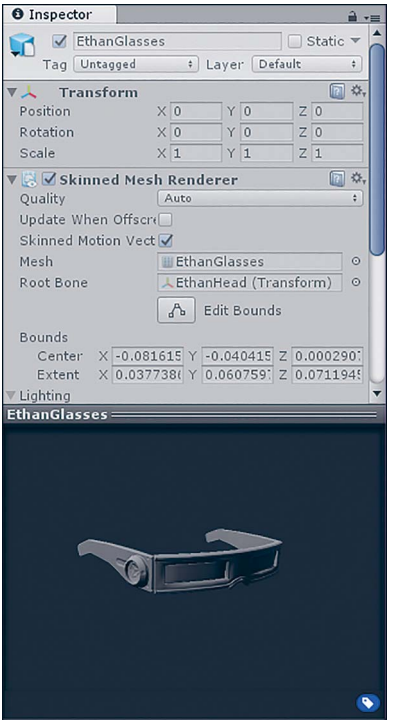


Рис. 18.3. Предпросмотр модели

Закончив разглядывать компоненты, сверните модель **Ethan.fbx**, щелкнув мышью по стрелке справа от ассета.

Настройка ассетов

Теперь, когда вы импортировали модель и анимацию (которая тоже импортировалась вместе с остальными ассетами), вам нужно настроить их. Процесс настройки анимации весьма похож на настройку моделей.

Выбрав модель, вы увидите список параметров импорта на панели **Inspector**. На вкладке **Models** приведены все параметры, которые настраивают импорт модели в Unity, но их пока можно не трогать. Сейчас нас интересует вкладка **Rig**. (Вкладку **Animations** мы обсудим чуть позже в этом часе).

Подготовка скелета

Настройка скелета выполняется в настройках импорта, на вкладке **Rig** панели **Inspector**. Здесь нас больше всего интересует свойство **Animation Type** (рис. 18.4). В раскрывающемся списке вы увидите четыре пункта: **No**, **Legacy**, **Generic** и **Humanoid**. Значение **None** позволяет Unity игнорировать скелет этой модели. Значение **Legacy** осталось от старой системы анимации Unity, оно нам не нужно. Значение **Generic** используется для всех негуманоидных моделей (простые модели, транспортные средства, здания, животные и т. д.), а также всех моделей, импортируемых в Unity по умолчанию для данного типа анимации. Наконец, значение **Humanoid** (вариант, который вы будете использовать) применяется для гуманоидных персонажей. Этот параметр позволяет Unity настроить анимацию так, как вам нужно.

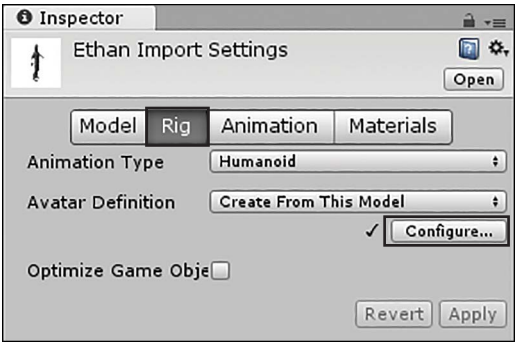


Рис. 18.4. Параметры скелета

Как вы видите, Итан-гуманоид уже настроен как полагается. Если модель гуманоидная, Unity автоматически выполняет процедуру картирования скелета. Если вы хотите убедиться в том, что это легко, вы можете присвоить свойству **Animation**

Type значение **Generic**, нажать кнопку **Apply**, а затем вернуть исходное значение (именно так эта модель была создана изначально, и никакой скрытой дополнительной работы тут нет). Чтобы увидеть работу, которую Unity делает автоматически, вы можете использовать инструмент построения скелетной иерархии, нажав кнопку **Configure** (см. рис. 18.4).

ПРАКТИКУМ

Изучение скелета Итана

В этом упражнении мы изучим скелет Итана, чтобы лучше понять, как создается модель со скелетом. Выполните следующие действия.

1. Если вы еще не сделали этого, создайте новый проект и импортируйте ассеты персонажей из пакета **Standard Assets**. Найдите ассет **Ethan.fbx** и выберите его, чтобы увидеть его настройки импорта на панели **Inspector**, как описано ранее в этом часе.
2. Нажмите кнопку **Configure** на вкладке **Rig**. Появится новая сцена. Также вы сможете сохранить старую сцену, если появится соответствующий запрос.
3. Настройте интерфейс так, чтобы видеть только панели **Hierarchy** и **Inspector**. (Как закрывать и перемещать панели, описано в часе 1.) Вы можете сохранить этот интерфейс и вернуться к дефолтному в любой момент.
4. Перейдя на вкладку **Mapping**, щелкайте мышью по зеленым кружкам (рис. 18.5). Обратите внимание, как подсвечивается соответствующий дочерний компонет **EthanSkeleton** на панели **Hierarchy** и отображается синий круг вокруг соответствующей точки скелета ниже контура. Посмотрите на все дополнительные точки на панели **Hierarchy**. Они не имеют значения для гуманоидов и, таким образом, не переназначаются. Беспокоиться не о чем. Эти точки играют определенную роль в правильном отображении модели при движении.

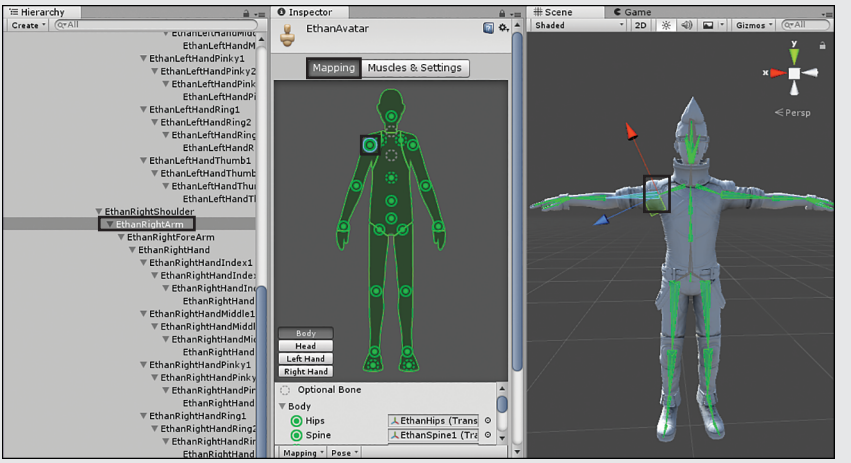


Рис. 18.5. Панель **Rigging** с выбранной правой рукой

5. Продолжайте исследовать другие части скелета, нажимая на тело, голову, левую руку и так далее. Все эти сочленения могут быть полностью перенастроены для любого гуманоида.
6. Нажмите кнопку **Done**, когда закончите. Обратите внимание, что временный Итан (клон) исчез с панели **Hierarchy**.

Теперь мы изучили Итана и увидели, как устроен его скелет. Он готов к работе!

Подготовка анимации

Здесь можно было бы использовать анимации, которыми комплектуется Итан, но это скучно и не дает прочувствовать гибкость системы Mecanim. Мы используем другие анимации, представленные в файлах примеров для часа 18. У каждой из них есть параметры, которые определяют, как анимация воспроизводится, и эти параметры можно настраивать под себя. Например, вам необходимо, чтобы анимация была правильно зациклена, чтобы у переходов не осталось видимых швов. В этом разделе мы выполним процедуру подготовки анимации.

Начнем с перетаскивания папки **Animations** из файлов примеров в редактор Unity. Мы будем работать с четырьмя анимациями: **Idle**, **WalkForwardStraight**, **WalkForwardTurnRight** и **WalkForwardTurnLeft** (хотя в папке *Animations* находятся только три файла, но скоро мы дадим пояснения). Каждая из этих анимаций должна быть настроена отдельно. Заглянув в папку *Animations*, вы увидите, что анимация — это файл с расширением *.fbx*. Это связано с тем, что сами анимации находятся внутри своих моделей по умолчанию. Но не беспокойтесь: в Unity можно их извлекать и редактировать.

Анимация Idle

Чтобы настроить анимацию **Idle** (бездействие), выполните следующие шаги (описание настроек приведено в таблице 18.1).

1. Выберите файл **Idles.fbx** в папке **Animations**. На панели **Inspector** перейдите на вкладку **Rig**. Измените тип анимации на **Humanoid** и нажмите кнопку **Apply**. Так мы сообщаем Unity о том, что анимация предназначена для гуманоида.
2. После того как скелет настроен, перейдите на вкладку **Animations** панели **Inspector**. Установите начальный кадр в позицию **128** и установите флажки **Loop Time** и **Loop Pose**. Кроме того, установите флажок **Bake into Pose** для всех свойств **Root Transform**. Убедитесь, что ваши настройки совпадают с приведенными на рис. 18.6, а затем нажмите кнопку **Apply**.
3. Чтобы проверить, что анимация теперь настроена правильно, разверните файл **Idles.fbx** (рис. 18.7). Мы уже говорили, где можно найти эту анимацию. (Модель нам не нужна — речь именно об анимации.)

ТАБЛ. 18.1. Важные параметры анимации

Настройка	Описание
Loop Time	Определяет, будет ли анимация зациклена
Root Transform	Задает, разрешено ли анимации изменять вращение объекта, вертикальное положение (ось *y*) и горизонтальное положение (плоскость *x*/*z*)
Bake into Pose	Определяет, может ли анимация перемещать объект. Если вы установите этот флажок, то анимация не будет изменять реальные свойства объекта, изменяться будет только отображение
Offset	Задает смещение, в соответствии с которым изменяется исходное положение анимации. Так, путем изменения значения смещения **Root Transform Rotation** вы вносите небольшие коррекции во вращение модели относительно оси *y*. Это полезно для исправления ошибок движения в анимации

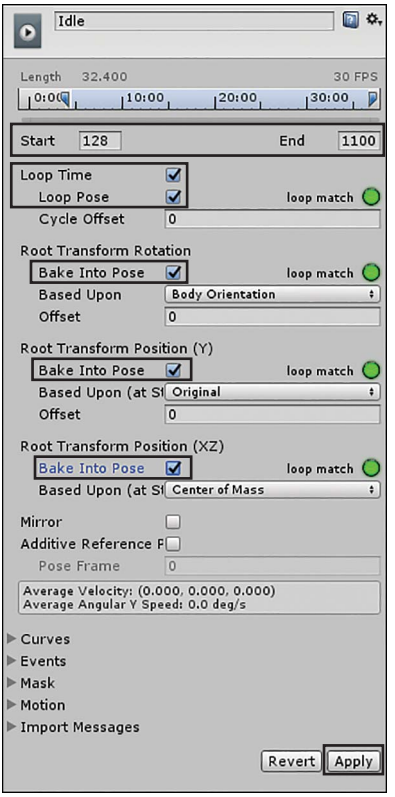


Рис. 18.6. Анимация **Idle**

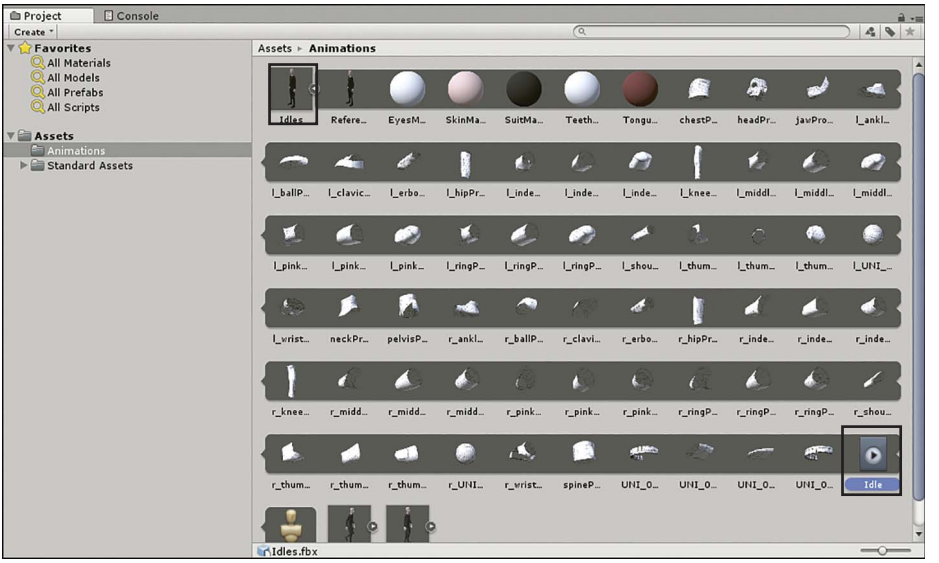


Рис. 18.7. Содержимое анимационного клипа

ПРИМЕЧАНИЕ

Красный свет, зеленый свет

Вы, возможно, заметили зеленые круги в настройках анимации (см. рис. 18.6). Это маленькие инструменты, которые помогают определить, как выстроена анимация. Дело в том, что круги зеленого цвета означают, что зацикливание будет бесшовным. Круг желтого цвета указывает на то, что анимация почти бесшовная, но именно «почти». Красный круг сообщает о том, что начало и конец анимации не совпадают, и шов будет очевидным. Если ваша анимация не зацикливается должным образом, вы можете изменить свойства **Start** и **End**, чтобы исправить положение дел.

Анимация WalkForwardStraight

Чтобы настроить анимацию **WalkForwardStraight**, выполните следующие действия.

1. Выберите файл **WalkForward.fbx** в папке **Animations** и постройте скелетную иерархию так же, как и для анимации **Idle**.
2. Измените название клипа (Take 001) на **WalkForwardStraight**, щелкнув по его имени (рис. 18.8).
3. Убедитесь, что на вкладке **Animations** ваши настройки совпадают с теми, которые показаны на рис. 18.8. Следует отметить два момента. Рядом со свойством **Root Transform Position (XZ)** вы видите красный круг. Это

хорошо, так как он означает, что в конце анимации модель находится в другом положении вдоль оси *x* и *z*. Поскольку это анимация ходьбы — нам того и надо. Еще нужно обратить внимание на индикатор **Average Velocity**. Вы наверняка заметили ненулевую скорость по осям *x* и *z*. Скорость по оси *z* как раз нам и нужна, чтобы модель двигалась вперед, а скорость по оси *x* — нет, чтобы модель не дрейфовала в сторону при ходьбе. Мы подкорректируем это на шаге 4.

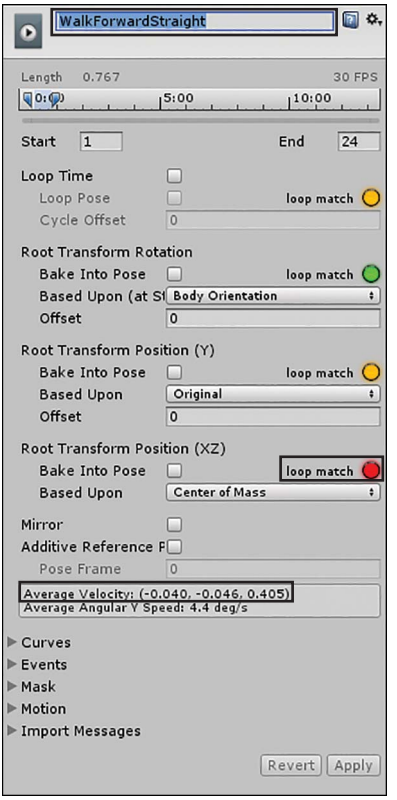


Рис. 18.8. Параметры анимации **WalkForwardStraight**

4. Чтобы отрегулировать скорость вдоль оси *x*, установите флажок **Bake in Pose** для свойств **Root Transform Rotation** и **Root Transform Position (Y)**. Кроме того, измените свойство **Root Transform Rotation Offset** таким образом, чтобы значение **X** свойства **Average Velocity** стало равным **0**.

5. Установите конечный кадр в позицию **244,9**, а начальный — в позицию **215,2** (именно в таком порядке), чтобы в анимацию вошли только кадры шагания.

6. Наконец, установите флажок **Loop Time** и **Loop Pose**.

7. Убедитесь, что ваши настройки соответствуют рис. 18.9, и нажмите кнопку **Apply**.

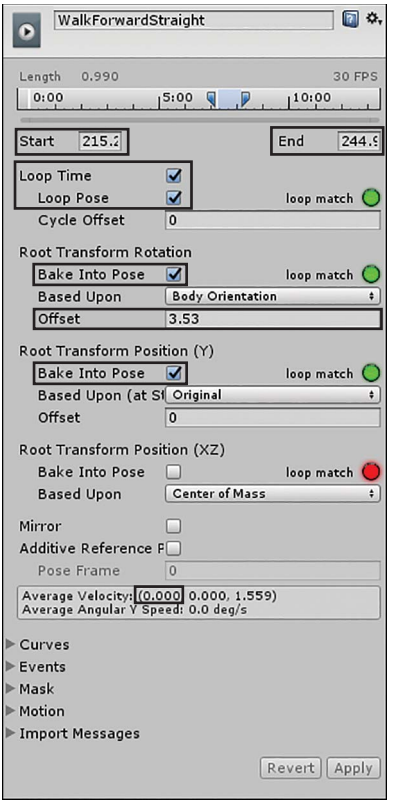


Рис. 18.9. Настройки анимации **WalkForwardStraight**

Анимация WalkForwardTurnRight

Анимация **WalkForwardTurnRight** позволяет модели плавно менять направление при ходьбе вперед. Она отличается от созданных ранее тем, что вам нужно сделать две анимации из одной. Это проще, чем кажется. Выполните следующие действия.

1. Выберите файл **WalkForwardTurns.fbx** в папке **Animations** и постройте скелетную иерархию так же, как для анимации **Idle**.

2. По умолчанию появится анимация с именем _7_a_U1_M_P_ WalkForwardTurnRight. Переименуйте ее, введя **WalkForwardTurnRight** в текстовое поле **Clip Name** и нажав клавишу **Enter**.

3. Выбрав клип **WalkForwardTurnRight**, установите его свойства, как показано на рис. 18.10. Меньшие значения времени начала и конца позволят сократить клип и оставить только движение по кругу вправо. (Не забудьте предварительно просмотреть клип.) Когда это будет сделано, нажмите кнопку **Apply**.

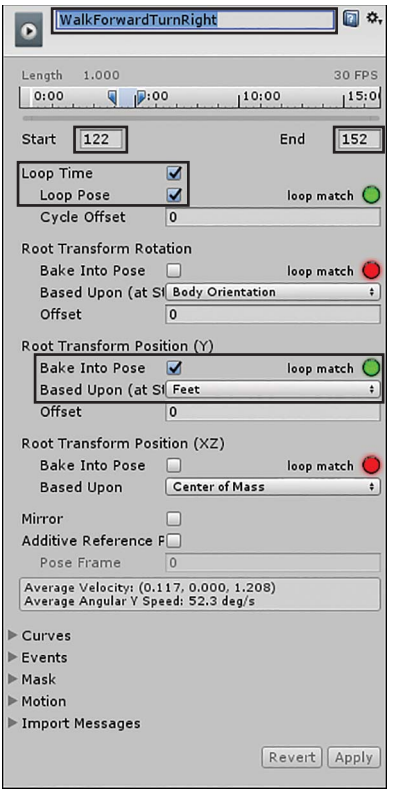


Рис. 18.10. Настройки анимации **WalkForwardTurnRight**

- Создайте анимационный клип **WalkForwardTurnLeft**, щелкнув мышью по значку + в списке клипов (рис. 18.11). Свойства клипа **WalkForwardTurnLeft** будут точно такими же, как у **WalkForwardTurnRight**, за исключением того, что вам нужно установить флажок **Mirror** (рис. 18.11). Не забудьте нажать кнопку **Apply**, когда закончите с настройками клипа.

Теперь все анимации настроены и готовы к работе. Осталось лишь создать аниматора.

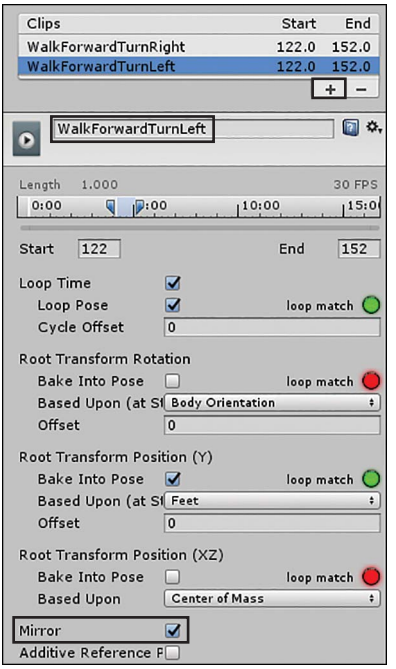


Рис. 18.11. Зеркальная анимация

Создание аниматора

Аниматоры в Unity являются ассетами, то есть они входят в состав проекта и существуют вне какой-либо конкретной сцены. Это хорошо, так как можно легко использовать аниматор снова и снова. Чтобы добавить аниматор в свой проект, нужно щелкнуть правой кнопкой мыши по папке на панели **Project** и выбрать команду **Create** ⇒ **Animator Controller** (но пока не делайте этого).

ПРАКТИКУМ

Настройка сцены

Здесь мы создадим сцену и подготовимся к остальным упражнениям этого часа. Обязательно сохраните ее, так как она понадобится нам позже. Выполните следующие действия.

1. Если вы еще не сделали этого, создайте новый проект и выполните подготовку модели и анимации, как и ранее.
2. Перетащите на сцену модель Итана (из папки **Assets\Standard Assets\Characters\ ThirdPersonCharacter\Models**) в позицию с координатами (0, 0, –5).

3. Сделайте компонент **Main Camera** дочерним для Итана (перетащив объект **Main Camera** на объект **Ethan** на панели **Hierarchy**) и расположите его в позиции с координатами (0, 1,5, –1,5) и вращением (20, 0, 0).
4. На панели **Project** создайте новую папку с именем **Animators**. Щелкните по ней правой кнопкой мыши и выберите команду **Create** ⇒ **Animator Controller**. Присвойте аниматору имя **PlayerAnimator**. Выбрав Итана на сцене, перетащите аниматора на свойство **Controller** компонента **Animator** на панели **Inspector** (рис. 18.12).

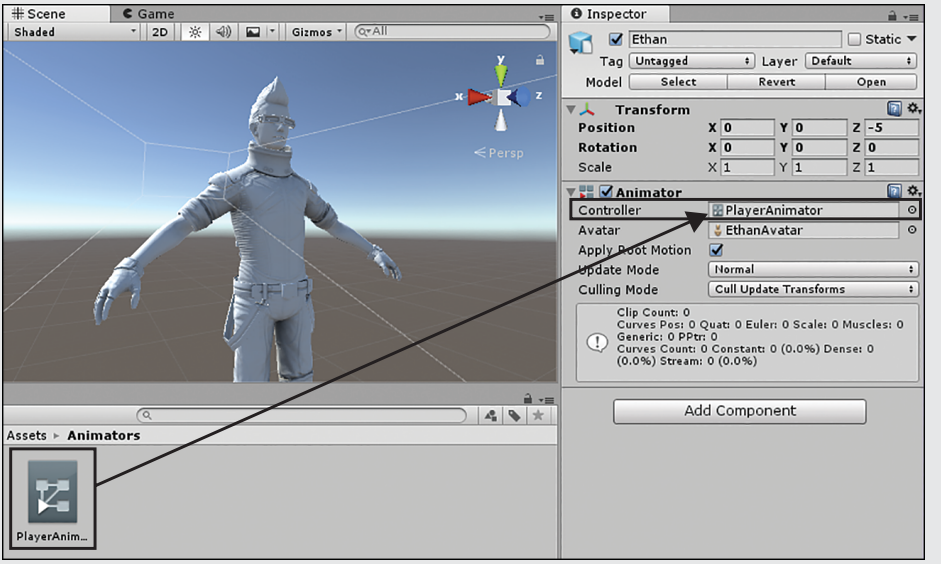


Рис. 18.12. Добавление аниматора к модели

5. Добавьте на сцену плоскость. Поместите ее в позиции с координатами (0, 0, –5), задайте масштаб (10, 1, 10).
6. Найдите файл *Checker.tga* в файлах примеров для часа 18 и импортируйте его в свой проект. Создайте новый материал под названием **Checker** и выберите файл **Checker.tga** в качестве альбедо для материала (рис. 18.13).

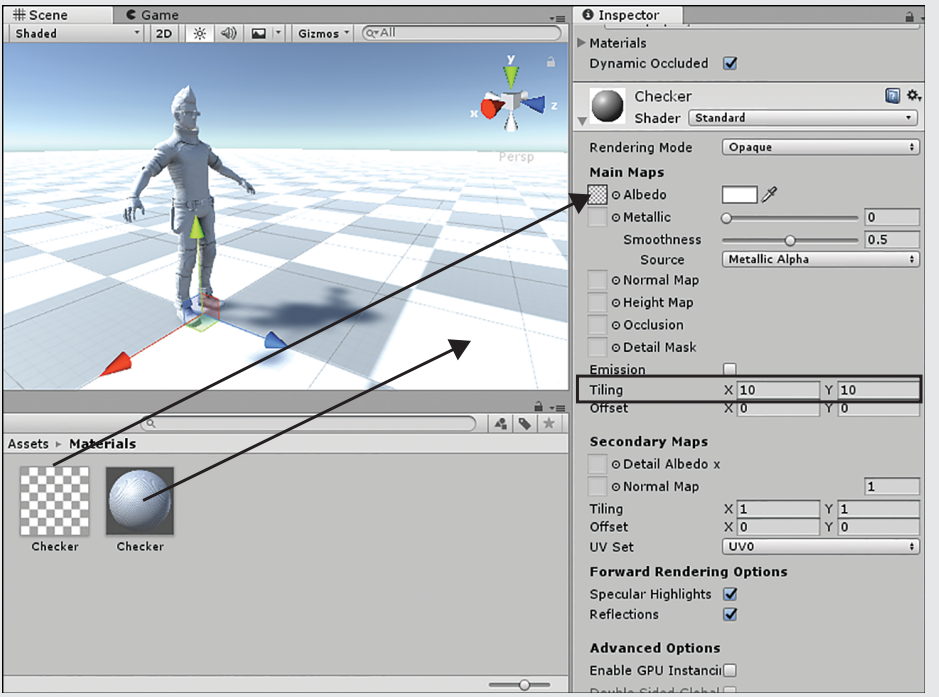


Рис. 18.13. Установка замощения для шахматной текстуры

7. Задайте свойствам **Tiling X** и **Tiling Y** значение **10** и примените материал к плоскости (пока она нам не нужна, но пригодится позже).

Панель Animator

Двойной щелчок по аниматору (или команда **Window** ⇒ **Animator**) открывает панель **Animator**. Функции на ней отображаются в виде графа, позволяя визуально создавать последовательности анимаций и комбинировать их. Вот где система Mecanim реально крута.

На рис. 18.14 показана панель **Animator**. Вы можете перемещаться по ней, удерживая среднюю кнопку мыши, а масштаб изменять с помощью колеса прокрутки. Созданный аниматор очень прост: есть только основной слой, нет параметров, а еще есть узлы **Entry**, **Exit** и **Any State**. (Эти компоненты мы обсудим более подробно позднее в этом часе.)

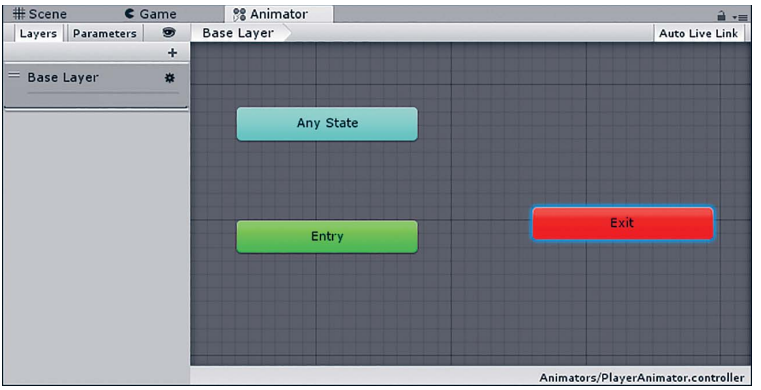


Рис. 18.14. Панель Animator

Анимация Idle

Первая анимация, которую мы применим к Итану, — **Idle**. Мы выполнили длительный процесс настройки ранее, и добавить эту анимацию сейчас будет просто. Найдите клип **Idle**, который хранится в файле **Idles.fbx** (см. рис. 18.7), и перетащите его на аниматор на панели **Animator** (рис. 18.15).

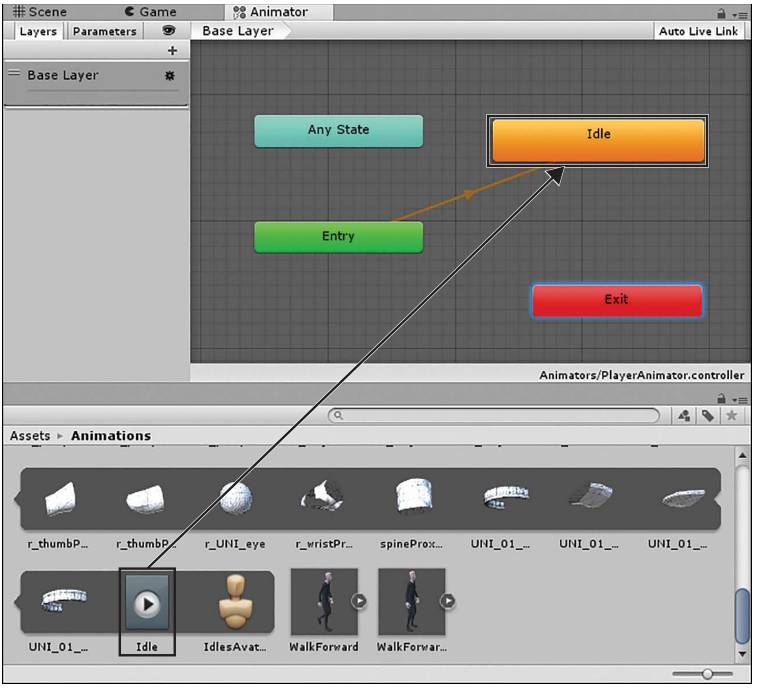


Рис. 18.15. Применение анимации **Idle**

Теперь вы можете запустить сцену и увидеть модель Итана, зацикленную в анимации **Idle**.

СОВЕТ

Как на катке

Когда вы запустите сцену, чтобы просмотреть анимацию **Idle** Итана, вы увидите, что ноги модели скользят по земле. Это связано с настройками данной конкретной анимации. Здесь персонаж определяет движение по своим бедрам, а не ногам (они слегка вращаются). Вы можете исправить проблему на панели **Animator**. Выберите состояние анимации **Idle**, а на панели **Inspector** установите флажок **Foot IK** (рис. 18.16). Теперь модель будет отслеживать свои ступни на земле. Так мы получим правильную анимацию персонажа, и ступни будут располагаться как надо. Кстати, аббревиатура IK расшифровывается как Inverse Kinematics — обратная кинематика. Подробно о ней в этой книге мы говорить не будем.

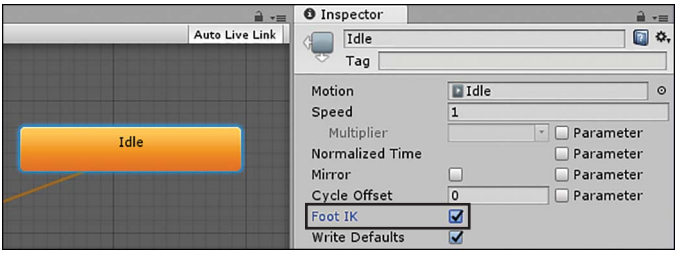


Рис. 18.16. Установка флажка **Foot IK**

Параметры

Параметры подобны переменным для аниматора. Вы можете задать их на панели **Animator**, а затем управлять ими с помощью скриптов. Эти параметры влияют на то, когда анимация перемещается и переходит одна в другую. Чтобы создать параметр, нажмите кнопку + на вкладке **Parameters** панели **Animator**.

ПРАКТИКУМ

Добавление параметров

В этом упражнении мы добавим два параметра. Нам понадобится проект и сцена, над которой вы работали ранее в этом часе. Выполните следующие действия.

1. Убедитесь, что вы выполнили все практикумы часа до этого момента.
2. На панели **Animator** перейдите на вкладку **Parameters**, а затем нажмите кнопку **+**, чтобы создать новый параметр. Выберите тип параметра **Float** и присвойте параметру имя **Speed** (рис. 18.17).

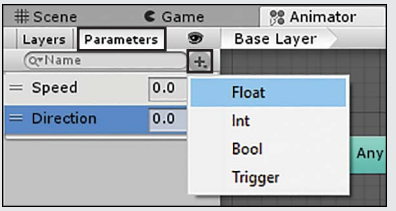


Рис. 18.17. Добавление параметров

3. Повторите шаг 2, чтобы создать параметр типа **Float** с именем **Direction**.

Состояния и деревья смешивания

Следующим шагом будет создание нового состояния, в котором находится модель. Оно определяет, как воспроизводится анимация. Вы уже создали состояние ранее, когда добавили анимацию **Idle** к контроллеру. Модель Итана будет иметь два состояния: **Idle** и **Walking** (бездействие и ходьба). С состоянием **Idle** уже разобрались. Поскольку состояние **Walking** подходит любой из трех анимаций, вам нужно создать состояние, в котором будет использоваться дерево смешивания, соединяющее одну или несколько анимаций в зависимости от параметров. Выполните следующие действия.

1. Щелкните правой кнопкой мыши по пустому пространству панели **Animator** и выберите команду **Create State** ⇒ **From New Blend Tree**. На панели **Inspector** присвойте созданному состоянию имя **Walking** (рис. 18.18).

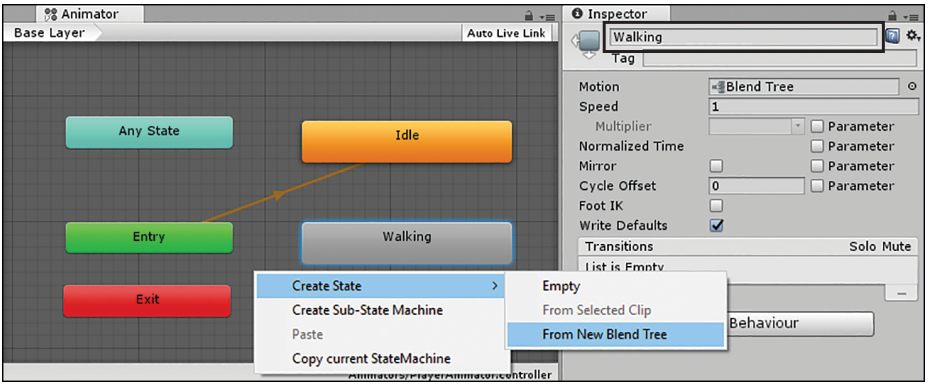


Рис. 18.18. Создание нового состояния

2. Дважды щелкните по новому состоянию, чтобы развернуть его, и выберите новое состояние **Blend Tree**. На панели **Inspector** откройте раскрывающийся список **Parameter** и выберите пункт **Direction**. Затем добавьте три движения, щелкнув мышью по значку + под списком движений и выбрав команду **Add**

Field Motion. Под графиком установите минимальное и максимальное значения, равные **−1** и **1**, соответственно (рис. 18.19).

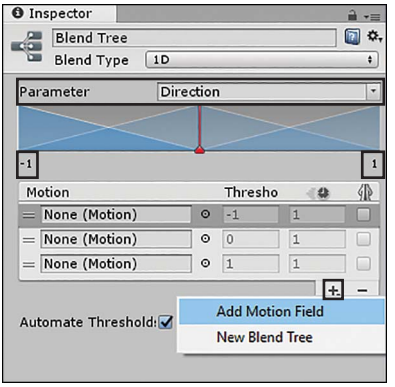


Рис. 18.19. Добавление полей движения

3. Перетащите каждую из трех анимаций ходьбы на одно из трех полей движения в следующем порядке: **WalkForwardTurnLeft**, **WalkForwardStraight**, **WalkForwardTurnRight** (рис. 18.20). Помните, что запускаемые клипы расположены в файле **WalkForwardTurns.fbx** и анимация движения вперед находится в файле **WalkForward.fbx**.

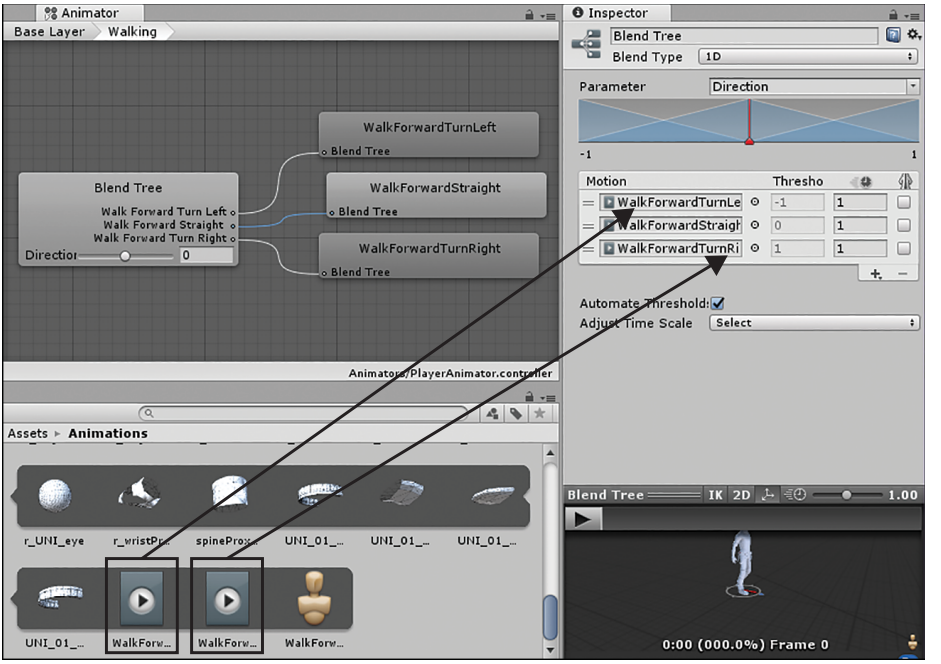


Рис. 18.20. Изменение минимальных значений и добавление анимации в дерево смешивания

Анимация ходьбы теперь готова к смешиванию по параметру **Direction**. По сути, вы настроили состояние смешивания на оценку параметра **Direction**. В зависимости от значения этого параметра дерево будет выбирать некоторый процент от каждой анимации, смешивать их и сгенерирует финальную уникальную анимацию. Например, если направление равно –1, будет воспроизводиться 100% анимации **WalkForwardTurnLeft**. Если направление равно .5, дерево будет воспроизводить 50% анимации **WalkForwardStraight** и 50% анимации **WalkForwardTurnRight**. Несложно понять, насколько это мощный инструмент! Чтобы выйти из подробного режима, нажмите кнопку **Base Layer** в верхней части панели **Animator** (рис. 18.21).

Рис. 18.21. Вид панели **Animator**

Переходы

Последнее, что вам нужно сделать, прежде чем завершить работу, — это сообщить аниматору, как переключаться между анимациями **Idle** и **Walking**. Вам нужно настроить два перехода — от **Idle** к **Walking** и наоборот. Чтобы создать переход, выполните следующие действия.

1. Щелкните правой кнопкой мыши по состоянию **Idle** и выберите команду **Make Transition**, чтобы создать белую линию, следующую за вашей мышью. Щелкните мышью по состоянию **Walking**, чтобы соединить его с состоянием **Idle**.

2. Повторите шаг 1, но теперь соедините состояние **Walking** с состоянием **Idle**.

3. Измените переход от состояния **Idle** к состоянию **Walking**, щелкнув мышью по белой стрелке между ними. Добавьте условие **Speed Greater 0,1**, как показано на рис. 18.22. Сделайте то же самое для перехода от состояния **Walking** к состоянию **Idle**, только укажите условие **Speed Less Than 0,1**.

4. Сбросьте флажок **Has Exit Time**, чтобы позволить анимации **Idle** прерываться при нажатии клавиши ходьбы.

Аниматор готов. Вы можете заметить, что при запуске сцены ни одна из анимаций движения не включается. Это связано с тем, что параметры скорости и направления не меняются. В следующем разделе вы узнаете, как изменить их с помощью скриптов.

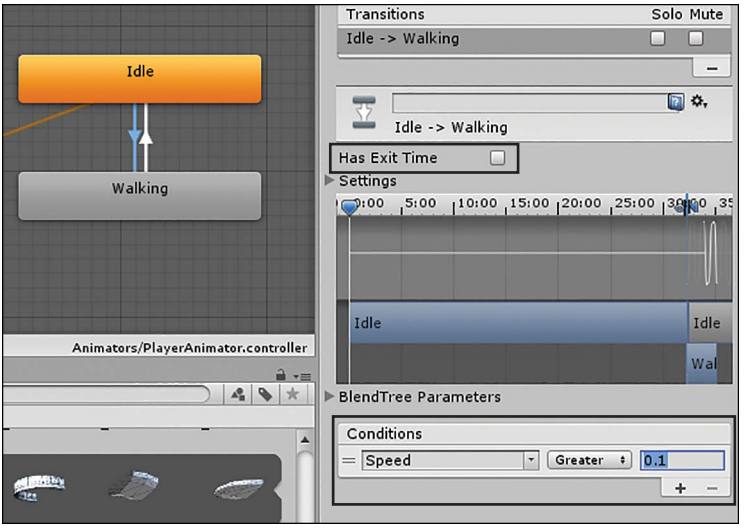


Рис. 18.22. Изменение переходов

Скрипты для аниматоров

Теперь, когда мы сделали модель, скелет, анимацию, аниматор, переходы и дерево смешивания, пора добавить ко всему этому интерактивности.

К счастью, код скриптов прост. Бóльшая часть тяжелой работы уже сделана в редакторе. На данный момент, все, что вам нужно, — настроить параметры, созданные в аниматоре, чтобы Итан пришел в движение. Поскольку заданные параметры настроены на числа с плавающей запятой (`float`), вам нужно вызвать следующий метод аниматора:

```
SetFloat (<имя>, <значение>);
```

ПРАКТИКУМ

Последний штрих

В следующих шагах мы добавим в наш проект скрипт, чтобы все заработало.

1. Создайте папку **Scripts** и добавьте в нее новый скрипт. Присвойте ему имя **AnimationControl**. Назначьте его модели Итана на сцене. (Это важно!)
2. Добавьте в скрипт **AnimationControl** следующий код:

```
Animator anim;
void Start ()
{
    // Получаем ссылку на аниматор
    anim = GetComponent<Animator> ();
}
```

```
void Update ()
{
    anim.SetFloat ("Speed", Input.GetAxis ("Vertical"));
    anim.SetFloat ("Direction", Input.GetAxis("Horizontal"));
}
```

3. Запустите сцену и обратите внимание, что анимации управляются горизонтальной и вертикальной осями ввода. (Если вы забыли, горизонтальная и вертикальная оси управляются клавишами **W**, **A**, **S**, **D** и клавишами со стрелками.)

Готово! Если вы запустите сцену после добавления этого скрипта, то увидите кое-что странное. Итан не только воспроизводит анимации ходьбы, поворота и бездействия, но и сама модель тоже двигается. Причины у этого две.

Первая заключается в том, что у анимаций есть встроенные движения. Они были добавлены аниматорами до вставки анимаций в Unity. Иначе вы должны были бы запрограммировать движение сами.

Вторая причина в том, что по умолчанию аниматор позволяет анимации перемещать модель. Это можно изменить, выбрав опцию **Apply Root Motion** компонента **Animator** (рис. 18.23), но в данном случае это приведет к странному поведению!

Финальный файл проекта есть в файлах примеров — вы можете открыть его и сравнить!

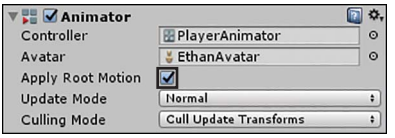


Рис. 18.23. Свойство **Apply Root Motion**

Резюме

В этом часе мы с нуля собрали простую анимацию. Мы начали с куба и добавили компонент **Animator**. Затем мы создали контроллер **Animator** и связали его с аниматором. Затем мы настроили состояния анимации и соответствующие движения. Наконец, вы узнали, как можно смешивать состояния.

Вопросы и ответы

Вопрос: **Можно ли создать ключевой кадр анимации гуманоидов в Unity?**

Ответ: Unity не позволяет делать это с гуманоидами, но вы можете найти в Интернете обходные пути. Например, вы можете создать анимацию в отдельном приложении 3D-моделирования и импортировать ее в Unity.

Вопрос: **Может ли объект перемещаться и аниматором, и физическим движком?**

Ответ: Вы можете это сделать (с некоторой осторожностью), но лучше избегать такого смешивания. Для любого движения вы должны точно знать, что контролирует игровой объект — физический движок или аниматор.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. На что должен ссылаться компонент **Animator**, чтобы работать?
2. Какой цвет у состояния по умолчанию анимации на вкладке **Animator**?
3. Сколько клипов (движений) может иметь состояние анимации?
4. Что нужно использовать для запуска переходов анимации из скрипта?

Ответы

1. Контроллер **Animator** должен быть создан и подключен к компоненту **Animator**. В случае гуманоидных персонажей также потребуется аватар.
2. Оранжевый.
3. Может быть по-разному. Состояние анимации может включать один клип, дерево смешивания или другой конечный автомат.
4. Параметры аниматора.

Упражнение

Требуется множество ресурсов, чтобы выстроить надежную и высококачественную систему анимации. В этом часе для достижения цели потребовались одни параметры, в другой ситуации потребуются другие. Здесь много всего разного, и определяющее значение имеют опыт и обучение.

Ваше упражнение сейчас — продолжить изучение системы Mecanim. Обязательно начните с чтения документации по этой системе Unity. Вы можете найти ее на сайте Unity по адресу **docs.unity3d.com/ru/current/Manual/AnimationSection.html**.

Вы можете также исследовать анимированный префаб Итана **ThirdPersonController.prefab**, расположенный в папке **Assets\Standard Assets\Characters\ThirdPersonCharacter\Prefabs**. Этот префаб гораздо сложнее, чем созданный нами Итан. У него три дерева смешивания — **Airborne**, **Grounded** и **Crouching**.

19-Й ЧАС

Система Timeline

Что вы узнаете в этом часе

- Введение в структуру системы Timeline
- Как добавлять клипы и выстраивать из них последовательности
- Более сложные случаи использования системы Timeline

В этом часе вы изучите очень мощный инструмент создания последовательностей в Unity: **Timeline**. Мы начнем со структуры и концепции таймлайна и компонента **Playable Director**. Затем мы рассмотрим, что такое клипы и как располагать их на таймлайне. Изучив различные способы использовать систему **Timeline**, чтобы внести в проект разнообразие, вы закончите этот час.

ПРИМЕЧАНИЕ

Его загадочность превосходит только его сила

В этом часе вы изучите различные компоненты системы **Timeline** в Unity. Следует отметить, что **Timeline** — очень мощный и вместе с тем сложный инструмент. На самом деле в этом часе мы рассмотрим лишь верхушку айсберга его возможностей. Вся система создана из простых, но модульных частей, позволяющих использовать ее множеством различных способов. То есть, если у вас возникнет вопрос вроде «Хм, интересно, могу ли я сделать с помощью таймлайна вот это», то ответ, вероятно, «да». Это не всегда легко, но, скорее всего, возможно!

Знакомство с Timeline

По сути, **Timeline** — инструмент создания последовательностей. Это означает, что с его помощью можно сделать так, чтобы определенные действия происходили в определенные моменты времени в определенном порядке. А что это за действия — зависит только от вашей фантазии. В некотором смысле таймлайн очень похож на контроллер **Animator** (см. час 18). Контроллер **Animator** используется для создания последовательности и управления анимациями объекта.

Ограничение заключается в том, что контроллер **Animator** может управлять только собой или дочерними объектами. Но он не может быть использован, например, для создания сцены, где два охранника травят байки, а за углом крадется вор. Для таких задач нужен **Timeline**. Вы можете применять его для создания последовательностей разных действий разных объектов в разное время.

Структура Timeline

Основной элемент последовательности называется *клипом*. Хотя это будет означать, что таймлайн используется для анимации, правда заключается в том, что клипом может быть что угодно — аудиоклип, событие данных, включение и отключение игровых объектов. Вы даже можете запрограммировать собственные клипы и заставить их делать что угодно.

Клипы размещаются на одном или нескольких *треках* (рис. 19.1). Они определяют, какие типы клипов могут быть размещены на них и какими объектами они управляют. Чтобы анимировать два персонажа, нужно два трека (по одному для каждого), с одной или несколькими анимациями на каждом.

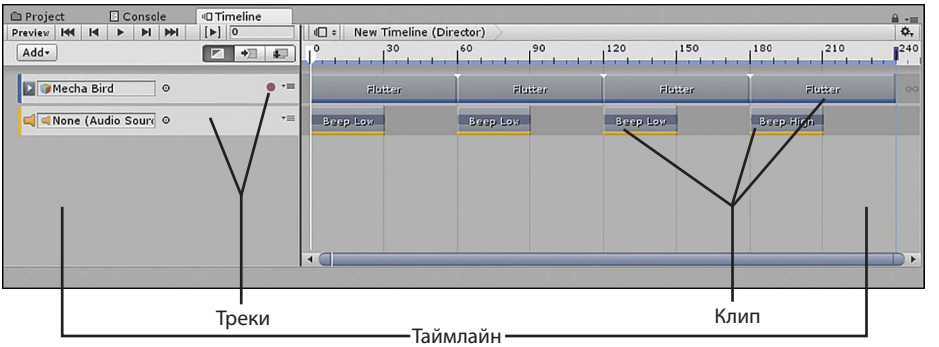


Рис. 19.1. Структура панели **Timeline**

Оба клипа и треки располагаются на ассете таймлайна (рис. 19.1). Этот ассет может использоваться на разных сценах или много раз на одной сцене. Таймлайн отслеживает объекты, которыми он управляет, через *привязки*. Их можно установить с помощью кода, но, как правило, достаточно лишь перетащить объект на таймлайн.

Наконец, в системе **Timeline** используется компонент **Playable Director** для управления таймлайном. Этот компонент добавляется к игровому объекту на сцене и определяет, когда таймлайн запускается, когда останавливается и что происходит, когда он заканчивается.

Создание таймлайнов

Как упоминалось ранее, таймлайн — это ассет, на который укладывается все остальное. Таким образом, первым шагом в работе с **Timeline** будет создание ассета таймлайна. Чтобы сделать это, щелкните правой кнопкой мыши по панели **Project** и выберите команду **Create** ⇒ **Timeline** (рис. 19.2).

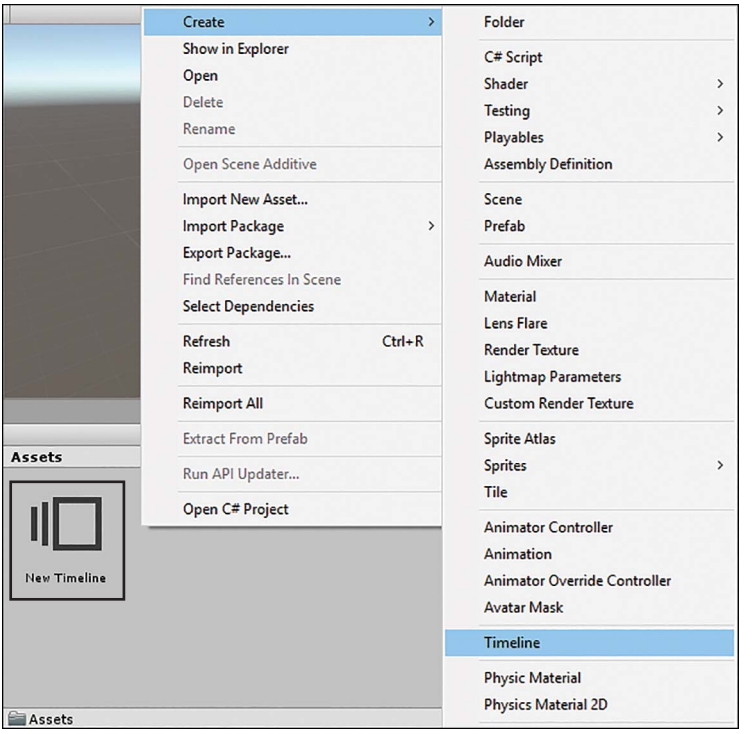


Рис. 19.2. Создание таймлайна

Когда таймлайн будет создан, он должен управляться компонентом **Playable Director**, который добавляется к игровому объекту в сцене. Чтобы добавить компонент **Playable Director** к объекту, вы можете выделить его, а затем выбрать команду **Add Component** ⇒ **Playables** ⇒ **Playable Director**. Затем вам нужно установить ассет таймлайна в качестве свойства **Playable** компонента **Playable Director**. Более простой способ — перетащить таймлайн с панели **Project** на игровой объект на панели **Hierarchy**, который должен управлять таймлайном (рис. 19.3).

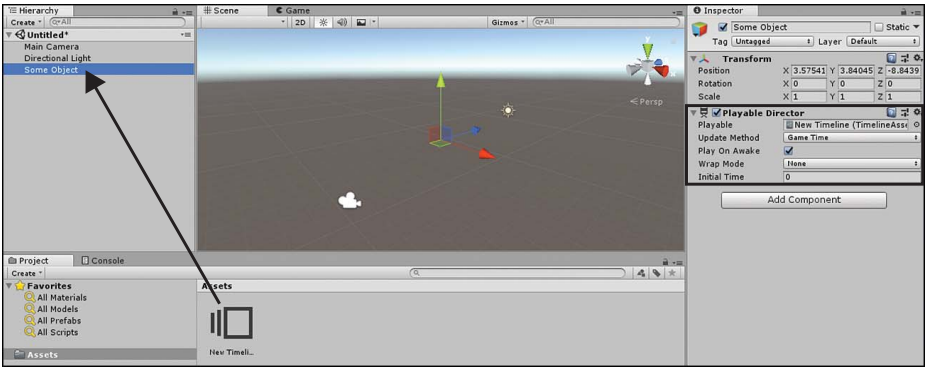


Рис. 19.3. Добавление компонента **Playable Director** к игровому объекту

ПРАКТИКУМ

Создание ассета таймлайна

Попрактикуемся: создайте ассет таймлайна и добавьте компонент **Playable Director** к игровому объекту. Обязательно сохраните проект и сцену, созданную здесь, так как они понадобятся нам позже в этом часе. Выполните следующие действия.

1. Создайте новый проект или сцену. Добавьте в проект новую папку с именем **Timelines**.
2. В папке **Timelines** создайте новый таймлайн, щелкнув правой кнопкой мыши и выбрав команду **Create** ⇒ **Timeline**.
3. Добавьте на сцену новый игровой объект (выбрав команду **GameObject** ⇒ **Create** ⇒ **Create Empty**). Присвойте ему имя **Director**.
4. Перетащите новый ассет таймлайна с панели **Project** на объект **Director** на панели **Hierarchy**. (Это не сработает, если вы попытаетесь перетащить его на сцену или на панель **Inspector**.)
5. Выберите игровой объект **Director** и убедитесь, что на панели **Inspector** появился компонент **Playable Director**.

Работа с таймлайнами

Создать таймлайн легко, но этого еще не достаточно для результата. Чтобы использовать таймлайн, вам нужны треки и клипы, которые управляют последовательностями объектов на панели **Scene**. Вся эта работа осуществляется на панели **Timeline**, которая очень похожа на панель **Animation**, рассмотренную в часе 17.

Панель Timeline

Для работы с таймлайном вам необходимо открыть панель **Timeline**. Вы можете сделать это, выбрав команду **Window** ⇒ **Timeline** или дважды щелкнув по ассету таймлайна на панели **Project**.

Вы увидите элементы управления для предварительного просмотра и воспроизведения, управления режимом, а также область для работы с треками и клипами (рис. 19.4).

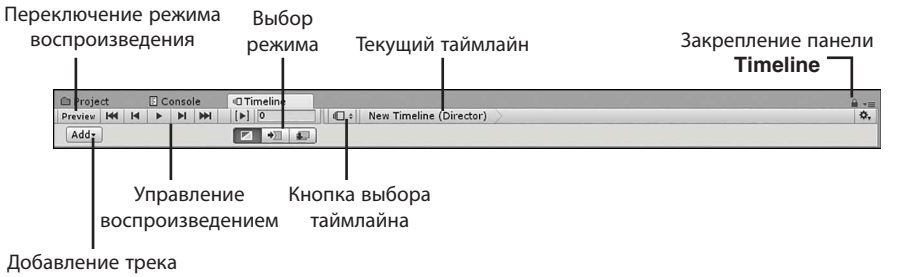


Рис. 19.4. Панель **Timeline**

Нет особого смысла изучать пустую панель **Timeline**. Вместо этого давайте двигаться дальше, и будем изучать панель по мере работы с ней.

СОВЕТ

Закрепление панели Timeline

Работая на панели **Timeline**, вы часто будете взаимодействовать с другими объектами на панели **Scene** или **Project**. Но в этом случае сбрасывается выделение с объекта **Director**, и панель **Timeline** становится пустой, что осложняет работу. Лучше всего закрепить панель **Timeline** (рис. 19.4). Тогда выбор других объектов не будет закрывать ее. Вы можете выбрать, какой таймлайн изменить, с помощью переключателя на панели **Timeline**. Намного удобнее!

ВНИМАНИЕ!

Режим предпросмотра

На панели **Timeline** есть режим предварительного просмотра (**Timeline**) (рис. 19.4). Когда этот режим включен, вы видите, как таймлайн будет воздействовать на объекты на сцене. Это крайне важно для выстраивания последовательностей и правильного воспроизведения. К счастью, при работе на панели **Timeline** режим предварительного просмотра включается автоматически. Однако при этом отключаются некорые функции, например вы не сможете изменять префабы. Если вы заметили, что что-то выглядит неправильно, попробуйте выйти из режима предварительного просмотра, чтобы убедиться, что панель **Timeline** не вводит вас в заблуждение.

Треки

Треки на таймлайне определяют действия, какой объект их совершает и что объект вообще может делать. Трек таймлайна выполняет немало функций. Чтобы

добавить его, нажмите кнопку **Add** на панели **Timeline** (см. рис. 19.4). В таблице 19.1 перечислены типы встроенных треков и описан их функционал.

ТАБЛ. 19.1. Типы треков на таймлайне

Тип	Описание
Group	Этот тип трека не имеет собственных функций. Он позволяет группировать треки для облегчения организации
Activation	Этот тип трека позволяет активировать/деактивировать игровые объекты
Animation	Этот тип трека позволяет воспроизводить анимацию объектов. Обратите внимание, что он имеет приоритет над любой анимацией, воспроизводимой аниматором
Audio	Этот тип трека позволяет запускать и останавливать воспроизведение аудиоклипов
Control	Этот тип трека позволяет управлять другими воспроизводимыми объектами. *Воспроизводимым* является любой ассет, созданный системой воспроизведения Unity. Наиболее распространенный способ использования этого типа трека — заставить таймлайн воспроизводить другой таймлайн. Таким образом, вы можете использовать таймлайн для выстраивания последовательностей и управления несколькими дочерними таймлайнами
Playable	Это специальный тип трека для управления пользовательскими воспроизводимыми ассетами (созданными для расширения системы **Timeline**)

ПРИМЕЧАНИЕ

Новые и собственные треки

В таблице 19.1 перечислены треки, которые были доступны в Unity на момент публикации этой книги. Система **Timeline** — относительно новая функция, и работа над ее возможностями продолжается. Список будет дополняться, и в следующих версиях появятся новые типы треков.

Кроме того, система **Timeline** расширяема. Если вам не хватает каких-то функций, вы можете создавать собственные треки. Уже сейчас многие разработчики занимаются этим. Готовые пользовательские треки можно найти в Asset Store. Если вам интересно, можете начать с библиотеки Default Playables Library (см. **assetstore.unity.com/packages/essentials/default-playables-95266**).

ПРАКТИКУМ

Добавление треков

Теперь мы добавим треки на созданный ранее таймлайн. Не забудьте сохранить эту сцену, так как она понадобится нам позже. Выполните следующие действия.

1. Откройте сцену из практикума «Создание ассета таймлайна». Добавьте на нее куб и поместите его в позицию с координатами (0, 0, 0).
2. Откройте панель **Timeline** (выбрав команду **Window** ⇒ **Timeline**). Выделите объект **Director** на сцене. Вы увидите ваш таймлайн на панели **Timeline** (но треков на нем пока нет).
3. Закрепите панель **Timeline**, щелкнув мышью по значку замка в правом верхнем углу (см. рис. 19.4).
4. Добавьте трек анимации, нажав кнопку **Add** и выбрав пункт **Animation Track**.
5. Привяжите куб к треку, перетащив куб с панели **Hierarchy** на свойство **Track Binding** на панели **Timeline** (рис. 19.5).

Рис. 19.5. Привязка трека

6. При появлении запроса нажмите кнопку **Create Animator on Cube**.
7. Добавьте аудиотрек, нажав кнопку **Add** и выбрав пункт **Audio Track**. Аудиотрек не обязательно должен быть привязан к игровому объекту, чтобы функционировать.

Клипы таймлайнов

После того, как треки размещены на таймлайне, нужно добавить клипы. Работа с ними немного варьируется в зависимости от типа трека, но общая функциональность всех клипов одинаковая. Чтобы добавить клип, щелкните правой кнопкой мыши по треку и выберите команду **Add From <*тип клипа*>**.

Например, если вы добавляете клип на аудиотрек, в контекстном меню будет присутствовать пункт **Add from Audio Clip** (рис. 19.6). Появится меню, позволяющее вам выбрать клип, который можно добавить на трек. Кроме того, можно перетащить соответствующий ассет с панели **Project** на трек на панели **Timeline**.

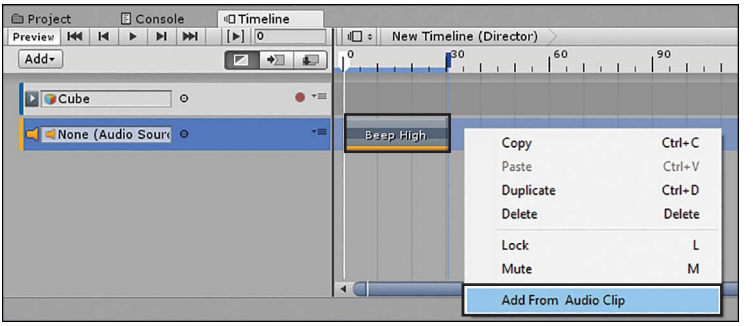


Рис. 19.6. Добавление на аудиотрек клипа **Beep High**

После этого вы сможете перемещать клип, чтобы настроить время воспроизведения. Вы также можете настроить размер клипа, тем самым изменяя его длительность. Вы можете даже обрезать или зациклить клип на определенное время. Следует отметить, что точная настройка длительности клипа зависит от типа трека.

Кроме перетаскивания клипов по треку для настройки воспроизведения, вы также можете выбрать клип, а затем изменить его параметры на панели **Inspector**. Так у вас будет дополнительный контроль и возможности настройки при работе с клипами (хотя и не такой быстрый, как простое перетаскивание).

ПРАКТИКУМ

Выстраивание последовательностей клипов

Наконец настало время добавить на таймлайн что-нибудь интересное! Мы будем добавлять клипы на треки таймлайна, созданные в практикуме «Добавление треков». Обязательно сохраните эту сцену еще раз, потому что мы продолжим работу с ней позже. Выполните следующие действия.

1. Откройте сцену, созданную в практикуме «Добавление треков». Найдите файлы примеров для часа 19 и импортируйте две папки: **Animations** и **Audio**.
2. На панели **Timeline** щелкните правой кнопкой мыши по треку анимации, связанному с объектом **Cube**, и выберите команду **Add from Animation Clip**.
3. В появившемся диалоговом окне **Select Animation Clip** выберите импортированный анимационный клип **Red**.
4. Повторите шаг 3, добавив клипы **Orange**, **Yellow**, **Green**, **Blue**, **Indigo** и **Violet** (рис. 19.7).
5. Щелкните правой кнопкой мыши по аудиотреку и выберите команду **Add from Audio Clip**. Импортируйте аудиоклип **Beep High**.
6. Продублируйте клип **High Beep** на аудиотреке шесть раз, выделив клип и нажав сочетание клавиш **Ctrl+D** в Windows или **⌘+D** в macOS.
7. Переместите аудиоклипы так, чтобы начало их звучания совпадало с изменением цвета куба.

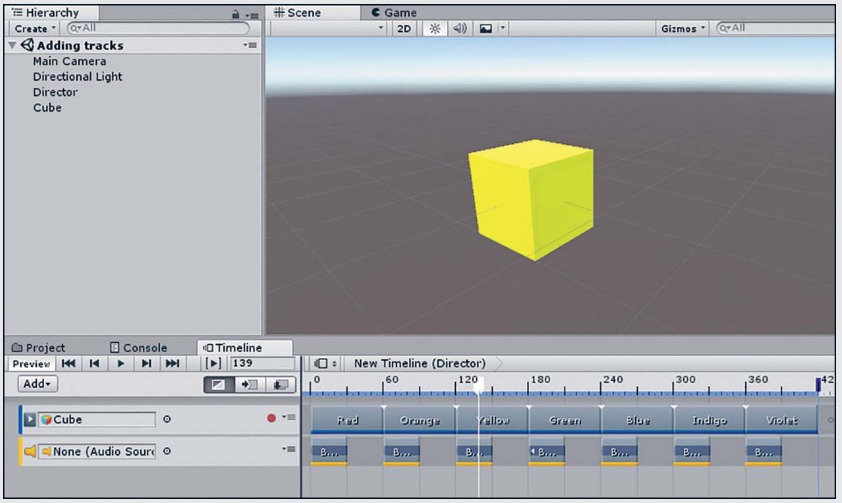


Рис. 19.7. Окончательные настройки таймлайна

8. Запустите предварительный просмотр таймлайна путем перемещения ползунка или нажатия кнопки **Play** на панели **Timeline**. Звук пока не воспроизводится. (Это будет добавлено в более поздней версии Unity). Сохраните и запустите сцену. Обратите внимание, что изменение цвета куба совпадает со звуком.

СОВЕТ

Быстрая анимация

Треки анимации полностью поддерживают и импортированные анимации, и созданные в Unity с помощью панели **Animation**. Но если вы хотите быстро анимировать

объект для использования с таймлайном, есть еще более простой способ. Выделив игровой объект, связанный с треком анимации, вы можете нажать кнопку **Record** непосредственно на самом треке (рис. 19.8). Эта функция идентична режиму **Record** на панели **Animation** (см. час 17). Единственное отличие состоит в том, что данные анимации хранятся непосредственно в ассете таймлайна. Используя этот метод, вы сможете быстро создавать простые анимации, которые добавят вашей кинематике глубины.

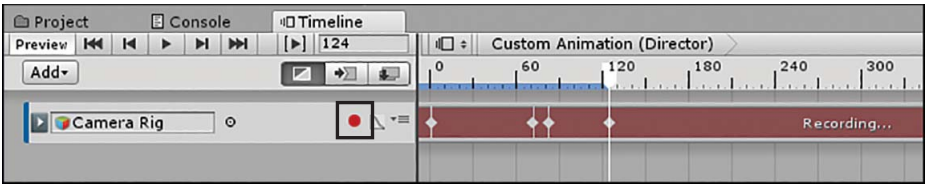


Рис. 19.8. Запись на панели **Timeline**

СОВЕТ

Отключение и блокировка

Каждый трек на панели **Timeline** можно заблокировать для предотвращения случайных изменений. Выделите его и нажмите клавишу **L**, либо щелкните правой кнопкой мыши по треку и выберите команду **Lock**. Кроме того, трек можно отключить, и тогда он не будет воспроизводиться вместе с таймлайном. Выделите его и нажмите клавишу **M** либо щелкните правой кнопкой мыши по треку и выберите команду **Mute**.

Расширяем границы возможностей

Мы пока лишь краешком коснулись того, что вы можете делать с таймлайном. Очевидно, эта система очень хороша для кинематики, но кроме того она отлично подходит для создания множества различных поведений. Например, можно задавать движения стражников, чтобы толпа людей выглядела более реалистично, или развернуть сложный набор эффектов экрана, когда персонаж получает урон.

Смешивание клипов на треке

Ранее в этом часе мы поработали с клипами в виде отдельных элементов на треке. Но не обязательно должно быть именно так. Вы можете использовать график, чтобы смешать два разных клипа и получить новый, комбинированный результат. Для этого необходимо лишь перетащить один клип на другой. На рис. 19.9 показан результат смешивания красного и оранжевого клипа из практикума «Выстраивание последовательностей клипов».

Смешивание допустимо не только для анимационных клипов. Вы можете также смешивать содержимое на аудиотреках и многих пользовательских треках,

которые создает сообщество Unity. Возможность смешивать клипы предоставляет невероятный уровень контроля и плавности «промежуточных кадров» между ключевыми кадрами.

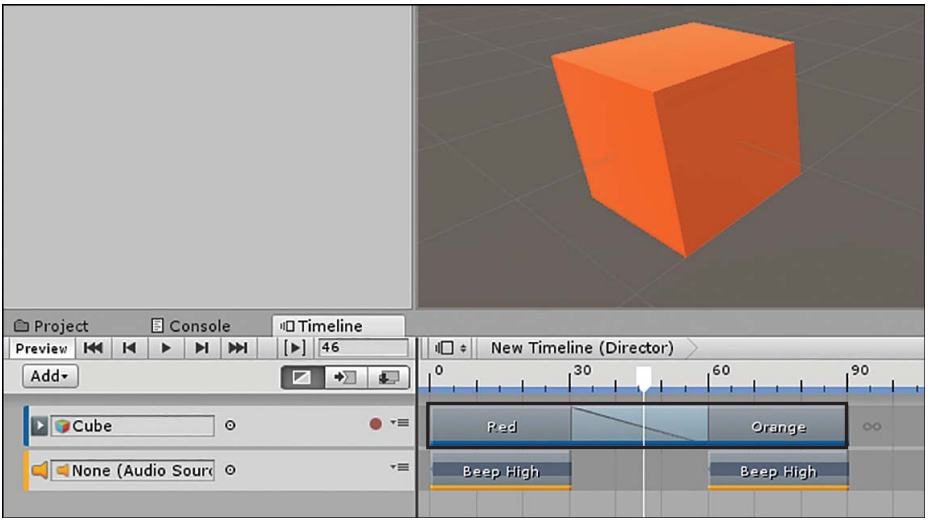


Рис. 19.9. Смешивание клипов на панели **Timeline**

ПРАКТИКУМ

Смешивание клипов

Далее вы увидите, как можно смешать клипы, добавленные в практикуме «Выстраивание последовательностей клипов», чтобы создать плавный переход между цветами радуги. Не забудьте сохранить эту сцену еще раз, так как она понадобится нам позже. Выполните следующие действия.

1. Откройте сцену, созданную в практикуме «Выстраивание последовательностей клипов». Убедитесь, что открыта панель **Timeline** и выбран игровой объект **Director**.
2. На панели **Timeline** перетащите клип **Orange** так, чтобы он наполовину перекрывал клип **Red**. На панели **Inspector** клип **Orange** должен начинаться с кадра 30 и заканчиваться на кадре 90.
3. Продолжайте смешивать цветовые клипы, пока не получите непрерывный микс (см. рис. 19.10).

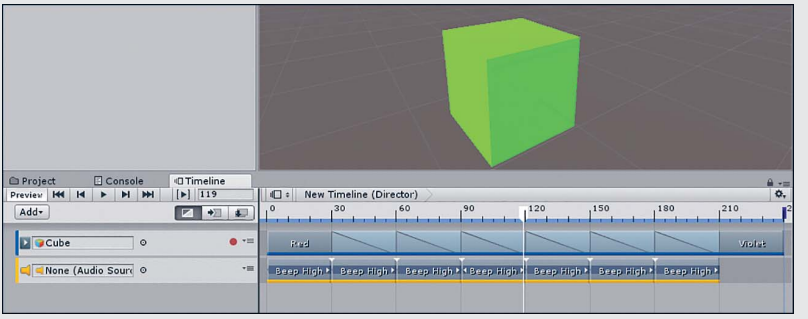


Рис. 19.10. Смешивание всех цветовых клипов

4. Сдвиньте аудиоклипы таким образом, чтобы каждый из них совпадал с началом анимации определенного цвета. Эти звуковые сигналы помогут идентифицировать момент наложения каждого цвета.

Использование скриптов с Timeline

В целом код, необходимый для создания пользовательских треков и клипов, довольно сложен и не обсуждается в этой книге. Но что-то простое вы сделать можете, например настроить время запуска таймлайна вместо его автоматического старта при запуске сцены. Так вы можете запускать кинематику или игровые события.

Чтобы написать код, который будет работать с системой **Timeline**, вы должны импортировать в Unity библиотеку `Playables`:

```
using UnityEngine.Playables;
```

Затем вы можете создать переменную типа `PlayableDirector` и использовать ее, чтобы управлять таймлайном. Два основных метода называются `Play()` и `Stop()`:

```
PlayableDirector director = GetComponent<PlayableDirector>();

director.Play();        // Старт таймлайна
director.Stop();        // Стоп таймлайна
```

Еще вы можете выбрать, какой именно объект воспроизводить. Можно, например, случайно выбирать один из нескольких таймлайнов или определять, какой именно таймлайн воспроизводить в зависимости от условий:

```
public PlayableAsset newTimeline;

void SomeMethod()
```

```
{
    director.Play(newTimeline);
}
```

Используя эти простые методы, вы раскроете бóльшую часть возможностей системы **Timeline** в процессе выполнения.

ПРАКТИКУМ

Запуск таймлайнов с помощью кода

В этом упражнении мы закончим сцену этого часа, настроив управление таймлайном с помощью скрипта.

1. Откройте сцену, созданную в практикуме «Смешивание клипов». Создайте папку с именем **Scripts** и добавьте в нее скрипт под названием **InputControl**.
2. Выделите объект **Director** и сбросьте флажок **Play on Awake** компонента **Playable Director**. Прикрепите скрипт **InputControl** к игровому объекту **Director**, а затем добавьте следующий код:

```
using UnityEngine;
using UnityEngine.Playables;

public class InputControl: MonoBehaviour
{
    PlayableDirector director;

    void Start()
    {
        director = GetComponent<PlayableDirector>();
    }

    void Update()
    {
        if (Input.GetButtonDown("Jump"))
        {
            if (director.state == PlayState.Paused)
                director.Play();
            else
                director.Stop();
        }
    }
}
```

3. Запустите сцену. Теперь при нажатии клавиши **Пробел** таймлайн будет включаться и выключаться.

Резюме

В начале часа мы исследовали систему **Timeline** в Unity. Мы узнали, как создать ассет таймлайна и заполнить его треками и клипами. Затем вы научились смешивать клипы, а в конце часа познакомились с тем, как управлять таймлайнами с помощью скриптов.

Вопросы и ответы

Вопрос: **В чем разница между таймлайном и воспроизводимым объектом?**

Ответ: Таймлайн и есть воспроизводимый объект. Более того, он воспроизводит другие такие объекты (которые тоже содержат воспроизводимые объекты). Понятно?

Вопрос: **Сколько таймлайнов может быть воспроизведено на сцене?**

Ответ: Вы можете создать и воспроизводить на сцене столько таймлайнов, сколько хотите. Но имейте в виду, что, если два таймлайна управляют одним объектом, один из них будет иметь приоритет над другим.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Какой компонент воспроизводит таймлайн на сцене?
2. Сколько в Unity существует встроенных типов треков?
3. Как называется процесс назначения треку объекта, которым он будет управлять?
4. Как смешать два клипа?

Ответы

1. Компонент **Playable Director.**
2. Пять (шесть, если считать групповой).
3. Это называется *привязкой*.
4. Перетащите один клип на другой на треке.

Упражнение

Это упражнение не имеет конкретного результата — предлагаем вам побаловаться с системой **Timeline**. Кроме самой практики работы с **Timeline**, вы можете также посмотреть видео по данной теме (**unity3d.com/learn/tutorials/s/animation**). Вот что можно попробовать.

1. Создать динамический пользовательский интерфейс с помощью анимации.
2. Создать шаблон патрулирующего стражника.
3. Настроить серию сложных анимаций камеры.
4. Используйте трек активации, чтобы создать эффект шатра или стробоскопа с помощью источников света.

20-Й ЧАС

Игра четвертая: «Бег с препятствиями»

Что вы узнаете в этом часе

- Как создать игру «Бег с препятствиями»
- Как создать мир для игры «Бег с препятствиями»
- Как создать объекты для игры «Бег с препятствиями»
- Как создать элементы управления для игры «Бег с препятствиями»
- Как можно дополнительно улучшить игру «Бег с препятствиями»

Давайте создадим еще одну игру! В этом часе мы сделаем игру «Бег с препятствиями» — и в ней мы будем… бегать по полосе препятствий! Начнем с проекта игры. Затем мы перейдем к созданию мира. После этого создадим объекты и элементы управления. В конце часа мы попробуем поиграть и посмотрим, что в игре можно улучшить.

СОВЕТ

Завершенный проект

Обязательно проработайте этот час, чтобы создать полный игровой проект. Если возникнут трудности, вы сможете найти полную копию игры в приложенных к книге файлах примеров для часа 20. Просмотрите ее, если вам понадобится помощь или вдохновение.

Проектирование

Мы изучили основы проектирования игр в часе 6. Теперь воспользуемся полученными знаниями.

Концепция

В этой игре у нас будет киборг, который должен пробежать по полосе препятствий в туннеле как можно дальше. У вас будут бонусы, позволяющие продлить время игры. Вы должны избегать препятствий, так как они замедляют вас. Игра заканчивается, когда истекает отведенное время.

Правила

Правила игры определяют, как в нее играть, а также намекают на некоторые свойства объектов. Правила игры «Бег с препятствиями» заключаются в следующем.

- Игрок может двигаться влево или вправо, а также «исчезать» (становиться прозрачным). Иными способами игрок двигаться не может.
- Сталкиваясь с препятствием, игрок замедляется на 50% в течение 1 секунды.
- Если игрок ловит бонус, время игры увеличивается на 1,5 секунды.
- Игрок ограничен стенками туннеля.
- Условие поражения — заканчивается время.
- Условия победы нет, игрок старается пробежать так далеко, как сможет.

Требования

Требования, предъявляемые к этой игре, очень просты.

- Текстура полосы препятствий.
- Текстура стены.
- Модель игрока.
- Пользовательский шейдер для исчезновения игрока (его рассмотрим позже в этом часе).
- Бонус и препятствие. Их мы создадим в Unity.
- Менеджер игры. Его мы создадим в Unity.
- Эффект частиц бонуса. Он будет создан в Unity.
- Интерактивные скрипты. Их мы создадим с помощью Visual Studio.

Чтобы сделать эту игру симпатичнее, мы возьмем ассеты, созданные сообществом. В данном случае мы используем текстуры и модель игрока из игры Adventure, выпущенной компанией Unity Technologies (**www.assetstore.unity3d.com/en/#!/**

content/76216). Мы также возьмем пользовательский шейдер, который нам любезно предоставил Энди Дабок (**github.com/andydbc/HologramShader**).

Создание игрового мира

В качестве мира для этой игры будут выступать три куба, из которых состоит тоннель. Тут все довольно просто — сложности и веселье начнутся позже.

Сцена

Перед созданием поверхности и ее функциональности нужно создать и настроить сцену. Итак, сначала сделайте следующее.

1. Создайте новый 3D-проект под названием **Gauntlet Runner**. Создайте новую папку с именем **Scene** и сохраните в ней сцену под именем **Main**.
2. Установите объект **Main Camera** в позицию с координатами (0, 3, −10,7) и вращением (33, 0, 0).
3. В файлах примеров для часа 20 найдите две папки, **Materials** и **Textures**. Перетащите эти папки на панель **Project**, чтобы импортировать их.
4. В папке **Materials** найдите материал **Dark Sky**. Он представляет собой скайбокс. Обратите внимание на его шейдер **Skybox/Procedural** (рис. 20.1). Перетащите материал **Dark Sky** с панели **Project** на сцену, чтобы небо заволокло тьмой.

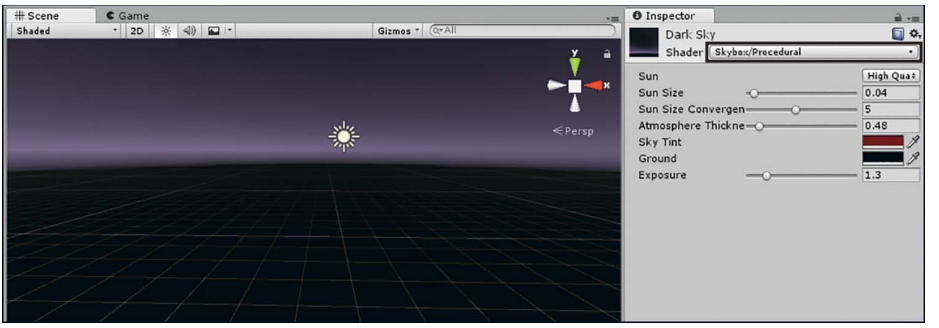


Рис. 20.1. Новый материал **Dark Sky**

Камера для этой игры будет находиться в фиксированном положении, чуть сверху над происходящим. Остальная часть мира пройдет под ней.

ПРИМЕЧАНИЕ

Метабезумие

В этом разделе вы импортировали три папки из приложенных ассетов. Внутри этих папок для каждого ассета есть файл с таким же именем и расширением.*meta*. Эти *метафайлы* хранят информацию, которая позволяет связывать ассеты друг с другом. Если эти файлы не были бы включены в приложенные ассеты, то материалы сохранили бы свои настройки, но назначение текстур для материалов не сохранилось бы. Аналогично, модель игрока не знала бы, какой у нее материал.

Полоса препятствий

Земля в этой игре будет представлять собой прокручивающуюся полосу препятствий. Однако в отличие от фона в игре «Капитан Бластер» (в часе 15), на самом деле ничего прокручиваться не будет. Мы объясним это более подробно в следующем разделе, а сейчас запомните, что вам нужно создать только один объект, который будет отвечать за прокрутку. Сама трасса будет состоять из основного куба и двух боковых стенок. Чтобы создать ее, выполните следующие действия.

1. Добавьте на сцену куб. Присвойте ему имя **Ground** и расположите в позиции с координатами (0, 0, 15,5), задайте вращение (0, 180, 0) и масштаб (10, 0,5, 50).

2. Добавьте на сцену четырехугольник под названием **Wall**, поместите его в позицию с координатами (–5,25, 1,2, 15,5), задайте вращение (0, −90, 0) и масштаб (50, 2, 1). Продублируйте объект Wall и поместите новый объект в позицию с координатами (5,25, 1,2, 15,5).

3. В папке **Materials**, которую вы импортировали ранее, рассмотрите материалы **Ground** и **Wall**. Материал **Ground** слегка металлический, у него есть тайлы текстур. Один из встроенных шейдеров частиц придает материалу **Wall** эффект свечения (рис. 20.2).

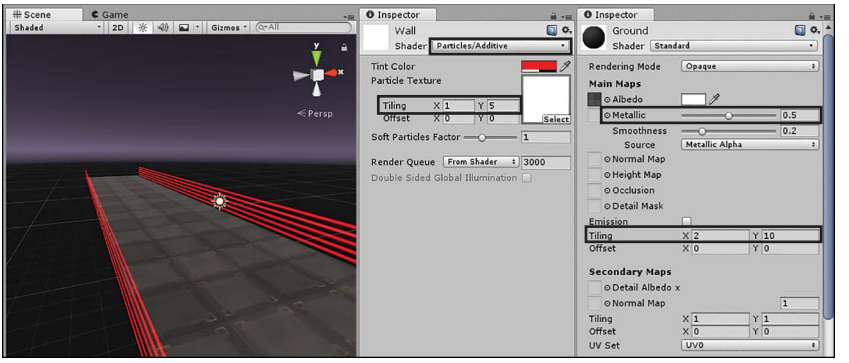


Рис. 20.2. Материалы для полосы

4. Перетащите материал **Ground** на объект **Ground**. Перетащите материал **Wall** на оба игровых объекта **Wall**.

Готово! С тоннелем все просто.

Прокрутка полосы препятствий

В часе 15 вы видели, как можно прокручивать фон, создав два экземпляра этого фона и чередуя их. В игре «Бег с препятствиями» мы воспользуемся более умным решением. Каждый материал имеет набор смещений текстуры. Вы можете увидеть их на панели **Inspector**, когда материал выбран (прямо под свойством **Tiling**). Нам предстоит изменять эти смещения во время игры с помощью скрипта. Если текстура установлена на повтор (по умолчанию), то она будет прокручиваться бесшовно. В результате, если все сделано правильно, объект будет визуально прокручиваться. Для создания этого эффекта выполните следующие действия.

1. Создайте новую папку с именем **Scripts**. Создайте новый скрипт под названием **TextureScroller**. Присоедините его к земле.

2. Добавьте в скрипт следующий код (заменив методы `Start()` и `Update()`, которые там есть):

```
public float speed = .5f;

Renderer renderer;
float offset;

void Start()
{
    renderer = GetComponent<Renderer>();
}

void Update()
{
    // Увеличение смещения в зависимости от времени
    offset += Time.deltaTime * speed;
    // Смещение в диапазоне от 0 до 1
    if (offset > 1)
        offset -= 1;
    // Применение смещения к материалу
    renderer.material.mainTextureOffset = new Vector2(0, offset);
}
```

3. Запустите сцену и обратите внимание на прокрутку полосы. Мы применили простой и эффективный способ создания прокрутки 3D-объекта.

Игровые сущности

Теперь, когда у нас есть подвижный мир, настало время настроить объекты: игрока, бонусы, препятствия и зону-триггер. Зона-триггер будет использоваться для уничтожения элементов, прошедших мимо игрока. Создавать точку спауна нам не нужно. Вместо этого мы рассмотрим другой способ реализации спауна: позволим игровому менджеру создавать бонусы и препятствия.

Бонусы

Бонусы в этой игре будут выглядеть как простые сферы с некоторыми эффектами. Мы создадим и разместим сферу, а затем превратим ее в префаб. Чтобы создать бонусы, выполните следующие действия.

1. Добавьте на сцену сферу. Поместите ее в позицию с координатами (0, 1,5, 42), задайте масштаб (0,5, 0,5, 0,5). Добавьте к сфере компонент **Rigidbody** и сбросьте флажок **Use Gravity**.

2. Создайте новый материал под названием **Powerup** и задайте ему желтый цвет. Определите свойству **Metallic** значение **0**, а свойству **Smoothness** — значение 1. Нанесите материал на сферу.

3. Добавьте в сферу точечный источник света (выбрав команду **Add Component** ⇒ **Rendering** ⇒ **Light**). Задайте свету желтый цвет.

4. Добавьте в сферу систему частиц (выбрав команду **Component** ⇒ **Effects** ⇒ **Particle System**). Если у частиц будет странный пурпурный цвет — не волнуйтесь. Дальше мы скажем, что с этим делать.

5. В основном модуле частиц задайте свойству **Start Lifetime** значение **0,5**, свойству **Start Speed** — значение **–5**, свойству **Start Size** — значение **0,3**, а свойству **Start Color** — светло-желтый цвет.

6. В модуле **Emission** задайте свойству **Rate over Time** значение **30**. В модуле **Shape** задайте свойству **Shape** значение **Sphere**, а свойству Radius — значение **1**.

7. В модуле **Renderer** задайте свойству **Render Mode** значение **Stretched Billboard** и свойству **Length Scale** — значение **5**.

8. Создайте новую папку с именем **Prefabs**. Присвойте сфере имя **Powerup** и перетащите ее с панели **Hierarchy** в папку **Prefabs**. Затем удалите бонус со сцены.

ВНИМАНИЕ!

Проблемы с частицами

В зависимости от версии Unity, при добавлении системы частиц для бонуса вы можете увидеть странные цветные квадраты вместо частиц. Если это произошло, вам необходимо вручную применить материал **Default Particle** к свойству модуля **Renderer** компонента **Particle System**. Чтобы сделать это, щелкните мышью по кружку рядом со свойством **Material** и выберите материал **Default-Particle** из списка (рис. 20.3).

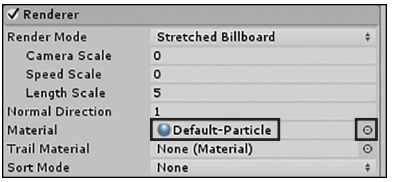


Рис. 20.3. Назначение материала **Default-Particle**

Настроив положение объекта до создания префаба, вы можете создать экземпляр префаба в этом месте. То есть точка спауна теперь не нужна.

На рис. 20.4 показан готовый бонус (хотя рисунок не передает всей красоты).

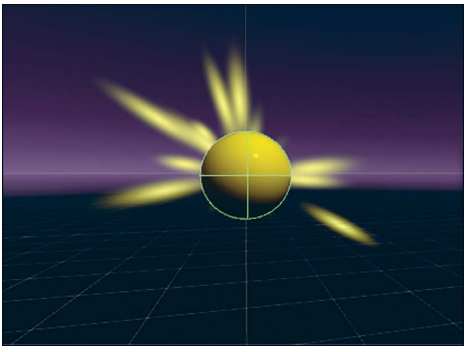


Рис. 20.4. Бонус

Препятствия

В этой игре препятствия будет представлены небольшими светящимися красными кубиками. Игрок сможет либо увернуться, либо проплыть сквозь них, если «исчезнет». Чтобы создать препятствия, выполните следующие действия.

1. Добавьте на сцену куб и присвойте ему имя **Obstacle**. Поместите его в позицию с координатами (0, 0,4, 42), задайте масштаб (1, 0,2, 1). Добавьте к кубу компонент **Rigidbody** и сбросьте флажок **Use Gravity**.

2. Добавьте препятствию компонент **Light** и задайте ему красный цвет.
3. Создайте новый материал под названием **Obstacle** и примените его к препятствию. Сделайте цвет материала красным, установите флажок **Emission**, задайте темно-красный цвет излучения (рис. 20.5).
4. Перетащите игровой объект **Obstacle** в папку **Prefabs**, чтобы превратить его в префаб. Удалите объект **Obstacle** со сцены.

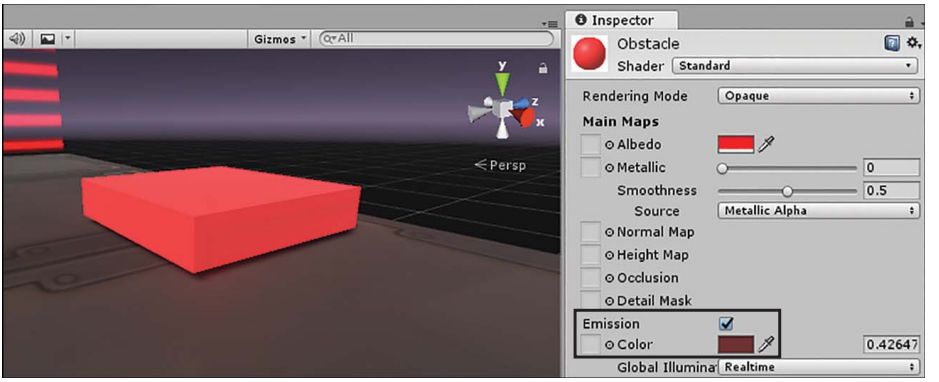


Рис. 20.5. Препятствие и материал

Зона-триггер

Как и в созданных вами ранее играх, нам понадобится зона-триггер, чтобы уничтожать объекты, прошедшие мимо игрока. Для ее создания выполните следующие действия.

1. Добавьте на сцену куб. Присвойте ему имя **TriggerZone** и поместите его в позицию с координатами (0, 1, −20), задав масштаб (10, 1, 1).
2. Для компонента **Box Collider** этого куба установите флажок **Is Trigger**.

Игрок

Создание игрока — это значительная часть работы. Он должен быть анимирован, управляем и иметь пользовательский шейдер. Мы говорили о шейдерах ранее, но еще не применяли пользовательские шейдеры. Прежде чем мы двинемся далее, давайте сделаем игрока.

1. В прилагаемых файлах примеров для этого часа найдите папки **Models** и **Animations** и перетащите их на панель **Project**, чтобы импортировать.
2. В папке **Models** выберите модель **Player.fbx**. Как мы говорили ранее, это ассет из доступной для свободного скачивания игры Adventure компании Unity Technologies.

3. На вкладке **Rig** измените тип анимации на **Humanoid**. Нажмите кнопку **Apply**. Теперь вы должны увидеть галочку рядом с кнопкой **Configure**. (Если вам нужно вспомнить, о чем здесь речь, обязательно повторите час 18.)

4. Перетащите модель игрока на сцену и расположите ее в позиции с координатами (0, 0,25, −8,5). Назначьте игровому объекту **Player** тег **Player**. (Напоминание: тег объекта можно найти в верхнем левом раскрывающемся списке на панели **Inspector**.)

5. Добавьте игроку капсульный коллайдер (выбрав команду **Add Component** ⇒ **Physics** ⇒ **Capsule Collider**). Установите флажок **Is Trigger**. Наконец, задайте свойству **Center Y** значение **0,7**, свойству **Radius** — значение **0,3** и свойству **Height** — значение **1,5.**

Теперь нужно подготовить и применить анимацию **Run**.

1. В папке **Animations** выберите файл **Runs.fbx**. На панели **Inspector** перейдите на вкладку **Rig** и измените тип анимации на **Humanoid**. Нажмите кнопку **Apply**.

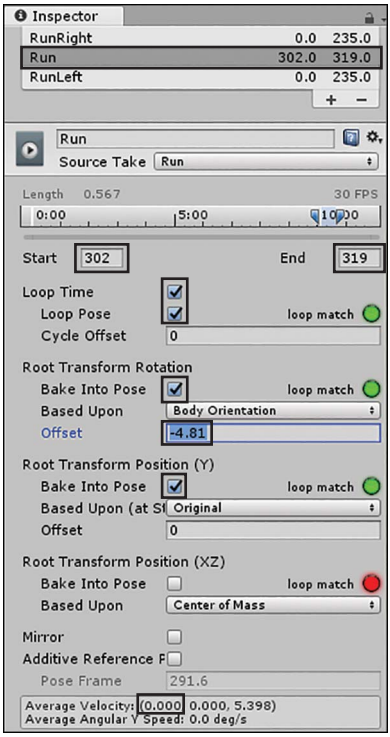


Рис. 20.6. Свойства анимации **Run**

2. На вкладке **Animations** обратите внимание на три клипа: **RunRight**, **Run** и **RunLeft**. Выберите клип **Run** и настройте свойства, как показано на рис. 20.6. (Чтобы избежать сноса, важно, чтобы значение по оси *x* параметра **Averag Velocity** было равно **0**.) Нажмите кнопку **Apply**.

3. Раскройте содержимое ассета модели **Runs.fbx**, щелкнув мышью по стрелке в правой части ассета на панели **Project**. Найдите клип **Run** и перетащите его на модель **Player** на сцене. Если вы сделали все правильно, в папке рядом с **Runs.fbx** появится контроллер аниматора под названием **Player**.

Если вы запустите сцену сейчас, вы увидите пару проблем. Во-первых, игрок убегает вдаль. Но ведь нам нужно лишь создать иллюзию того, что игрок перемещается, а на самом деле он не должен двигаться. Вторая проблема заключается в том, что анимация воспроизводится слишком быстро, и в результате ноги игрока скользят по земле. Выполните следующие действия, чтобы устранить эти проблемы.

1. Чтобы удалить движение и настроить игрока бежать на месте, выберите на сцене игровой объект **Player**. Для компонента **Animator** сбросьте флажок **Apply Root Motion**.

2. На панели **Project** дважды щелкните по контроллеру аниматора **Player** (в папке **Animations**), чтобы открыть панель **Animator**.

3. На панели **Animator** выберите состояние **Run**. На панели **Inspector** присвойте свойству **Speed** значение **0,7** и установите флажок **Foot IK** (рис. 20.7). Запустите вашу сцену и посмотрите, что изменилось в поведении игрока.

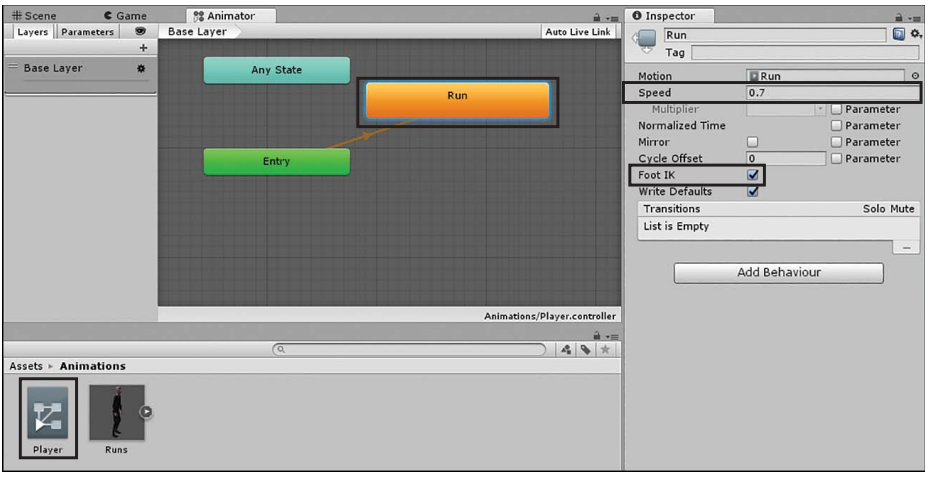


Рис. 20.7. Свойства состояния **Run**

Игрок готов (пока). В следующих разделах мы добавим код, который позволит создать в игре некоторые интересные функциональные возможности.

Элементы управления

Настало время добавить элементы управления и интерактивность, чтобы игра стала, собственно, игрой. Поскольку позиции бонусов и препятствий уже заданы в префабах, нам нет необходимости создавать точку спауна. Таким образом, почти все управление будет возложено на менеджера игры.

Скрипт зоны-триггера

Первый скрипт, который вы должны сделать, предназначен для зоны триггера. Помните, что зона триггера уничтожает предметы, которые проходят мимо игрока. Создайте новый скрипт под названием **TriggerZone** и прикрепите его к игровому объекту зоны триггера. Поместите в скрипт следующий код:

```
void OnTriggerEnter(Collider other)
{
    Destroy(other.gameObject);
}
```

Скрипт крайне прост — он только уничтожает любой объект, с которым контактирует триггер.

Скрипт менеджера игры

Скрипт менеджера игры отвечает за бóльшую часть происходящего. Для начала создайте пустой игровой объект и присвойте ему имя **Game Manager**. Он будет контейнером для наших скриптов. Создайте новый скрипт под названием **GameManager** и прикрепите его к объекту **Game Manager**. В приведенном далее коде для менеджера есть некоторые сложности, так что не забудьте внимательно прочитать каждую строку, чтобы понять, что она делает. Добавьте в скрипт следующий код:

```
public TextureScroller ground;
public float gameTime = 10;

float totalTimeElapsed = 0;
bool isGameOver = false;

void Update()
{
    if (isGameOver)
        return;
```

```
    totalTimeElapsed += Time.deltaTime;
    gameTime -= Time.deltaTime;

    if (gameTime <= 0)
        isGameOver = true;
}

public void AdjustTime(float amount)
{
    gameTime += amount;
    if (amount < 0)
        SlowWorldDown();
}

void SlowWorldDown()
{
    // Отмена «ускорения» мира
    // Мир замедляется на 1 секунду
    CancelInvoke();
    Time.timeScale = 0.5f;
    Invoke("SpeedWorldUp", 1);
}

void SpeedWorldUp()
{
    Time.timeScale = 1f;
}

// Обратите внимание, что здесь используется старая система интерфейсов Unity
void OnGUI()
{
    if (!isGameOver)
    {
        Rect boxRect = new Rect(Screen.width / 2-50, Screen.height - 100,
                100, 50);
        GUI.Box(boxRect, "Time Remaining");
        Rect labelRect = new Rect(Screen.width / 2-10, Screen.height - 80,
                20, 40);
        GUI.Label(labelRect, ((int)gameTime).ToString());
    }
    else
    {
        Rect boxRect = new Rect(Screen.width / 2-60, Screen.height / 2-100,
                120, 50);
        GUI.Box(boxRect, "Game Over");
        Rect labelRect = new Rect(Screen.width / 2-55, Screen.height / 2-80,
                90, 40);
        GUI.Label(labelRect, "Total Time: " +(int)totalTimeElapsed);
```

```
            Time.timeScale = 0;
        }
}
```

ПРИМЕЧАНИЕ

Старая система UI

Обратите внимание, что, как и в «Великолепном гонщике» (который мы создали в часе 6), игра «Бег с препятствиями» использует старую систему графического интерфейса Unity. В большинстве случаев вы не стали бы использовать такой интерфейс для реальной игры, но тут мы вернемся к нему, чтобы сэкономить время и силы. Не волнуйтесь: вы получите возможность добавить новый интерфейс самостоятельно, о чем мы скажем в конце этого часа.

Помните, что одно из правил игры заключается в том, что все замедляется, когда игрок попадает в препятствие. Мы реализуем это за счет изменения переменной `Time.timeScale` в течение игры. Остальные переменные будут отвечать за время и состояния в игре.

Метод `Update()` следит за временем. Он добавляет время с момента последнего кадра (`Time.deltaTime`) к переменной `totalTimeElapsed`. Он также проверяет, завершилась ли игра — а это происходит, когда время становится равно 0. Если игра закончена, метод устанавливает флаг `isGameOver`.

Методы `SlowWorldDown()` и `SpeedWorldUp()` работают в сочетании друг с другом. Всякий раз, когда игрок попадает в препятствие, вызывается метод `SlowWorldDown()`. Он замедляет время. Затем вызывается метод `Invoke()`, который говорит: «вызови указанный здесь метод через *x* секунд», при этом вызываемый метод указывается в кавычках, а время в секундах. Вы могли заметить вызов метода `CancelInvoke()` в начале метода `SlowWorldDown()`. Это отменяет любые методы `SpeedWorldUp()`, ожидающие вызова после столкновения с препятствием. В предыдущем коде через 1 секунду вызывается метод `SpeedWorldUp()`. Он ускоряет игру, возвращая ее к нормальному темпу.

Метод `AdjustTime()` вызывается всякий раз, когда игрок сталкивается с бонусом или препятствием. Он изменяет количество оставшегося времени. Если значение изменения отрицательно (препятствие), данный метод вызывает метод `SlowWorldDown()`.

Наконец, метод `OnGUI` выводит на экран оставшееся и прошедшее время игры.

Скрипт игрока

Скрипт **Player** имеет две функции: управление движением и столкновениями игрока и управление исчезновением. Создайте новый скрипт под названием

Player и прикрепите его к игровому объекту **Player** на сцене. Добавьте в скрипт следующий код:

```
[Header("References")]
public GameManager manager;
public Material normalMat;
public Material phasedMat;

[Header("Gameplay")]
public float bounds = 3f;
public float strafeSpeed = 4f;
public float phaseCooldown = 2f;

Renderer mesh;
Collider collision;
bool canPhase = true;

void Start()
{
    mesh = GetComponentInChildren<SkinnedMeshRenderer>();
    collision = GetComponent<Collider>();
}

void Update()
{
    float xMove = Input.GetAxis("Horizontal") * Time.    deltaTime *
                strafeSpeed;

    Vector3 position = transform.position;
    position.x += xMove;
    position.x = Mathf.Clamp(position.x, -bounds, bounds);
    transform.position = position;

    if (Input.GetButtonDown("Jump") && canPhase)
    {
        canPhase = false;
        mesh.material = phasedMat;
        collision.enabled = false;
        Invoke("PhaseIn", phaseCooldown);
    }
}

void PhaseIn()
{
    canPhase = true;
    mesh.material = normalMat;
    collision.enabled = true;
}
```

В этом скрипте используются так называемые *атрибуты*. Атрибуты — это специальные теги, которые модифицируют код. Как вы видите, в данном коде задействован атрибут `Header`, с помощью которого на панели **Inspector** выводится строка заголовка (проверьте это в редакторе.)

Первые три переменные содержат ссылки на менеджера игры и два материала. Когда игрок исчезает, материалы меняются местами. Остальные переменные обрабатывают настройки игры — границы уровня и скорость бокового движения игрока.

Метод `Update()` начинается с перемещения игрока за счет управления. Затем он проверяет, не вышел ли игрок за пределы тоннеля. Это выполняется с помощью метода `Mathf.Clamp()`, который не дает игроку покидать коридор. Затем метод `Update()` проверяет, не нажимал ли игрок клавишу **Пробел** (который **Input Manager** называет «`Jump`»). Если пользователь нажимает клавишу **Пробел**, игрок исчезает (то есть его коллайдер отключается), и игрок какое-то время становится прозрачным для препятствий и теряет способность подбирать бонусы.

Скрипт Collidable

И бонусы, и препятствия должны двигаться в сторону игрока. Они также изменяют время игры, когда игрок сталкивается с ними. Таким образом, вы можете применить к ним один и тот же скрипт. Создайте скрипт под названием **Collidable** и добавьте его к префабам бонуса и препятствия. Вы можете сделать это, выделив оба на панели **Inspector** и выбрав команду **Add Component ⇒ Scripts ⇒ Collidable**. Добавьте в скрипт следующий код:

```
public GameManager manager;
public float moveSpeed = 20f;
public float timeAmount = 1.5f;

void Update()
{
    transform.Translate(0, 0, -moveSpeed * Time.deltaTime);
}

void OnTriggerEnter(Collider other)
{
    if (other.tag == "Player")
    {
        manager.AdjustTime(timeAmount);
        Destroy(gameObject);
    }
}
```

Этот скрипт прост. Задаются переменные для скрипта менеджера игры — скорость движения и величина изменения времени. При каждом вызове метода `Update()`

объект перемещается. При столкновении объекта с чем-либо он проверяет, не столкнулся ли он с игроком. Если это так, скрипт сообщает менеджеру игры, а затем уничтожает объект.

Скрипт Spawner

Скрипт **Spawner** отвечает за создание объектов в сцене. Поскольку данные о положении заданы в префабах, вам не нужен отдельный спаунер — достаточно написать скрипт для объекта **Game Manager**. Создайте новый скрипт под названием **Spawner** и прикрепите его к объекту **Game Manager**. Добавьте в скрипт следующий код:

```
public GameObject powerupPrefab;
public GameObject obstaclePrefab;
public float spawnCycle = .5f;

GameManager manager;
float elapsedTime;
bool spawnPowerup = true;

void Start()
{
    manager = GetComponent<GameManager>();
}

void Update()
{
    elapsedTime += Time.deltaTime;
    if (elapsedTime > spawnCycle)
    {
        GameObject temp;
        if (spawnPowerup)
            temp = Instantiate(powerupPrefab) as GameObject;
        else
            temp = Instantiate(obstaclePrefab) as GameObject;

        Vector3 position = temp.transform.position;
        position.x = Random.Range(-3f, 3f);
        temp.transform.position = position;
        Collidable col = temp.GetComponent<Collidable>();
        col.manager = manager;

        elapsedTime = 0;
        spawnPowerup = !spawnPowerup;
    }
}
```

Этот скрипт содержит ссылки на игровые объекты «бонус» и «препятствие». Следующие переменные управляют временем и порядком появления объектов.

Бонусы и препятствия будут порождаться поочередно, и специальный флаг станет следить за тем, что именно должно порождаться.

В методе `Update()` увеличивается счетчик истекшего времени, а затем проверяется, пора ли порождать новый объект. Если пора — скрипт проверяет, какой объект нужно породить. Затем появляется бонус или препятствие. Новый объект смещается вправо или влево на случайную величину, а также получает ссылку на менеджера игры. Затем в методе `Update()` уменьшается истекшее время, а флаг, отвечающий за тип порождаемого объекта, меняется на противоположный.

Собираем все воедино

Теперь вы готовы перейти к последнему этапу работы над игрой. Нужно связать между собой скрипты и объекты. Сначала выделите объект **Game Manager** на панели **Hierarchy**. Перетащите объект **Ground** к его соответствующему свойству в компоненте **Game Manager Script** (рис. 20.8). Перетащите префабы **Powerup** и **Obstacle** на соответствующие свойства в компоненте **Spawner (Script)**.

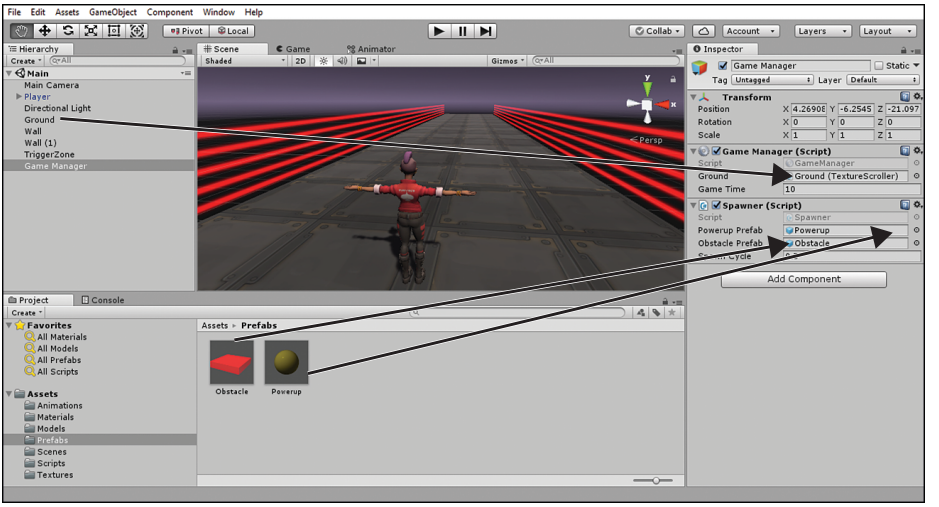


Рис. 20.8. Перетаскивание объектов к их свойствам

Теперь выделите объект **Player** на панели **Hierarchy** и перетащите объект **Game Manager** на свойство **Manager** компонента **Player (Script)** (рис. 20.9). Кроме того, если вы заглянете в папку **Model** на панели **Project**, то увидите материал **PhasedOut**. Рядом с ним находится пользовательский шейдер **Hologram**. Рассмотрите его, если вам интересно, но имейте в виду, что разработка собственных шейдеров — сложная штука, и в этой книге не рассматривается. Встроенные визуальные средства для создания шейдеров в Unity в порядке эксперимента добавлены

в версии Unity 2018.1. Когда вы будете готовы, перетащите материалы **Player** и **PhasedOut** на соответствующие свойства компонента **Player (Script)**.

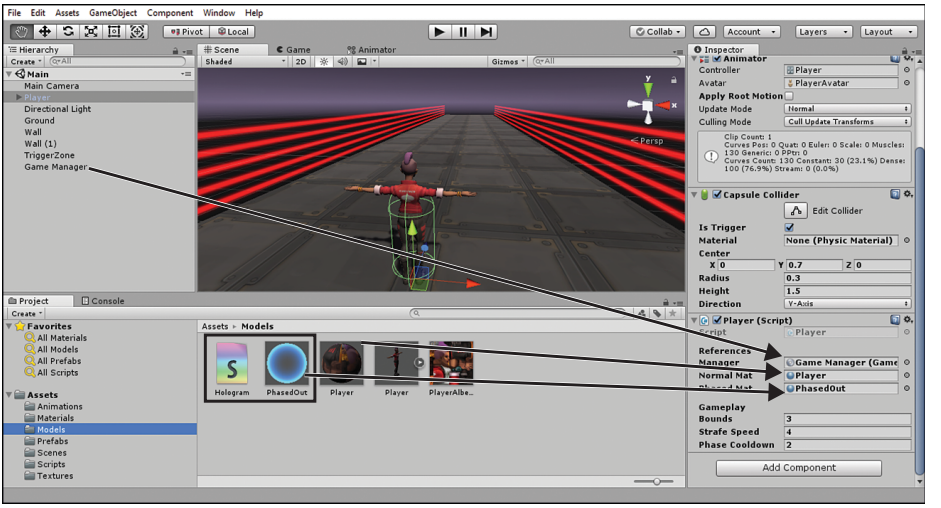


Рис. 20.9. Реализация управления игрой и добавление материалов в скрипты игрока

Наконец, выберите префаб **Obstacle** и задайте свойству **Time Amount** скрипта **Collidable** значение **−0,5**. Готово, можно играть!

Простор для совершенствования

Как и всегда, игра не может быть полностью закончена, пока вы не проверите и не отшлифуете ее. Теперь настало время поиграть и посмотреть, что вам нравится, а что нет. Не забывайте следить за функциями, которые, по-вашему, действительно идут игре на пользу. Точно так же обратите внимание на отрицательные моменты. Не забывайте делать заметки о любых идеях, которые возникают у вас относительно будущих итераций игры. Попробуйте завлечь ваших друзей поиграть, запишите их мнения. Все это поможет сделать игру уникальной и более приятной.

Резюме

В этом часе мы создали игру «Бег с препятствиями». Мы начали с разметки основных элементов проекта игры. Затем мы создали коридор и эффект прокрутки за счет небольшого трюка с текстурами. Потом мы сделали игровые сущности, элементы управления и написали скрипты. И последнее, но важное, — мы протестировали игру и даже собрали отзывы.

Вопросы и ответы

Вопрос: **Движение объектов и пола под ногами в игре «Бег с препятствиями» не совпадает, это нормально?**

Ответ: В нашем случае да. Для идеальной синхронизации требуется глубокое тестирование и настройка. Это еще один момент, который вы могли бы усовершенствовать.

Вопрос: **Продолжительность «исчезновения» равна времени восстановления — значит, игрок может быть прозрачным всегда?**

Ответ: Да. Но тогда игра продлится всего 10 секунд, так как игрок не может собирать бонусы, будучи прозрачным.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Каково условие поражения в игре «Бег с препятствиями»?
2. Как работает прокрутка фона?
3. Как игра «Бег с препятствиями» управляет скоростью всех объектов на сцене?

Ответы

1. Игрок проигрывает, когда заканчивается время.
2. Коридор остается неподвижным, а прокручивается текстура. В результате нам кажется, что игрок движется.
3. Игра изменяет временной масштаб всей игры в целом. Но обратите внимание, что бонусы и препятствия при этом движутся так же, как и до этого. В чем дело?

Упражнение

Настало время попытаться внедрить изменения, которые вы наверняка наметили во время тестирования. Вам нужно сделать игру уникальной. Будем надеяться, что вы сумели выявить какие-нибудь недостатки игры или что-то еще, что можно было бы улучшить. Например, можно сделать следующее.

1. Попробуйте добавить другие бонусы и препятствия.
2. Попробуйте избавиться от использования старого GUI и перейти к новой системе интерфейсов Unity.
3. Попробуйте увеличить или уменьшить сложность, варьируя частоту появления бонусов или препятствий. Можно изменить количество времени, добавляемое бонусом, длительность замедления, эффект замедления мира и даже скорость отдельных объектов.
4. Придайте бонусам и препятствиям более интересный вид. Поэкспериментируйте с текстурами и эффектами частиц, чтобы улучшить его.
5. Попробуйте вывести на экран пройденное расстояние. Можно также постепенно увеличивать скорость игры, чтобы со временем проходить становилось сложнее.

21-Й ЧАС
Работа со звуком

Что вы узнаете в этом часе

- Основы работы со звуком в Unity
- Как использовать источники звука
- Как управлять звуковым сопровождением с помощью скриптов
- Как использовать аудиомикшеры

В этом часе мы изучим работу со звуком в Unity. Начнем с основ, затем исследуем компонент источника звука и его работу. Кроме того, мы рассмотрим отдельные звуковые клипы и их роль в этом процессе. Затем вы узнаете, как управлять звучанием с помощью скриптов, а в конце часа мы поработаем с микшерами.

Введение в работу со звуком

Большинство переживаний включает в себя также звуки, связанные с ними. Представьте себе, что на фильм ужасов будет наложен закадровый смех. То, что должно быть страшным, станет смешным. В видеоиграх звук имеет очень большое значение, хотя игроки не всегда осознают это в полной мере. Удар колокола подсказывает, что игрок открывает секрет. Грохот пушек добавляет нотки реализма к баталиям.

В Unity легко добавить в игру звуковые эффекты.

Как устроено звуковое сопровождение

Чтобы озвучить сцену, вам нужны три компонента: **Audio Listener** (слушатель), **Audio Source** (источник) и аудиоклип (сам звук). **Audio Listener** — главный компонент аудиосистемы, он «слушает» происходящее на сцене. Это как бы ваши «уши» в игровом мире. По умолчанию на вновь созданной сцене такой компонент прилагается к **Main Camera** (рис. 21.1). У компонента **Audio Listener** нет свойств, и для его активации ничего делать не нужно.

Обычно компонент **Audio Listener** прикрепляется к игровому объекту, представляющему собой игрока. Если вы помещаете компонент **Audio Listener** на любой другой объект, нужно удалить его из объекта **Main Camera**. На сцене единовременно может быть только один компонент **Audio Listener**.

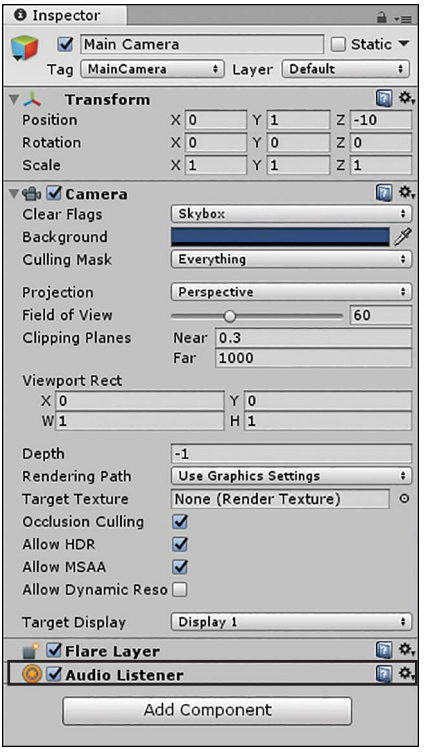


Рис. 21.1. Компонент **Audio Listener**

Компонент **Audio Listener** именно слышит звук, но не производит его. А вот компонент **Audio Source**, который можно поместить на любой объект на сцене (даже на объект с **Audio Listener**), занимается именно этим. У него есть много свойств и настроек, и мы рассмотрим их в отдельном разделе в конце этого часа.

Последнее, что нужно для работы со звуком, это *аудиоклип* — звуковой файл, который воспроизводится компонентом **Audio Source**. У аудиоклипов есть определенные свойства, которые влияют на параметры воспроизведения. Unity поддерживает следующие аудиоформаты: *.aif*, *.aiff*, *.wav*, *.mp3*, *.ogg*, *.mod*, *.it*, *.s3m* и *.xm*.

Компоненты **Audio Listener**, **Audio Source** и аудиоклип вместе позволяют нашей игре звучать.

2D- и 3D-Audio

Сначала мы рассмотрим разницу между 2D- и 3D-звуком. 2D-аудиоклипы применяются чаще всего. Они воспроизводятся с одинаковой громкостью, независимо от близости компонента **Audio Listener** к компоненту **Audio Source** на сцене. 2D-звуки лучше всего использовать для меню, предупреждений, саундтрека или любых других сигналов, которые всегда должны звучать одинаково. Это преимущество 2D-звуков одновременно является их самым большим недостатком. Если каждый звук в вашей игре будет воспроизводиться с одинаковой громкостью, независимо от вашего местонахождения, играть станет просто неприятно и нереалистично.

3D-звуки позволяют решить эту проблему. Такие клипы воспроизводятся тише или громче, в зависимости от того, насколько близко компонент **Audio Listener** находится к компоненту **Audio Source**. В сложных аудиосистемах, таких как в Unity, 3D-звуки могут даже имитировать эффект Доплера (подробнее об этом позже). Если вам нужно добиться реализма на насыщенной звуками сцене — вам нужен 3D-звук.

Размерность каждого аудиоклипа задается на компоненте **Audio Source**, который его воспроизводит.

Компонент Audio Source

Как упоминалось ранее в этом часе, **Audio Source** — это компонент, который воспроизводит на сцене аудиоклипы. Звучание 3D-звука определяется расстоянием между компонентами **Audio Source** и **Audio Listener**. Чтобы добавить компонент **Audio Source** к игровому объекту, выделите нужный объект и выберите команду **Add Component** ⇒ **Audio** ⇒ **Audio Source**.

Компонент **Audio Source** имеет ряд свойств, которые предоставляют мощный уровень контроля над воспроизведением звука на сцене, они приведены в таблице 21.1. В дополнение к ним в разделе **3D Sound Settings**, который мы рассмотрим чуть позже, есть ряд параметров, применимых к 3D-аудиоклипам.

ТАБЛ. 21.1. Свойства компонента Audio Source

Свойство	Описание
Audio Clip	Позволяет выбрать фактически воспроизводимый звуковой файл
Output	(Опционально) Позволяет выводить звучание клипа в аудиомикшер
Mute	Определяет, будет ли данный звук отключен
Bypass Effects	Определяет, будут ли к данному источнику применяться звуковые эффекты. Выбор этого свойства отключает эффекты

Bypass Listener Effects	Определяет, будут ли к данному источнику применяться звуковые эффекты компонента **Audio Listener**. Выбор этого свойства отключает эффекты
Bypass Reverb Zones	Определяет, будут ли к данному источнику применяться звуковые эффекты зон реверберации. Выбор этого свойства отключает эффекты
Play On Awake	Определяет, будет ли источник начинать воспроизведение звука при запуске сцены
Loop	Определяет, будет ли источник звука перезапускать аудиоклип после окончания воспроизведения
Priority	Задает приоритет источника звука: 0 для наиболее важного, 255 — для наименее важного. Значение 0 используется для звуков, которые должны играть всегда
Volume	Задает громкость звучания, значение 1 соответствует 100%
Pitch	Тон звучания
Stereo Pan	Задает положение поля стереокомпонента 2D-звука
Spatial Blend	Задает, как 3D-движок влияет на источник звука. Это свойство определяет, относится звук к типу 2D или 3D
Reverb Zone Mix	Задает величину выходного сигнала, передаваемого в зону реверберации

ПРИМЕЧАНИЕ

Приоритет звуков

В каждой системе предусмотрено конечное число звуковых каналов. Оно непостоянно и зависит от многих факторов, таких как аппаратные средства и операционная система компьютера. Поэтому в большинстве аудиосистем используется система приоритетов, где звуки воспроизводятся в том порядке, который им задан, пока не будет задействовано максимальное число каналов. После того, как все каналы заняты, звуки с более низким приоритетом будут вытесняться звуками высокого приоритета. Главное помнить, что в Unity меньшее число означает более высокий фактический приоритет!

Импорт аудиоклипов

Компоненту **Audio Source** нечего воспроизводить, если нет аудиоклипа. В Unity импорт аудиофайлов осуществляется так же легко, как импорт всего остального. Вам нужно перетащить требуемые файлы на панель **Project**, чтобы добавить их к ассетам проекта. Звуковые файлы, используемые в этом часе, любезно предоставлены Джереми Генделем (**handelabra.com**).

ПРАКТИКУМ

Раз, раз, проверка звука...

В этом упражнении мы попробуем добавить звук в Unity и проверим, как все работает. Не забудьте сохранить сцену, так как она понадобится нам позже в этом часе. Выполните следующие действия.

1. Создайте новый проект или сцену. Найдите папку **Sounds** в файлах примеров для часа 21 и перетащите ее на панель **Project** в Unity, чтобы импортировать.
2. Создайте куб и поместите его в позицию с координатами (0, 0, 0).
3. Добавьте к кубу компонент **Audio Source** (выбрав команду **Add Component** ⇒ **Audio** ⇒ **Audio Source**).
4. Найдите файл **looper.ogg** среди импортированных звуков и перетащите его на свойство **Audio Clip** куба (рис. 21.2).
5. Убедитесь, что флажок **Play On Awake** установлен, и запустите сцену. Обратите внимание на воспроизведение звука. Звучание должно прекратиться примерно через 20 секунд (если вы не зададите повтор, конечно).

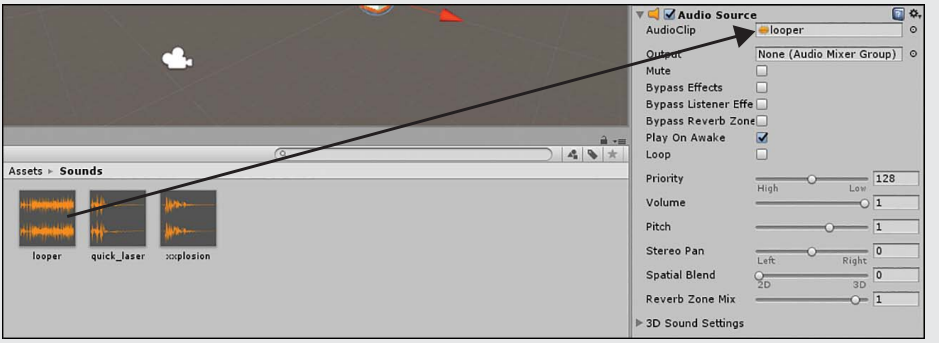


Рис. 21.2. Добавление аудиоклипа для компонента **Audio Source**

ПРИМЕЧАНИЕ

Кнопка Mute Audio

В верхней части панели **Game** есть кнопка **Mute Audio** (между кнопками **Maximize on Play** и **Stats**). Если звуков не слышно, проверьте, не нажата ли эта она.

Тестирование звука на панели Scene

Сложновато и невыгодно запускать сцену каждый раз, когда нужно проверить звук. Мало этого, так еще надо найти необходимый звук в потенциально огромном мире, что не всегда легко или даже возможно. Так что лучше проверять звук на панели **Scene**.

Для этого вам необходимо включить озвучку сцены: щелкните мышью по переключателю, показанному на рис. 21.3, и появится виртуальный компонент **Audio Listener**. Он располагается в вашей системе отсчета на панели **Scene** (а не там, где находится реальный компонент **Audio Listener** на сцене).

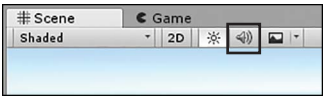


Рис. 21.3. Переключатель звука

ПРАКТИКУМ

Добавление звука на сцену

В этом упражнении мы проверим звучание на панели **Scene**. Нам понадобится сцена, созданная в предыдущем практикуме. Если вы еще не подготовили ее, сделайте это сейчас. Выполните следующие действия.

1. Откройте сцену, созданную в предыдущем практикуме.
2. Активируйте переключатель звука на сцене (рис. 21.3).
3. Перемещайтесь по сцене. Обратите внимание, что громкость одинаковая независимо от расстояния до куба, излучающего звук. По умолчанию все источники воспроизводят 2D-звук.
4. Перетащите ползунок **Spatial Blend** в сторону **3D** (рис. 21.4). Попробуйте снова перемещаться по сцене. Обратите внимание, что теперь звук становится тише, когда вы отдаляетесь.

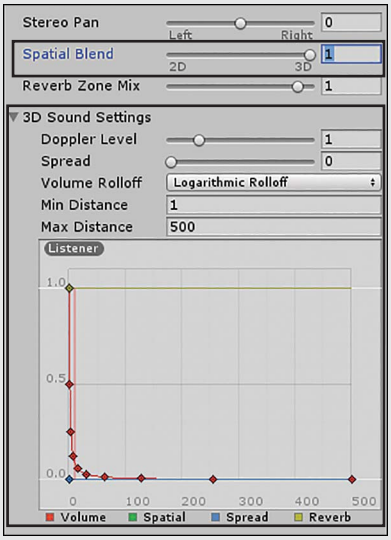


Рис. 21.4. Настройка 3D-звучания

3D-звук

Как упоминалось ранее в этом часе, все звуки — 2D по умолчанию. Переключиться на 3D-звучание легко: присвойте параметру **Spatial Blend** значение **1**, при этом все звуки будут также зависеть от 3D-эффектов, и здесь уже играет роль расстояние и движение. Эффекты настраиваются с помощью элементов управления в разделе **3D Sound Settings** компонента звука (см. рис. 21.4).

В таблице 21.2 описаны различные свойства 3D-звука.

СОВЕТ

Использование графика

Экспериментируя с настройками в разделе **3D Sound Settings**, посмотрите на график. Он расскажет вам, как громко звук будет воспроизводиться на различных расстояниях — это отлично помогает при настройке. Как раз то обстоятельство, когда картина красноречивее тысячи слов!

ТАБЛ. 21.2. Свойства 3D-звука

Свойство	Описание
Doppler Level	Определяет силу эффекта Доплера (звук искажается, когда вы движетесь к нему или от него). При значении 0 эффект не будет применяться
Spread	Определяет разброс динамиков системы. При значении 0 все динамики находятся в одном месте, и сигнал, по существу, монофонический. Оставьте это свойство по умолчанию, если вы плохо разбираетесь в аудиосистемах
Volume Rolloff	Определяет, как сильно громкость звука изменяется на расстоянии. Значение по умолчанию — **Logarithmic**. Вы также можете выбрать вариант **Linear** или настроить собственную кривую затухания
Min Distance	Расстояние от источника, на котором звук играет с громкостью 100%. Чем выше число, тем больше расстояние
Max Distance	Определяет предел, за которым звук уже не слышен

2D-звук

Иногда нужно, чтобы определенный звук воспроизводился независимо от его положения на сцене. Наиболее распространенный пример — фоновая музыка. Чтобы переключить звучание компонента **Audio Source** из режима 3D в 2D, перетащите ползунок **Spatial Blend** в сторону значения **2D** (оно используется

по умолчанию, см. рис. 21.4). Обратите внимание, что вы также можете установить ползунок в позицию между значениями **2D** и **3D**, то есть звук будет всегда слышен в некоторой степени, независимо от расстояния.

Описанные выше свойства, такие как **Priority**, **Volume**, **Pitch** и прочие, относятся и к 2D-, и к 3D-звукам. Настройки в разделе **3D Sound Settings** применяются только к 3D-звукам.

Скрипты озвучки

Воспроизведение звука созданным компонентом **Audio Source** — это, безусловно, круто, при условии, что мы того и добивались. Если вы хотите, чтобы звук воспроизводился в определенное время, или вам нужно воспроизводить различные звуки с помощью одного компонента **Audio Source**, потребуются скрипты. К счастью, управлять звуками с помощью кода не слишком сложно. В большинстве случаев это работает на тех же принципах, что и любой аудиоплеер, к которому вы привыкли: выберите песню и нажмите «Воспроизведение». Управление звуками с помощью скриптов осуществляется через переменные и методы в составе класса `Audio`.

Начало и прекращение звучания

Чтобы работать со звуками в скриптах, первое, что вам нужно сделать, это получить ссылку на компонент **Audio Source**. Используйте следующий код:

```
AudioSource audioSource;
void Start ()
{
    // Поиск источника звука на кубе
    audioSource = GetComponent<AudioSource> ();
}
```

Данная ссылка хранится в переменной `AudioSource`, и вы можете вызывать по этой ссылке методы. Базовые функции, которые нам нужны, отвечают за запуск и прекращение вопроизведения аудиоклипа. Эти действия контролируются двумя методами — `Start()` и `Stop()`. Их использование выглядит следующим образом:

```
audioSource.Start();    // Запускает клип
audioSource.Stop();     // Останавливает клип
```

Данный код будет воспроизводить клип, указанный в качестве значения свойства **Audio Clip** компонента **Audio Source**. Вы также можете запускать клип с некоторой задержкой. Для этого используется метод `PlayDelayed()`, который принимает один параметр — время ожидания перед воспроизведением клипа в секундах. Выглядит это так:

```
audioSource.PlayDelayed(<длительность в секундах>);
```

Можно определить, воспроизводится ли клип в данный момент, проверив значение переменной `isPlaying`, которая относится к объекту `audioSource`. Чтобы получить доступ к этой переменной, используйте следующий код:

```
if(audioSource.isPlaying)
{
    // Клип воспроизводится
}
```

Значение этой переменной `True`, если звук воспроизводится, и `False`, если это не так.

ПРАКТИКУМ

Запуск и остановка воспроизведения

Выполните следующие шаги, чтобы использовать скрипты для запуска и остановки воспроизведения звукового клипа.

1. Откройте сцену, созданную в практикуме «Добавление звука на сцену».
2. На игровом объекте **Cube**, созданном ранее, найдите компонент **Audio Source**. Сбросьте флажок **Play On Wake** и установите флажок **Loop**.
3. Создайте папку с именем **Scripts**, а в ней — скрипт под названием **AudioScript**. Прикрепите его к кубу и замените весь код на следующий:

```
using UnityEngine;

public class AudioScript: MonoBehaviour
{
    AudioSource audioSource;

    void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }

    void Update()
    {
        if (Input.GetButtonDown("Jump"))
        {
            if (audioSource.isPlaying == true)
                audioSource.Stop();
            else
                audioSource.Play();
        }
    }
}
```

4. Запустите сцену. Вы можете запускать и останавливать звук, нажимая клавишу **Пробел**. Обратите внимание, что аудиоклип вопроизводится с начала каждый раз, когда вы запускаете его.

СОВЕТ

Неупомянутые свойства

Все свойства компонента **Audio Source**, которые перечислены на панели **Inspector**, доступны для скриптов. Например, свойство **Loop** — с помощью переменной `audioSource.loop`.

Как упоминалось ранее, все эти переменные используются в сочетании со звуковым объектом. Попробуйте найти что-нибудь еще!

Смена аудиоклипов

С помощью скриптов вы можете легко задать, какие именно аудиоклипы должны воспроизводиться. Для этого нужно менять значение свойства **Audio Clip** в коде, прежде чем использовать метод `Play()`. Всегда нужно останавливать воспроизведение текущего аудиоклипа перед запуском нового, иначе клип не переключится.

Чтобы сменить аудиоклип компонента **Audio Source**, нужно назначить переменную типа `AudioClip` переменной `clip` объекта `audioSource`. Например, если у вас есть аудиоклип с именем `newClip`, вы можете назначить его компоненту **Audio Source** и воспроизводить, используя следующий код:

```
audioSource.clip = newClip;
audioSoure.Play();
```

Вы можете легко создать подборку звуковых клипов и переключать их с помощью скриптов. Мы займемся этим в упражнении в конце часа.

Аудиомикшеры

Мы уже видели, как звук воспроизводится компонентом **Audio Source** и слушается компонентом **Audio Listener**. Этот процесс крайне прост, но мы рассматривали «сферический случай в вакууме». Мы одновременно воспроизводили только один аудиоклип или небольшое их количество. Трудности возникают тогда, когда приходится настраивать громкость и эффекты каждого звука. Весьма мучительно постоянно искать и настраивать каждый компонент **Audio Source** на сцене или в префабе. И вот тут нам нужны аудиомикшеры. *Аудиомикшер* — это такой ассет, который работает буквально как диджейский пульт, — позволяет точно управлять звучанием.

Создание аудиомикшеров

Создать и использовать аудиомикшер проще простого. Сперва щелкните правой кнопкой мыши по панели **Project** и выберите команду **Create** ⇒ **Audio Mixer**. Затем дважды щелкните мышью по созданному ассету микшера, чтобы открыть панель **Audio Mixer** (рис. 21.5). Как вариант, это можно сделать командой **Window** ⇒ **Audio Mixer**.

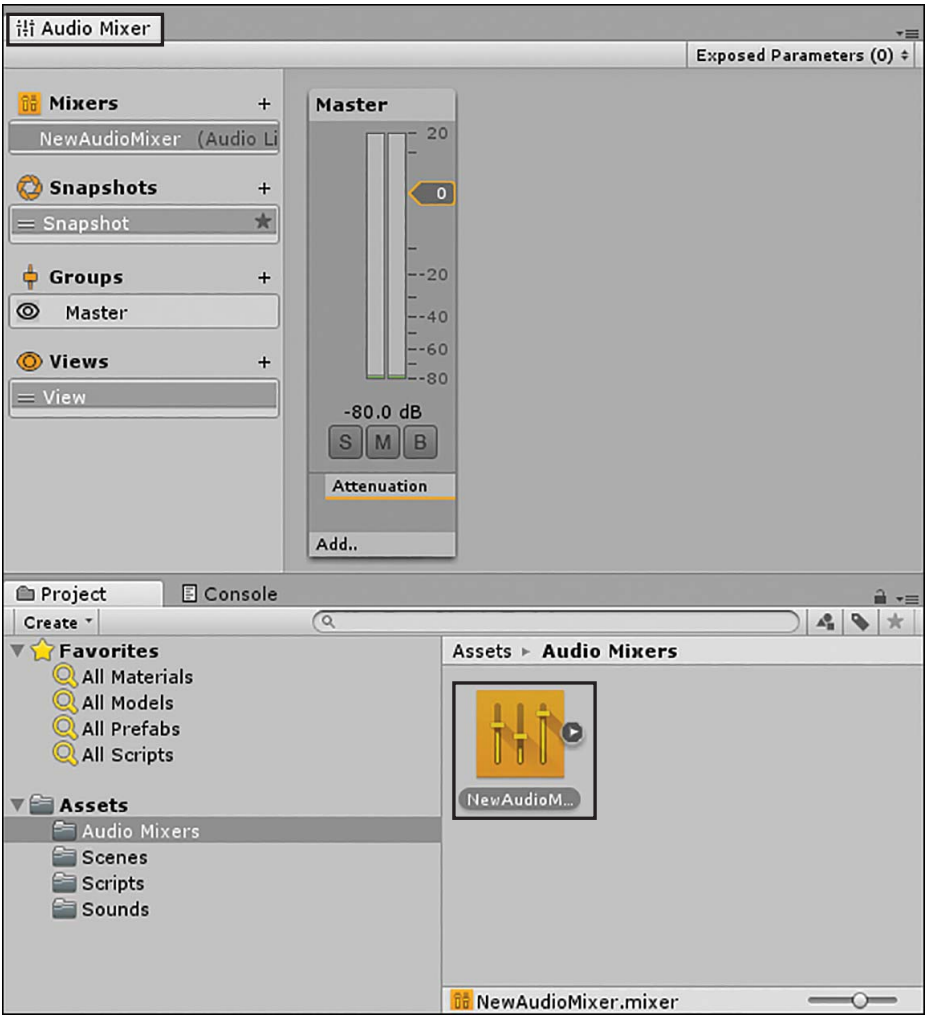


Рис. 21.5. Панель **Audio Mixer**

Перенаправление звука в аудиомикшер

Теперь у вас есть аудиомикшер, и нужно пропустить звуки через него. Для этого требуется настроить выход компонента **Audio Source** на одну из групп аудиомикшера. По умолчанию, микшер имеет только одну группу, которая называется **Master**. Вы можете добавить несколько дополнительных групп, чтобы лучше организовать работу (рис. 21.6). Группы могут даже представлять собой целые микшеры и реализовывать модульный подход к управлению звуком.

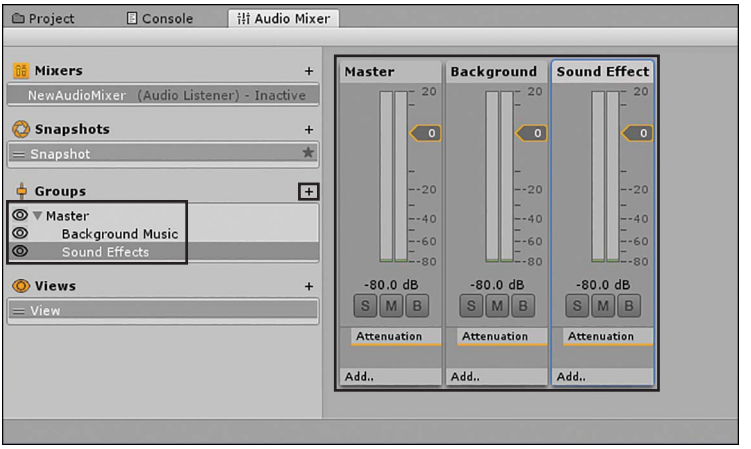


Рис. 21.6. Добавление групп

Создав нужные группы аудиомикшера, укажите в свойстве **Output** компонента **Audio Source** свою группу (рис. 21.7). Теперь вы сможете использовать аудиомикшер для регулировки громкости и эффектов аудиоклипа. Кроме того, микшер имеет приоритет над свойством **Spatial Blend** компонента **Audio Source** и обрабатывает звук как 2D. Используя аудиомикшер, вы можете регулировать уровень громкости и эффектов всех источников звука одновременно (рис. 21.7).

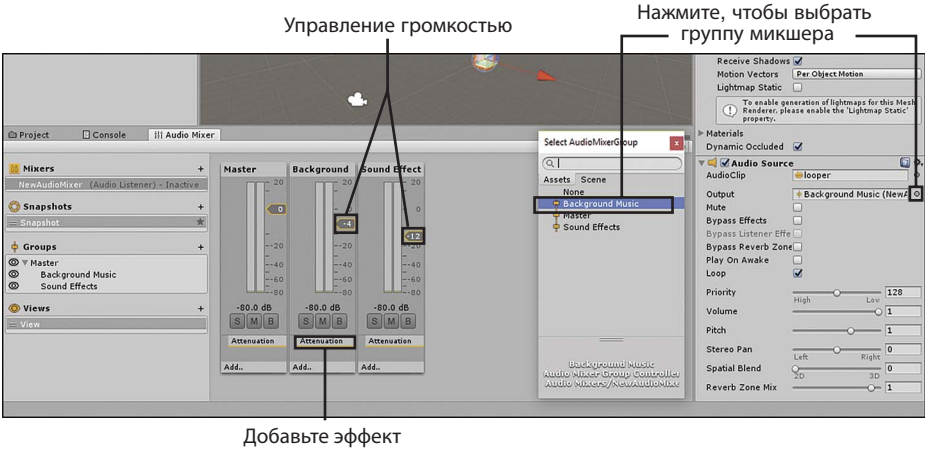


Рис. 21.7. Настройка маршрутизации звука

Резюме

В этом часе вы узнали об озвучке проектов в Unity. Мы начали с основ работы со звуковыми файлами и необходимыми компонентами. Затем мы изучили компонент **Audio Source**. Вы узнали, как проверить звук на панели **Scene** и как использовать 2D- и 3D-звучание. В конце часа вы научились управлять звучанием с помощью скриптов и аудиомикшеров.

Вопросы и ответы

Вопрос: **Сколько в среднем аудиоканалов используется в системе?**

Ответ: Индивидуально для каждой системы. Большинство современных игровых платформ позволяет воспроизводить одновременно десятки и даже сотни аудиоклипов. Главное знать свою целевую платформу и грамотно использовать систему приоритетов.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Какие компоненты нужны для работы звука?
2. Верно или нет: 3D-звуки воспроизводятся с одинаковой громкостью, независимо от расстояния между компонентами **Audio Source** и **Audio Listener**?
3. Какой метод позволяет воспроизводить аудиоклип после некоторой задержки?

Ответы

1. Компоненты **Audio Source** и **Audio Listener**, а также сам аудиоклип.
2. Нет. Это 2D-звуки воспроизводятся с одинаковой громкостью независимо от расстояния между компонентами **Audio Source** и **Audio Listener**.
3. `PlayDelayed()`.

Упражнение

В этом упражнении мы создадим простую звуковую карту. Она позволит вам воспроизводить один из трех звуков. Вы также сможете запускать и прекращать звучание и включать/выключать повторное воспроизведение.

1. Создайте новый проект или сцену. Добавьте на сцену куб, расположите его в позиции с координатами (0, 0, −10) и добавьте к нему компонент **Audio Source**. Обязательно сбросьте флажок **Play On Awake**. Найдите папку **Sounds** в файлах примеров для часа 21 и перетащите ее в папку **Assets**.

2. Создайте папку **Scripts** и скрипт под названием **AudioScript**. Прикрепите его к кубу и замените содержимое следующим кодом:

```
using UnityEngine;

public class AudioScript: MonoBehaviour
{
    public AudioClip clip1;
    public AudioClip clip2;
    public AudioClip clip3;

    AudioSource audioSource;

    void Start()
    {
        audioSource = GetComponent<AudioSource>();
        audioSource.clip = clip1;
    }

    void Update()
    {
        if (Input.GetButtonDown("Jump"))
        {
            if (audioSource.isPlaying == true)
                audioSource.Stop();
            else
                audioSource.Play();
        }

        if (Input.GetKeyDown(KeyCode.L))
        {
            audioSource.loop =!audioSource.loop; // управление
                                                 зацикливанием
        }

        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
```

```
            audioSource.Stop();
            audioSource.clip = clip1;
            audioSource.Play();
        }
        else if (Input.GetKeyDown(KeyCode.Alpha2))
        {
            audioSource.Stop();
            audioSource.clip = clip2;
            audioSource.Play();
        }
            else if (Input.GetKeyDown(KeyCode.Alpha3))
        {
            audioSource.Stop();
            audioSource.clip = clip3;
            audioSource.Play();
        }
    }
}
```

3. В редакторе Unity выберите куб на сцене. Перетащите аудиофайлы **looper.ogg**, **quick_laser.ogg** и **xxplosion.ogg** из папки **Sounds** на свойства **Clip1**, **Clip2** и **Clip3** скрипта.

4. Запустите сцену. Теперь вы можете переключать звуки с помощью клавиш **1**, **2** и **3**. Вы также можете управлять звучанием, нажимая клавишу **Пробел**. Наконец, вы можете включать/выключать повторное воспроизведение с помощью клавиши **L**.

22-Й ЧАС

Разработка для мобильных устройств

Что вы узнаете в этом часе

- Что нужно для мобильной разработки
- Как использовать акселерометр устройства
- Как использовать сенсорный дисплей устройства

Мобильные устройства, такие как смартфоны и планшеты, — это популярные игровые девайсы. В этом часе мы изучим разработку мобильных приложений с помощью Unity для устройств под управлением операционных систем Android и iOS. Сначала мы рассмотрим требования к разработке мобильных приложений. Затем мы научимся принимать входные данные от акселерометра устройства и использовать сенсорный дисплей.

ПРИМЕЧАНИЕ

Требования

В этом часе мы рассматриваем исключительно разработку для мобильных устройств. Поэтому, если у вас нет мобильного устройства (под управлением операционной системы iOS или Android), понять практические упражнения в полной мере у вас не получится. Но это не так страшно: вы все равно сможете сделать мобильную игру, хотя и не сможете поиграть в нее.

Подготовка к мобильной разработке

В Unity разработка для мобильных устройств выполняется очень просто. Вам будет приятно узнать, что она практически идентична разработке для других платформ. Самая большая разница в том, что на мобильных платформах, как правило, нет клавиатуры или мыши. Если иметь это в виду при разработке, вы сможете развернуть однажды созданную игру на любой платформе. Такой уровень

кросс-платформенных возможностей невероятно крут. Прежде чем приступить к работе с мобильными устройствами в Unity, вам потребуется выполнить кое-какие настройки на компьютере.

ПРИМЕЧАНИЕ

Разнообразие устройств

Существует множество различных мобильных устройств. На момент публикации этой книги компания Apple выпускала два класса игровых мобильных устройств: iPad и iPhone/iPod. Под управлением операционной системы Android работает бессчетное количество смартфонов и планшетов, и некоторые устройства — под управлением Windows. У каждого из этих типов устройств слегка отличается аппаратная часть и, соответственно, настройка. В этом уроке мы продемонстрируем процедуру установки. Невозможно написать точное руководство, которое подошло бы для всех устройств. На самом деле уже выпущено несколько руководств по разработке в Unity для устройств под управлением операционной системы iOS и Android (Google), в которых все это описано подробнее и лучше. Мы будем ссылаться на них при необходимости.

Настройка среды

Прежде чем открывать редактор Unity и создавать игру, вам нужно настроить среду разработки. Специфика настройки различается в зависимости от целевого устройства и желаемого результата, но общие шаги смотрите ниже.

1. Установите набор инструментальных средств разработки программного обеспечения (SDK) для целевого устройства.
2. Убедитесь, что компьютер распознает и может работать с этим устройством (но это важно только если вы собираетесь тестировать игру непосредственно на устройстве).
3. Разрабатывая под Android-устройства, нужно также сообщить Unity, где расположен набор SDK.

Если эти шаги кажутся вам сложноватыми, не волнуйтесь. Существует множество ресурсов, которые помогут вам с решениями данных задач. Лучшее всего начать с документации Unity, которая доступна по адресу **docs.unity3d.com/ru/current/Manual/UnityManual.html**.

Как вы видите на рис. 22.1, в документации Unity есть указания по разработке для устройств под управлением операционных систем iOS и Android. Эти документы обновляются по мере того, как в программу вносятся изменения. Если вы не планируете тестирование на устройстве, переходите к следующему разделу. В противном случае сначала выполните действия по настройке среды разработки, описанные в документации Unity.

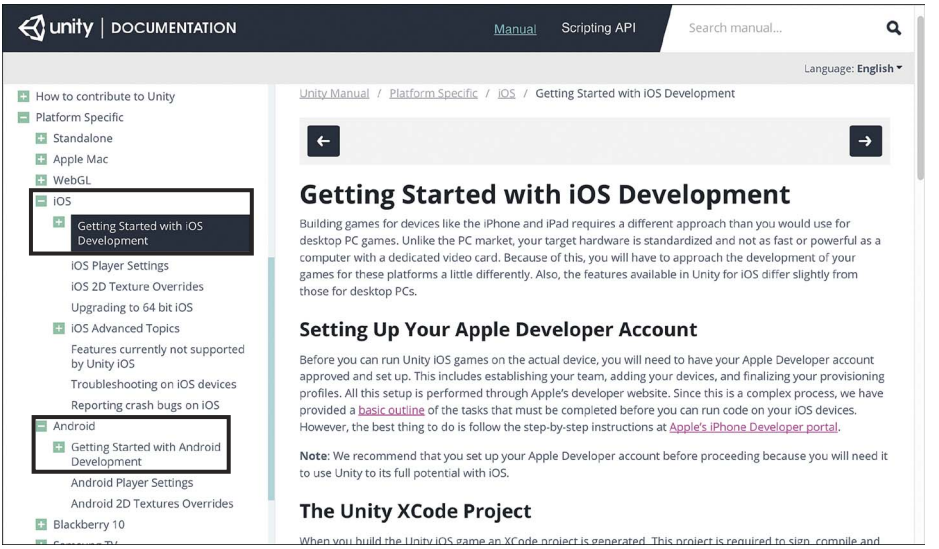


Рис. 22.1. Документация по платформам

Unity Remote

Самый простой способ проверить игру на устройстве — собрать проект, скинуть получившиеся файлы на устройство, а затем запустить игру. Это довольно трудоемко и быстро надоедает. Еще один способ — создать игру, а затем запустить ее в эмуляторе устройства, работающего под управлением операционной системы IOS или Android. Но это тоже трудоемкий процесс, для которого, прежде всего, нужно настроить эмулятор. Эти системы могут быть полезны, если вы проводите обширное тестирование производительности, рендеринга и прочего. Для простейшего тестирования есть способ получше: использовать *Unity Remote*.

Unity Remote — это приложение, которое можно скачать из магазина приложений вашего мобильного устройства. Приложение позволяет проверять проекты на вашем мобильном устройстве, работая в редакторе Unity. В двух словах, это означает, что вы можете проверять игру на устройстве в режиме реального времени *одновременно* с разработкой, а также использовать устройство для передачи входных данных с устройства в игру. Более подробную информацию о Unity Remote можно найти по ссылке **docs.unity3d.com/Manual/UnityRemote5.html**.

Введите запрос *Unity Remote* в строке поиска магазина приложений вашего устройства. Там вы сможете скачать и установить его так же, как любое другое приложение (см. рис. 22.2).

После установки приложение Unity Remote работает как дисплей и контроллер для вашей игры. Вы сможете использовать его для передачи данных о касаниях,

сигналах акселерометра и сенсорного дисплея в Unity. Это очень круто, потому что позволяет проверять игры (хотя бы в базовом понимании) без установки мобильных наборов SDK или специализированных средств разработчика.

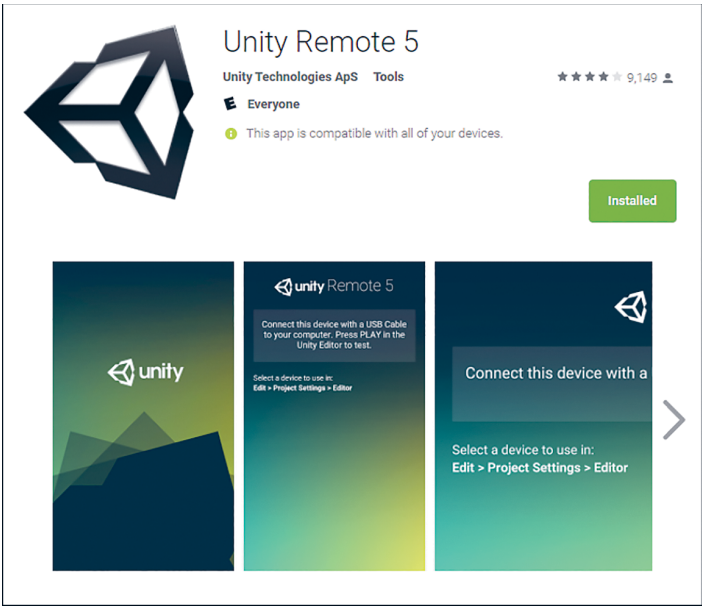


Рис. 22.2. Unity Remote в магазине Google Play

ПРАКТИКУМ

Настройка устройства для тестирования

Выполнив упражнение, вы корректно настроите свою среду мобильной разработки. Здесь вы будете использовать Unity Remote на вашем устройстве, чтобы взаимодействовать со сценой в Unity. Если у вас нет настроенного устройства, вы не сможете выполнять все эти действия, но сможете получить представление о том, что в принципе происходит. Если что-то не работает, значит, ваша среда не настроена должным образом. Выполните следующие действия.

1. Создайте новый проект или сцену и добавьте кнопку пользовательского интерфейса в центр экрана.
2. На панели **Inspector** задайте свойству **Pressed Color** кнопки красный цвет.
3. Запустите сцену и убедитесь, что при нажатии кнопка меняет свой цвет. Остановите сцену.
4. Подключите мобильное устройство к компьютеру с помощью USB-кабеля. Когда компьютер распознает устройство, запустите приложение Remote Unity на устройстве.

5. В редакторе Unity выполните команду **Edit** ⇒ **Project Settings** ⇒ **Editor** и выберите тип устройства в раскрывающемся списке **Unity Remote** ⇒ **Device** панели **Inspector**. Обратите внимание, что, если мобильная платформа не задана в Unity (Android или iOS), эта опция в настройках редактора отображаться не будет.
6. Запустите сцену снова. Спустя секунду вы должны увидеть интерфейс с кнопкой на вашем мобильном устройстве. Теперь вы можете нажать кнопку на экране устройства, чтобы изменить ее цвет.

Акселерометры

Большинство современных мобильных устройств имеет встроенные акселерометры. Акселерометр передает информацию (отдельно по трем осям) об ориентации устройства в пространстве, то есть наклонено устройство или лежит на ровной поверхности. На рис. 22.3 показаны оси акселерометра мобильного устройства и как они расположены в портретной ориентации.

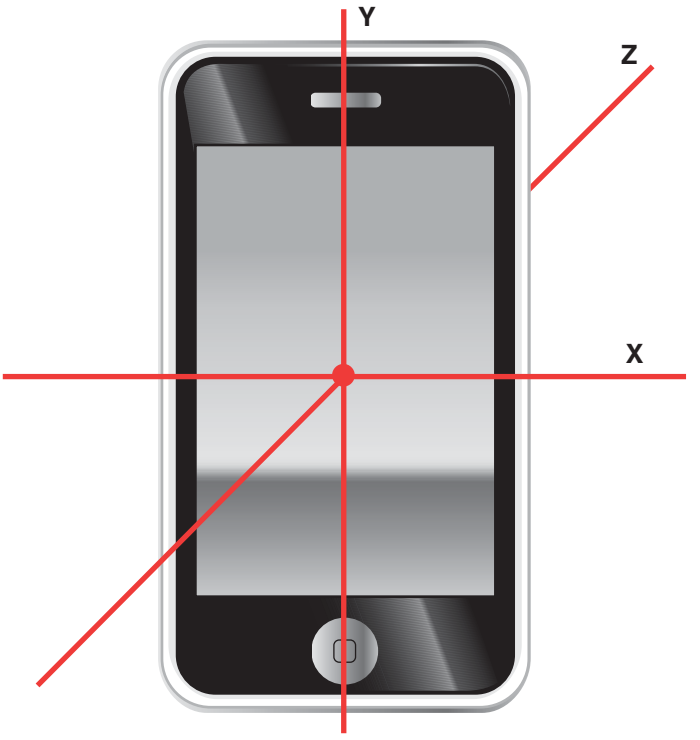


Рис. 22.3. Оси акселерометра

Как вы видите на рис. 22.3, оси устройства по умолчанию совпадают с 3D-осями в Unity, если вы будете держать устройство вертикально в портретной ориентации прямо перед собой. В другой ориентации вам будет необходимо преобразовать данные акселерометра. Например, если телефон, изображенный на рис. 22.3, у вас в альбомной ориентации (то есть боком), вы должны будете использовать ось *x* акселерометра телефона как ось *y* в Unity.

Проектирование игр для акселерометра

Чтобы использовать акселерометр мобильного устройства при разработке, нужно учитывать несколько моментов. Во-первых, вы можете одновременно использовать только две оси акселерометра. Причина этого заключается в том, что независимо от ориентации устройства, одна ось всегда будет находиться под воздействием силы тяжести. Рассмотрим ориентацию устройства на рис. 22.3. Видно, что если оси *x* и *z* управляются наклоном устройства, ось *y* будет передавать отрицательные значения, так как гравитация тянет ее вниз. Если повернуть телефон так, чтобы он лежал на поверхности лицевой стороной вверх, вы можете использовать только оси *x* и *y*. В этом случае ось *z* находится под воздействием силы тяжести.

Во-вторых, акселерометр передает не очень точные данные. Мобильные устройства считывают значения через определенный интервал времени, а сами значения всегда приближенные. В результате графики данных акселерометра могут быть прерывистыми и неравномерными. В этом случае нужно перемещать зависящие от акселерометра объекты плавно, либо брать среднее значение наклона из последних нескольких показаний. Кроме того, акселерометры передают входные значения от –1 до +1 при полном вращении устройства на 180 градусов. Никто не играет в игры, наклоняя устройство полностью, так что входные значения, как правило, меньше, чем с клавиатуры (например, от –0,5 до 0,5).

Использование акселерометра

Чтение значений с акселерометра выполняется так же, как с любого другого пользовательского ввода: с помощью скриптов. Все, что вам нужно сделать, это считывать значения из переменной Vector3 с именем `acceleration`, относящейся к объекту `Input`. Таким образом вы можете получить доступ к данным по всем трем осям, введя следующий код:

```
Input.acceleration.x;
Input.acceleration.y;
Input.acceleration.z;
```

Используя эти значения, вы можете управлять игровыми объектами.

ПРИМЕЧАНИЕ

Несовпадение осей

При использовании информации с акселерометра в сочетании с Unity Remote вы можете заметить, что оси располагаются не так, как описано ранее. Это связано с тем, что Unity Remote ориентирует игру в зависимости от соотношения сторон. То есть Unity Remote автоматически отображается в альбомной ориентации (длинным краем устройства параллельно земле) и преобразовывает оси. Поэтому, когда вы используете Unity Remote, ось *x* располагается вдоль длинного края устройства, а ось *y* — вдоль короткого. Это может показаться странным, но, скорее всего, вы будете использовать устройство именно так, и вопросов будет меньше.

ПРАКТИКУМ

Перемещение куба силой мысли... ну, или телефона

В этом упражнении вы будете использовать акселерометр мобильного устройства, чтобы перемещать куб по сцене. Очевидно, что для этого вам потребуется настроенное и подключенное мобильное устройство с акселерометром. Выполните следующие действия.

1. Создайте новый проект или сцену. (Если вы создадите новый проект, не забудьте изменить настройки редактора, как описано в предыдущем разделе.) Добавьте на сцену куб и поместите его в позицию с координатами (0, 0, 0).
2. Создайте новый скрипт под названием **AccelerometerScript** и прикрепите его к кубу. Поместите следующий код в метод `Update()` скрипта:

```
float x = Input.acceleration.x * Time.deltaTime;
float z = -Input.acceleration.z * Time.deltaTime;
transform.Translate(x, 0f, z);
```

3. Убедитесь в том, что мобильное устройство подключено к компьютеру. Держите устройство параллельно земле и запустите приложение Unity Remote. Запустите сцену. Обратите внимание, что куб перемещается, когда вы наклоняете телефон. Также проанализируйте, какие оси телефона перемещают куб по осям *x* и *z*.

Сенсорный ввод

Мобильные устройства обычно управляются емкостными сенсорными экранами, они позволяют определить, когда и где пользователь касается дисплея, и, как правило, могут отслеживать несколько касаний одновременно. Точное число отслеживаемых касаний различно в зависимости от устройства.

Касание экрана — это не только координаты касания с точки зрения устройства. На самом деле, о каждом касании хранится много информации. В Unity касание сохраняется в переменной `Touch`. Это означает, что каждый раз, когда вы

касаетесь экрана, генерируется переменная Touch. Она будет существовать до тех пор, пока ваш палец контактирует с экраном. Если вы проводите пальцем по экрану, переменная Touch отслеживает это. Переменные Touch хранятся вместе в коллекции под названием touches, которая входит в объект Input. Если в данный момент нет касаний экрана, коллекция touches пуста. Для доступа к ней можно использовать следующий код:

```
Input.touches;
```

С помощью коллекции touches вы можете перебирать переменные Touch для обработки данных. Это будет выглядеть примерно так:

```
foreach(Touch touch in Input.touches)
{
    // Действия
}
```

В таблице 22.1 перечислены все свойства переменных типа Touch.

ТАБЛ. 22.1. Свойства переменной Touch

Свойство	Описание
deltaPosition	Изменение позиции касания с момента последнего обновления. Полезно для обнаружения экранного жеста (проведения по экрану)
deltaTime	Количество времени, которое прошло с момента последнего изменения касания
fingerId	Уникальный индекс касания. Может принимать значения, например, от 0 до 4 на устройствах, которые позволяют обрабатывать пять касаний одновременно
phase	Текущая фаза прикосновения: Began, Moved, Stationary, Ended или Canceled (начато, перемещено, неподвижно, закончено, отменено)
position	Координата точки касания на экране
tapCount	Число касаний на экране

Эти свойства могут быть использованы для управления сложными взаимодействиями между пользователем и игровыми объектами.

ПРАКТИКУМ

Отслеживание касаний

В этом упражнении вы будете отслеживать касания и выводить их данные на экран. Очевидно, что для этого вам нужно настроить и подключить мобильное устройство, поддерживающее мультисенсорный ввод. Выполните следующие действия.

1. Создайте новый проект или сцену.
2. Создайте новый скрипт под названием **TouchScript** и прикрепите его к объекту **Main Camera**. Поместите в скрипт следующий код:

```
void OnGUI()
{
    foreach (Touch touch in Input.touches)
    {
        string message = "";
        message += "ID: " +touch.fingerId + "\n";
        message += "Phase: " +touch.phase.ToString() + "\n";
        message += "TapCount: " +touch.tapCount + "\n";
        message += "Pos X: " +touch.position.x + "\n";
        message += "Pos Y: " +touch.position.y + "\n";

        int num = touch.fingerId;
        GUI.Label(new Rect(0 + 130 * num, 0, 120, 100), message);
    }
}
```

3. Убедитесь, что мобильное устройство подключено к компьютеру. Запустите сцену. Коснитесь экрана и обратите внимание на информацию, которая будет выведена в Unity (рис. 22.4). Подвиньте палец и посмотрите, как изменятся данные. Теперь прикоснитесь несколькими пальцами одновременно. Поводите ими по экрану и оторвите от экрана в случайном порядке. Как отслеживается каждый контакт в отдельности? Сколько касаний обрабатывается одновременно?

Рис. 22.4. Вывод сенсора на экране

ВНИМАНИЕ!

Работаем по команде!

В практикуме «Отслеживание касаний» вы создали метод `OnGUI()`, который собирает информацию о различных касаниях на экране. Часть кода, где строка `message` составляется из сенсорных данных, не так уж проста. Выполнение слишком большого количества обработок в методе `OnGUI()` может значительно снизить эффективность проекта, такого следует избегать. Этот метод применен здесь исключительно для наглядности. Всегда помещайте обновляемый код в метод `Update()`. Кроме того, вам пригодится новый пользовательский интерфейс, потому что он работает намного быстрее.

Резюме

В этом часе вы узнали об использовании Unity для разработки игр для мобильных устройств. Сначала мы настроили среду разработки для устройств под управлением операционных систем Android и iOS. Затем вы научились работать с акселерометром устройства. В конце часа мы поэкспериментировали с отслеживанием сенсорных касаний в Unity.

Вопросы и ответы

Вопрос: **Могу ли я создать игру один раз и развернуть ее на все основные платформы, включая мобильные?**

Ответ: Конечно! Но нужно помнить, что мобильные устройства, как правило, не имеют такой большой вычислительной мощности, как компьютеры. Таким образом, у пользователей мобильных устройств могут возникнуть некоторые проблемы с производительностью, если в игре много ресурсоемкой обработки или эффектов. Вы должны убедиться, что ваша игра работает эффективно, если планируете развертку на мобильных платформах.

Вопрос: **Каковы различия между устройствами под управлением операционных систем iOS и Android?**

Ответ: С точки зрения Unity большой разницы между ними нет. Обе операционные системы рассматриваются как мобильные устройства. Но имейте в виду, что существуют некоторые различия между отдельно взятыми устройствами (например, мощность процессора, время автономной работы, операционная система), которые могут повлиять на игру.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Какой инструмент позволяет передавать входные сигналы с устройства в Unity, когда сцена запущена?
2. Сколько осей акселерометра можно реально использовать одновременно?
3. Сколько касаний устройство может обрабатывать одновременно?

Ответы

1. Приложение Unity Remote.
2. Две оси. Третья ось всегда находится под действием силы тяжести, в зависимости от того, как устройство ориентировано в пространстве.
3. Это полностью зависит от устройства. В последний раз я проверял устройство под управлением операционной системы iOS, которое отслеживало 21 касание. Этого достаточно для всех ваших пальцев рук и ног, и не только!

Упражнение

В этом упражнении мы попробуем перемещать объекты по сцене с помощью сенсорного дисплея мобильного устройства. Очевидно, что для этого вам нужно настроить и подключить мобильное устройство, поддерживающее мультисенсорный ввод. Если это не сделано, то вы можете, по меньшей мере, прочитать упражнение.

1. Создайте новый проект или сцену. Выберите команду **Edit** ⇒ **Project Settings** ⇒ **Editor** и задайте свойству **Device** подходящее значение для распознавания Unity Remote.
2. Добавьте на сцену три куба и назовите их **Cube1**, **Cube2** и **Cube3**. Поместите их в позиции с координатами (−3, 1, −5), (0, 1, −5) и (3, 1, −5) соответственно.
3. Создайте новую папку с именем **Scripts**. Создайте новый скрипт **InputScript** в папке **Scripts** и прикрепите его к трем кубам.
4. Добавьте следующий код в метод `Update()` скрипта:

```
foreach (Touch touch in Input.touches)
{
    float xMove = touch.deltaPosition.x * 0.05f;
    float yMove = touch.deltaPosition.y * 0.05f;

    if (touch.fingerId == 0 && gameObject.name == "Cube1")
        transform.Translate(xMove, yMove, 0F);

    if (touch.fingerId == 1 && gameObject.name == "Cube2")
        transform.Translate(xMove, yMove, 0F);

    if (touch.fingerId == 2 && gameObject.name == "Cube3")
        transform.Translate(xMove, yMove, 0F);
}
```

5. Запустите сцену и коснитесь экрана тремя пальцами. Обратите внимание, что все три куба перемещаются независимо друг от друга, а отрыв одного пальца от экрана не влияет на управление другими кубами.

23-Й ЧАС
Доработка и развертка

Что вы узнаете в этом часе

- Как управлять сценами в игре
- Как сохранять данные и объекты, находящиеся на другой сцене
- Различные настройки проигрывателя
- Как развернуть игру на платформе

В этом часе вы узнаете все о доработке и развертке игры. Сначала — как можно перемещаться между сценами. Затем вы изучите способы сохранения данных и игровых объектов между сценами. После этого мы рассмотрим проигрыватель Unity и его настройки. Затем вы узнаете, как создать и развернуть игру.

Управление сценами

Все, что вы делали в Unity ранее, находилось на одной сцене. Так тоже можно создавать большие и сложные игры, но, как правило, гораздо проще использовать несколько сцен. Идея сцены в том, что она представляет собой автономный набор игровых объектов. Таким образом, при переходе между сценами все существующие игровые объекты уничтожаются и создаются новые. Хотя и этого можно избежать, как описано далее.

ПРИМЕЧАНИЕ

Начнем по новой — что такое сцена?

Мы изучили суть сцен еще в начале книги, а сейчас вернемся к этому понятию с новыми знаниями, которыми вы уже владеете. В принципе, сцена — это нечто вроде уровня в игре. Но с играми, которые постепенно становятся сложнее или где уровни генерируются динамически, это не всегда верно. Сцену иногда удобно представлять как набор ассетов. Игра с несколькими уровнями, в которых используются одни и те же объекты, на самом деле может состоять из одной сцены. А вот когда вам нужно избавиться от одной группы объектов и загрузить другую — тогда понадобится новая сцена. Не стоит разделять уровни по различным сценам просто потому, что вы можете. Это имеет смысл только в том случае, если того требует геймплей и управление ассетами.

ПРИМЕЧАНИЕ

Сборка?

Этот час посвящен двум ключевым понятиям: *сборке* и *развертыванию*. Несмотря на то что они часто означают одно и то же, разница все-таки есть. *Собрать* проект подразумевает отдать Unity команду превратить ваш проект в набор исполняемых файлов. Сборка для операционной системы Windows создает файл с расширением.exe с папками данных, а для macOS — файл .d*mg* со всеми данными игры. Развертывание означает отправку исполняемого файла на платформу для запуска. Например, при сборке для Android создается файл.a*pk* (игра), а затем он развертывается на устройстве Android для воспроизведения. Эти параметры задаются в настройках сборки Unity, которые мы рассмотрим в ближайшее время.

Создание последовательности сцен

Реализовать переход между сценами относительно легко. Но нужно немножко настроек. Сперва надо добавить сцены в настройках сборки проекта следующим образом.

1. Откройте параметры сборки, выбрав команду меню **File ⇒ Build Settings**.
2. Перетащите все сцены, которые нужны вам в окончательном проекте, в область **Scenes in Build** диалогового окна **Build Settings** (рис. 23.1).

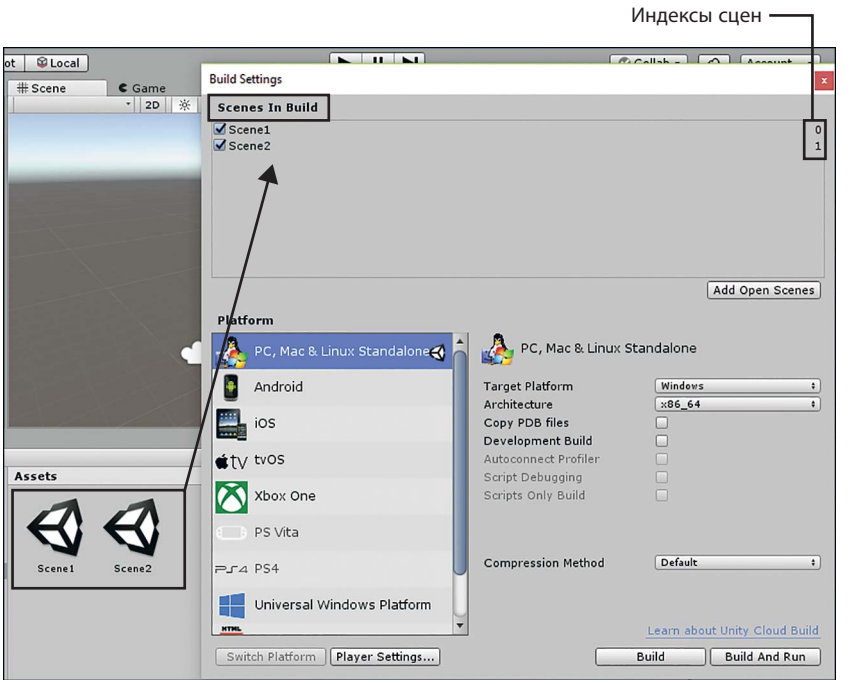


Рис. 23.1. Добавление сцен в настройках сборки

3. Обратите внимание на номер, который появляется рядом с каждой сценой в области **Scenes in Build** диалогового окна **Build Settings**. Эти числа вам понадобятся позже.

ПРАКТИКУМ

Добавление сцен в настройках сборки

В этом упражнении мы добавим сцены в настройках сборки проекта. Обязательно сохраните этот проект, так как он понадобится нам в следующем разделе. Выполните указанные ниже действия.

1. Создайте новый проект и добавьте в него новую папку с именем **Scenes**.
2. Выберите команду меню **File** ⇒ **New Scene**, чтобы создать новую сцену, а затем команду **File** ⇒ **Save Scene As**, чтобы сохранить ее. Сохраните сцену в папке *Scenes* под именем **Scene1**. Повторите этот шаг, чтобы сохранить еще одну сцену под именем **Scene2**.
3. Откройте параметры сборки (выбрав команду **File** ⇒ **Build Settings**). Перетащите сцену **Scene1** в область **Scenes in Build** диалогового окна **Build Settings**, а затем сцену **Scene2** — туда же. Убедитесь, что сцена **Scene1** имеет индекс 0, а **Scene2** — индекс 1. Если это не так, измените их порядок, перетащив сцены.

Переключение между сценами

Теперь, когда порядок сцен установлен, переключаться между ними будет легко. Чтобы изменить сцену, вы можете использовать метод `LoadScene()`, который является частью класса `SceneManager`. Чтобы сообщить Unity, что вам нужен этот класс, добавьте следующую строку в верхней части скрипта:

```
using UnityEngine.SceneManagement;
```

Метод `LoadScene()` принимает один параметр, который является либо целым числом (индекс сцены), либо строкой (имя сцены). Поэтому для загрузки сцены с индексом **1** и именем **Scene2** можно использовать любой из двух способов:

```
SceneManager.LoadScene(1);              // Загрузка по индексу
SceneManager.LoadScene("Scene2");       // Загрузка по имени сцены
```

Этот метод уничтожает все существующие игровые объекты и загружает указанную сцену. Обратите внимание, что эта команда является немедленной и необратимой, поэтому вам нужно быть точно уверенными, что вы хотите сделать именно это. (Префаб **LevelManager** из упражнения в конце часа 14 использует этот метод.)

СОВЕТ

Асинхронная загрузка сцены

Ранее в этой книге вся сцена загружалась сразу. То есть, когда вы с помощью класса `SceneManager` сменяете сцену, текущая сцена выгружается, а затем загружается следующая. Это хорошо для небольших сцен, но, если вы попытаетесь загрузить очень большую, между уровнями появится пауза, во время которой отобразится черный экран. В таких случаях вы можете воспользоваться *асинхронной* загрузкой сцены, когда новая сцена загружается «за кадром», пока основная игра продолжается. Когда новая сцена будет загружена, сцена переключится. Этот метод не дает мгновенных результатов, но помогает избежать задержек в игре. В целом, асинхронная загрузка сцены — сложная штука, и в этой книге мы ее не рассматриваем.

Сохранение данных и объектов

Теперь, когда вы научились переключаться между сценами, вы наверняка заметили, что данные после переключения не сохраняются. Дело в том, что все ваши сцены были полностью независимыми и не имели функции сохранения данных. Однако в более сложных играх сохранение данных (часто называемое *сериализацией*) становится реальной необходимостью. В этом разделе вы узнаете, как сохранять объекты между сценами и как сохранить данные в файл, чтобы можно было обратиться к ним позже.

Сохранение объектов

Простейший способ сохранения данных — это не уничтожать объекты с данными. Например, если у вас есть объект, у которого есть скрипт, содержащий информацию об уровне здоровья, инвентаре, счете и так далее, самый простой способ сохранить его — попросту не уничтожать. Существует несложный способ сделать это, который связан с использованием метода `DontDestroyOnLoad()`. Метод `DontDestroyOnLoad()` принимает один параметр: игровой объект, который нужно сохранить. Например, если ваш игровой объект хранится в переменной с именем `Brick`, вы можете написать следующее:

```
DontDestroyOnLoad (Brick);
```

Поскольку метод принимает игровой объект, объекты могут вызывать его сами на себя, используя ключевое слово `this`. Чтобы объект мог сохранить себя, поместите следующий код в метод `Start()` скрипта этого объекта:

```
DontDestroyOnLoad (this);
```

Теперь при переключении сцены сохраненные объекты никуда не денутся.

ПРАКТИКУМ

Сохранение объектов

В этом упражнении мы переместим куб с одной сцены на другую. Для этого вам понадобится проект, созданный в предыдущем практикуме. Если вы еще не выполнили его, сделайте это сейчас. Не забудьте сохранить проект, он пригодится нам в следующем разделе. Выполните указанные ниже действия.

1. Загрузите проект, созданный в предыдущем практикуме. Загрузите сцену **Scene2** и добавьте туда сферу. Поместите ее в позицию с координатами (0, 2, 0).
2. Загрузите сцену **Scene1**, добавьте куб сцены и поместите его в позицию с координатами (0, 0, 0).
3. Создайте папку **Scripts** и в ней — новый скрипт под названием **DontDestroy**. Прикрепите его к кубу.
4. Измените код в скрипте **DontDestroy** так, чтобы он содержал следующее:

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class DontDestroy: MonoBehaviour
{
    void Start()
    {
        DontDestroyOnLoad(this);
    }

    // Простенький трюк для обнаружения щелчков мышью по объекту
    void OnMouseDown()
    {
        SceneManager.LoadScene(1);
    }
}
```

   Этот код сообщает кубу, что он должен сохраняться между сценами. Кроме того, мы использовали небольшой трюк для обнаружения кликов по кубу со стороны игрока (`OnMouseDown()`, запишите себе название). Когда пользователь щелкает мышью по кубу, загружается сцена **Scene2**.
5. Запустите сцену и обратите внимание, что после щелчка мышью по кубу на панели **Game** сцена меняется, и рядом с кубом появляется сфера (рис. 23.2).

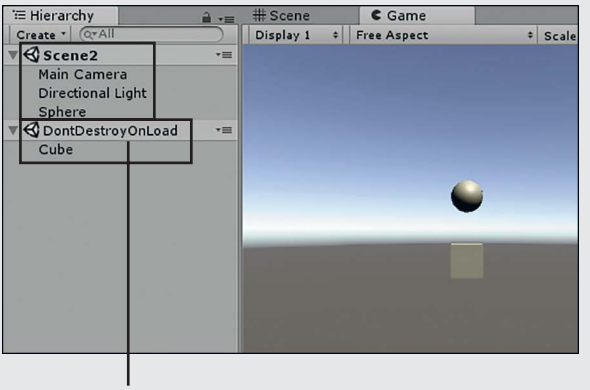


Рис. 23.2. Сохраненный куб на сцене **Scene2**

ВНИМАНИЕ!

Темные сцены

Вы наверняка заметили, что при переключении новая сцена может оказаться темнее, даже если она освещена. Это происходит из-за отсутствия полных данных об освещении для динамически загружаемых сцен.

К счастью, решение несложное: загрузите сцену в Unity и выберите команду **Window** ⇒ **Lighting** ⇒ **Settings**. Сбросьте флажок **Auto Generate**, а затем нажмите кнопку **Generate Lightning**. Так мы инструктируем Unity прекратить использование временных расчетов освещения и выполнять вычисления на ассете. Рядом со сценой появится папка с таким же именем, как и у сцены. После этого с освещением сцены после загрузки будет все в порядке.

Сохранение данных

Иногда вам нужно сохранить данные в файл, чтобы получить доступ к ним позже: например, счет игрока, настройки или инвентарь. Для этого существует большое количество сложных и многофункциональных способов, но есть и простое решение под названием ***PlayerPrefs***. Это объект, который сохраняет данные в файл на вашем компьютере. Разумеется, вы можете использовать его и чтобы извлечь данные обратно.

Сохранить данные в **PlayerPrefs** так же просто, как присвоить им имя. Методы сохранения зависят от типа данных. Например, чтобы сохранить целое число, вызывается метод `SetInt()`, а чтобы извлечь его, вызывается метод `GetInt()`. Таким образом, код, позволяющий сохранить число 10 в **PlayerPrefs** и получить значение обратно, будет выглядеть следующим образом:

```
PlayerPrefs.SetInt ("score", 10);
PlayerPrefs.GetInt ("score");
```

Кроме того, существуют способы сохранять строки (`SetString()`) и числа с плавающей запятой (`SetFloat()`). С помощью этих методов вы можете легко сохранять в файл любые данные.

ПРАКТИКУМ

Использование файла *PlayerPrefs*

В этом упражнении мы попробуем сохранить данные в файл **PlayerPrefs**. Для этого вам понадобится проект, созданный в предыдущем практикуме. Если вы еще не выполнили его, сделайте это. Здесь мы будем использовать устаревший графический интерфейс. Нам сейчас важно поработать с **PlayerPrefs**, а не с интерфейсом. Выполните следующие действия.

1. Откройте проект, созданный в предыдущем практикуме, и убедитесь, что загружена сцена **Scene1**. Создайте в папке **Scripts** новый скрипт под названием **SaveData** и добавьте в него следующий код:

```
public string playerName = "";

void OnGUI()
{
    playerName = GUI.TextField(new Rect(5, 120, 100, 30),
playerName);

    if (GUI.Button(new Rect(5, 180, 50, 50), "Save"))
    {
        PlayerPrefs.SetString("name", playerName);
    }
}
```

2. Прикрепите скрипт к объекту **Main Camera**. Сохраните сцену **Scene1** и загрузите сцену **Scene2**.
3. Создайте новый скрипт под названием **LoadData** и прикрепите его к объекту **Main Camera**. Добавьте в скрипт следующий код:

```
string playerName = "";

void Start()
{
    playerName = PlayerPrefs.GetString("name");
}

void OnGUI()
{
    GUI.Label(new Rect(5, 220, 50, 30), playerName);
}
```

4. Сохраните сцену **Scene2** и перезагрузите сцену **Scene1**. Запустите ее. Введите ваше имя в текстовом поле и нажмите кнопку **Save**. Теперь щелкните мышью

по кубу, чтобы загрузить сцену **Scene2**. (Переключение сцен за счет куба было реализовано в предыдущем практикуме.) Обратите внимание, что введенное имя появляется на экране. Таким образом, мы сохранили данные в файл **PlayerPrefs**, а затем выгрузили их в другой сцене.

ВНИМАНИЕ!

Безопасность данных

Использование файла **PlayerPrefs** для сохранения данных — это очень простой, но не самый безопасный вариант. Данные хранятся в незашифрованном файле на жестком диске. В результате игроки могут легко открыть файл и изменить данные в нем. Это позволяет нечестно получить плюшки или даже сломать игру. Имейте в виду, что метод **PlayerPrefs**, как следует из названия, предназначен для сохранения настроек игрока. Так получилось, что он полезен и для других задач. Надежной защиты данных добиться трудно, и в книге мы не будем говорить об этом. Файл **PlayerPrefs** вполне подойдет вам на раннем этапе разработки игр, но в будущем вам понадобятся более сложные и безопасные средства сохранения данных игрока.

Настройки проигрывателя Unity

В Unity есть несколько параметров, которые влияют на то, как игра будет работать после сборки. Они называются настройками проигрывателя и управляют такими вещами, как значок игры и соотношения сторон экрана. Вообще таких параметров много, и обычно они интуитивно понятны. Если вы выберете команду **Edit** ⇒ **Project Settings** ⇒ **Player**, на панели **Inspector** появится область **Player Settings**. Изучите эти настройки и прочитайте о них в следующих разделах, чтобы узнать, для чего они нужны.

Кросс-платформенные настройки

Первые настройки проигрывателя, которые вы увидите, — кросс-платформенные (рис. 23.3), то есть они применяются к собранной игре независимо от платформы (Windows, iOS, Android, macOS и так далее). Большинство параметров в этом разделе интуитивно понятны. **Product Name** — это название игры. **Default Icon** — значок игры — графический файл с равными сторонами и кратным размером, например 8 × 8, 16 × 16, 32 × 32, 64 × 64 и так далее. Если файл не соответствует этому формату, масштабирование исказит изображение значка. Можно также задать пользовательский указатель мыши (**Default Icon**) и определить, где находится позиция щелчка мыши (**Cursor Hotspot**).

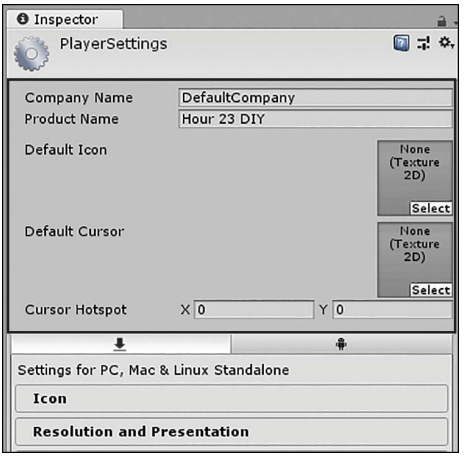


Рис. 23.3. Кросс-платформенные настройки

Специфические для платформы настройки

Эти настройки специфичны для каждой отдельной платформы. Некоторые параметры в данном разделе повторяются, но вам придется настроить каждый из них для каждой выбранной платформы, прежде чем собрать игру. Вы можете выбрать определенную платформу, щелкнув мышью по ее значку на панели (рис. 23.4). Обратите внимание, что вы видите значки только тех платформ, которые выбрали в процессе установки среды Unity. Как видно на рис. 23.4, на компьютере установлены только платформы **Standalone (PC, Mac и Linux)** и **Android**.

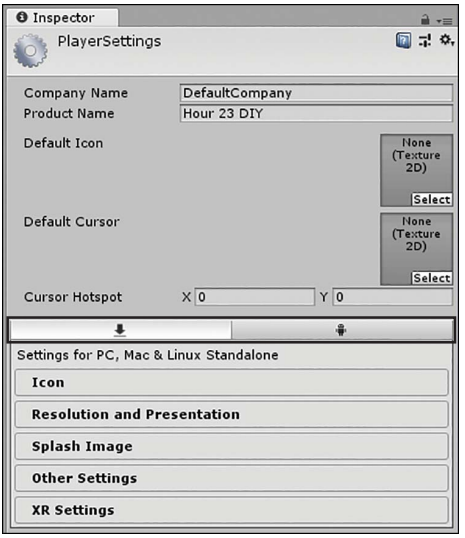


Рис. 23.4. Выбор платформы

Многие параметры требуют глубокого знания специфики платформы, с которой вы работаете. Не стоит изменять эти настройки, пока вы как следует не разберетесь, как функционирует конкретная платформа. Другие настройки достаточно просты и изменяются только в определенных случаях. Например, **Resolution** (Разрешение) и **Presentation** (Представление) касаются размеров окна игры. Для компьютеров может использоваться оконный или полноэкранный режим, а также большой набор различных поддерживаемых соотношений сторон. Включая или отключая эти соотношения, вы тем самым разрешаете или запрещаете варианты экранного разрешения, которые игрок может выбрать во время игры.

Настройки значка подбираются автоматически, если вы задали его изображение в свойстве **Default Icon** в разделе кросс-платформенных настроек. Вы увидите, что сгенерируются различные размеры этого значка. Вот почему важно, чтобы он имел корректные размеры. Вы также можете задать заставку для игры в разделе **Splash Image**. Заставка — это изображение, которое появляется при первом запуске игры.

ПРИМЕЧАНИЕ

Так много настроек...

Вы, наверное, заметили, что многие настройки области **Player Settings** на панели **Inspector** в данном разделе не рассматриваются. Большинство параметров уже установлено по умолчанию, так что вы можете собрать игру с дефолтными настройками. Другие параметры используются для декорирования или добавления функций. Не стоит трогать настройки, если вы не понимаете, что они делают, так как изменения могут привести к проблемам при запуске игры. Проще говоря, лучше настраивать только самые необходимые параметры, пока вы не разобрались со всеми остальными.

Сборка игры

Представьте, что вы создали свою первую игру. Вы выполнили всю работу и сделали в редакторе все, что собирались. Вы даже сконфигурировали проигрыватель и настроили все так, как вы хотите. Ну что ж… Настало время собрать игру. Пора вам узнать о двух окнах настроек, используемых в этом процессе. Первое из них — **Build Settings**. В нем вы задаете конечные результаты процесса сборки. Второе окно — **Game Settings**, где перечислены параметры, которые сможет просматривать и изменять игрок.

Настройки сборки

Диалоговое окно **Build Settings** содержит параметры сборки игры. Именно здесь можно задать платформу, для которой игра будет собрана, а также перечислить сцены игры. Вы уже видели это диалоговое окно, но теперь остановимся на нем подробнее.

Чтобы открыть его, выберите команду **File** ⇒ **Build Settings**. Теперь вы можете изменить и настроить игру так, как хотите. На рис. 23.5 показано диалоговое окно **Build Settings** и различные его элементы.

В этом окне в разделе **Platform** вы можете задать платформу для сборки. Если вы выбираете новую платформу, вам нужно нажать кнопку **Switch Platform**, чтобы переключиться на нее.

При нажатии кнопки **Player Settings** на панели **Inspector** появится область **Player Settings**.

Раздел **Scenes in Build** вы уже видели ранее, в нем вы задаете, какие сцены и в каком порядке будут присутствовать в игре. Вам также доступны различные параметры сборки для конкретной платформы, которую вы выбрали. Настройки для устройств под управлением операционной системы macOS и Linux в принципе интуитивно понятны. Единственное, стоит отметить свойство **Development Build**, которое позволяет запустить игру с отладчиком и профайлером производительности.

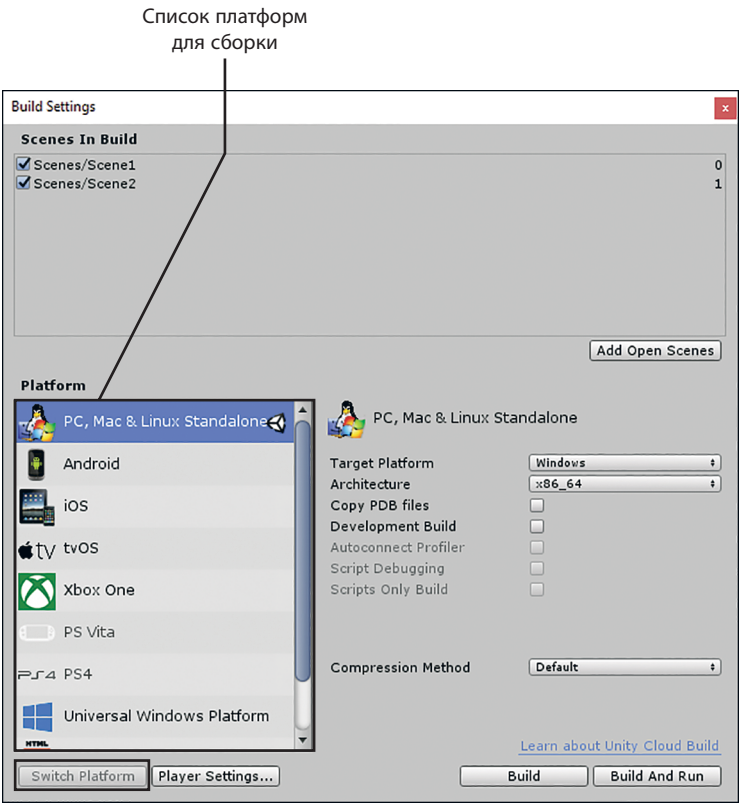


Рис. 23.5. Диалоговое окно **Build Settings**

Когда вы будете готовы, нажмите кнопку **Build**, чтобы только собрать игру, или кнопку **Build and Run**, чтобы собрать и сразу запустить игру. Формат файла, который создаст Unity, зависит от выбранной платформы.

Настройки игры

Когда собранная игра запускается из исполняемого файла (не из Unity), перед игроком появляется диалоговое окно **Game Settings** (рис. 23.6). Здесь игроки настраивают параметры игры.

Первое, что вы можете заметить, — это название игры, которое отображается в строке заголовка окна. Кроме того, заставка, которую вы задали в области **Player Settings** на панели **Inspector**, отображается в верхней части диалогового окна. Первая вкладка, **Graphics**, используется для определения разрешения, в котором игроки хотят запустить игру. Список доступных разрешений определяется соотношением сторон, которые вы разрешили/запретили в области **Player Settings** на панели **Inspector**. Игроки также могут выбрать запуск в оконном или полноэкранном режиме и настройки качества графики.

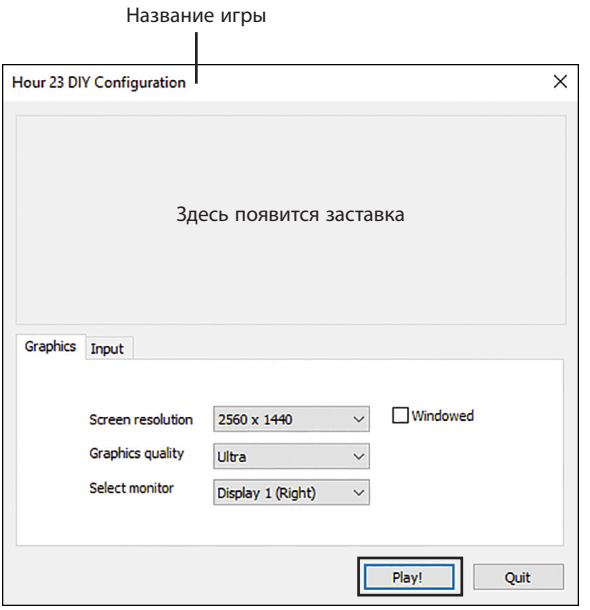


Рис. 23.6. Диалоговое окно **Game Settings**

Далее можно переключиться на вкладку **Input** (рис. 23.7) и задать входные оси на определенные клавиши.

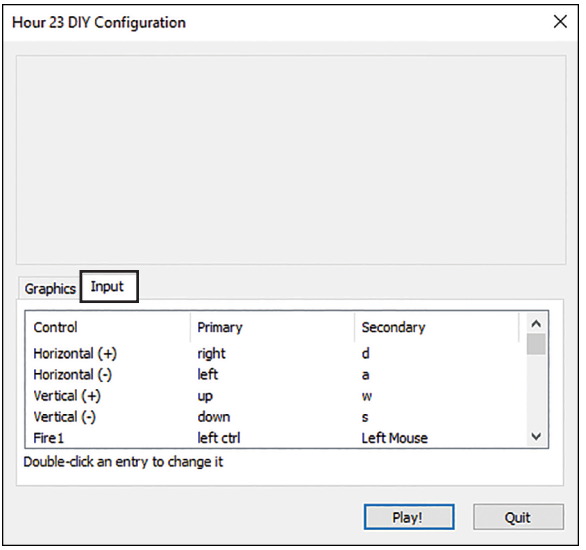


Рис. 23.7. Настройки ввода

ПРИМЕЧАНИЕ

Ну я же говорил!

Вы, возможно, помните, что ранее в этой книге было сказано, что всегда следует стремиться к тому, чтобы вводы от игрока привязывались к осям, а не конкретным клавишам. И вот причина. Если вы применяете определенные клавиши вместо осей, игрок будет вынужден использовать схему управления, которую вы задали. Если вы считаете, что это не имеет большого значения, вспомните, что многие люди (люди с ограниченными возможностями, например) используют нестандартные устройства ввода. Если вы не позволите им переназначать управление, играть у них не получится. Использование осей вместо конкретных клавиш требует не столь уж много работы с вашей стороны, но может сильно повлиять на отношение аудитории к вашей игре.

Когда игроки выберут параметры, которые они хотят, они нажимают кнопку **Play!** и наслаждаются игрой.

Резюме

В этом часе вы узнали все о сборке игры в Unity. Сначала вы научились переключать сцены в Unity, используя метод `LoadScene()` класса `SceneManager`. Затем вы узнали, как сохранять игровые объекты и данные. Наконец, вы прошлись по настройкам проигрывателя. В конце часа вы научились собирать свои игры.

Вопросы и ответы

Вопрос: Многие параметры казались важными. Почему мы их не рассмотрели?

Ответ: По правде говоря, большинство этих параметров вам и не нужны. Дело в том, что они не важны… пока. Большинство настроек весьма специфичны для платформы, и здесь мы их не рассматриваем. Вместо того чтобы писать талмуд о настройках, которыми вы, может быть, и не воспользуетесь никогда, мы оставляем эти вопросы на ваше усмотрение (изучение).

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Как присвоить индексы сценам в игре?
2. Верно или нет: данные можно сохранять с помощью объекта **PlayerPrefs**.
3. Какие размеры может иметь значок игры?
4. Верно или нет: настройки входов в настройках игры позволяют игроку переназначить управление.

Ответы

1. После добавления сцены в список сцен для сборки ей присваивается индекс.
2. Верно.
3. Значок должен быть квадратом со сторонами с размером, кратным 2: 8×8, 16×16, 32×32 и так далее.
4. Нет. Игрок может переназначить входы, которые привязываются к осям, а не конкретным клавишам.

Упражнение

В этом упражнении вы соберете игру для операционной системы вашего компьютера и поэкспериментируете с различными функциями. Само упражнение не такое уж большое, но вы можете попробовать различные параметры и понаблюдать за результатом. Поскольку это вариант пробной сборки, готового проекта в файлах примеров для книги на этот раз нет.

1. Откройте любой созданный вами ранее проект или создайте новый.
2. Перейдите к области **Player Settings** на панели **Inspector** и настройте проигрыватель так, как вам нравится.
3. Перейдите к диалоговому окну **Build Settings** и убедитесь, что все нужные сцены добавлены в список для сборки.
4. Убедитесь в том, что выбрана платформа **PC, Mac & Linux Standalone**.
5. Соберите игру, нажав кнопку **Build**.
6. Найдите файл игры, которую вы только что собрали, и запустите его. Поэкспериментируйте с различными настройками и посмотрите, как они влияют на игровой процесс.

24-Й ЧАС
Финальные штрихи

Что вы узнаете в этом часе

- Что мы успели сделать
- Что делать дальше
- Какие есть полезные ресурсы

В этом часе мы подведем черту под нашим вводным путешествием в Unity. Сначала мы вспомним, чему вы научились. Затем посмотрим, что можно делать дальше и как продолжать совершенствовать свои навыки. В конце вы узнаете, какие ресурсы помогут вам продолжить обучение.

Наши достижения

Когда вы долго работаете над чем-либо, часть действий забывается. Поэтому полезно проанализировать себя до обучения и после и сравнить одно и другое. Такой анализ прогресса мотивирует и дает чувство удовлетворенности, так что давайте посмотрим на некоторые числа.

Девятнадцать часов обучения

В первую очередь, вы потратили 19 часов (или больше), активно изучая различные элементы разработки игр на движке Unity. Вот в каких вопросах вы теперь разбираетесь.

- Как использовать редактор Unity и многие его панели и диалоги.
- Игровые объекты, их преобразования. Вы узнали о двумерной и трехмерной, о локальной и глобальной системах координат. Вы стали профессионалами в использовании встроенных геометрических примитивов Unity.
- Модели. В частности, вы узнали, что модели состоят из текстур и шейдеров, применяемых к материалам, которые, в свою очередь, применяются

к мешам. Вы узнали, что меши состоят из треугольников, которые состоят из множества точек в 3D-пространстве.

- Как создать ландшафт в Unity. Вы собирали уникальные ландшафты и освоили инструменты, необходимые для создания любого мира, который только можно вообразить (многие ли так умеют?). Вы улучшили ваши миры с помощью эффектов и элементов среды.
- Все о камерах и источниках света.
- Программирование в Unity. Если вы никогда не программировали ранее, то это очень круто. Хорошая работа!
- О столкновениях, физических материалах и рейкастинге. Иными словами, вы изучили основы взаимодействия объектов через физику.
- Префабы, экземпляры.
- Как создать пользовательский интерфейс (UI), используя мощные средства управления в Unity.
- Как управлять игроками с помощью контроллеров Unity. Кроме того, вы создали пользовательский контроллер 2D-персонажа для своих проектов.
- Как создавать 2D-миры с 2D-тайлмапами.
- Как создавать удивительные эффекты частиц с использованием различных систем частиц. Вы также подробно изучили каждый модуль систем частиц.
- Как работать с новой системой анимации Unity Mecanim. В процессе вы узнали, как переназначить модели скелет, чтобы использовать анимации, которые создавались для другой модели.
- Вы также научились редактировать анимации, чтобы создавать собственные анимационные клипы.
- Вы научились создавать прямо-таки целые фильмы с помощью таймлайнов.
- Как управлять звуком в ваших проектах. Вы узнали, как работать с 2D- и 3D-звуками, как зацикливать и переключать аудиоклипы.
- Как разрабатывать игры для мобильных устройств. Вы узнали, как тестировать игры с помощью приложения Unity Remote и использовать акселерометр устройства и сенсорный экран.
- Как отшлифовать игру с использованием нескольких сцен и сохранения данных. Вы узнали, как можно собрать и развернуть игру.

Список крутой, но это далеко не все, что вы узнали в книге. Надеюсь, прочитав его, вы вспомнили каждый из пунктов!

Четыре полноценные игры

На протяжении этой книги вы создали четыре игры: «Великолепный гонщик», «Шар хаоса», «Капитан Бластер» и «Бег с препятствиями». Вы проработали концепцию каждой из них, определили правила и требования. Затем вы создали все объекты для каждой игры. Вы разместили каждый объект, игрока, мир, мяч, метеор, и все такое прочее. Вы написали все скрипты и внесли интерактивность. Затем, главное, вы проверили все игры, определили их сильные и слабые стороны.

Вы поиграли в ваши игры, и, наверное, дали поиграть товарищам. Вы подумали, что можно улучшить, а может, даже применили эти улучшения. Давайте вспомним игровые механики и концепции, которые вы использовали.

- «Великолепный гонщик». Это 3D-гонка с таймером и контроллером от первого лица. В игре был полностью созданный и текстурированный ландшафт, содержащий водные преграды, триггеры и огни.
- «Шар хаоса». Эта 3D-игра отличается большим количеством столкновений и физического взаимодействия. Вы использовали физические материалы, чтобы создать упругую арену, а также создали мишени по углам, которые взаимодействовали с объектами.
- «Капитан Бластер». Это космический 2D-шутер в ретростиле, где мы использовали прокрутку фона и 2D-эффекты. Здесь впервые ваш игрок мог проиграть. Вы использовали готовые модели и текстуры, чтобы игра выглядела круче.
- «Бег с препятствиями». Эта 3D-игра нацелена на сбор бонусов и уворачивание от препятствий. Мы использовали анимации Mecanim и сторонние модели, а также хитроумные манипуляции с координатами, чтобы создать эффект прокрутки в 3D.

Вы приобрели опыт в разработке игр, сборке, тестировании и подгонке игр под новое оборудование. Неплохо. Совсем неплохо.

Более пятидесяти сцен

Изучая книгу, вы создали более пятидесяти сцен. Но это не главное. Важнее то, что вы проработали по крайней мере пятьдесят различных интересных концепций. Это довольно много и вполне достаточно для старта.

Вы, вероятно, уже поняли, зачем нужен этот раздел. Вы проделали много работы, и вы должны гордиться этим. Вы своими руками использовали огромную часть игрового движка Unity. Это знание будет вам опорой и дальше на вашем пути.

Куда двигаться дальше

Даже если вы от корки до корки прочитали эту книгу, назвать ваше обучение законченным нельзя. Более того, в нашей отрасли это, похоже, вообще невозможно, ведь все так быстро меняется. Тем не менее вот несколько советов, куда вы можете двинуться дальше.

Делайте игры

Нет, серьезно — делайте игры. Это самое крутое и полезное занятие. Если вы хотите узнать больше о Unity, или даже ищете работу разработчика, или у вас уже есть работа, но вы хотите совершенствоваться, — делайте игры. Распространенное заблуждение среди новичков в игровой индустрии (или индустрии программного обеспечения в целом), что одного знания достаточно, чтобы найти работу или улучшить свои навыки. Но истина прямо противоположна. Опыт — вот что главное. Делайте игры. Пусть для начала они будут небольшими, вроде тех, которые вы создали в этой книге. Если вы попытаетесь выполнить сразу слишком большую задачу, это может привести к разочарованию и неудовлетворенности. Творите что хотите — главное, делайте игры.

Работайте с людьми

Существует много сообществ, которые разрабатывают игры на продажу или для удовольствия. Присоединяйся к ним. Ваш опыт работы в Unity будет полезен для них. Помните, что у вас уже есть четыре игры в портфолио. Работа с коллегами научит вас групповой динамике. Кроме того, она позволяет выйти на качественно новый уровень сложности игр. Попробуйте найти художника и звукорежиссера, чтобы ваши игры были богаты с точки зрения красочности и звука. Вы увидите, что работа в команде — это лучший способ узнать больше о ваших сильных и слабых сторонах. Это придаст вам уверенности и обеспечит лучшее представление о реальности.

Пишите об этом

Пишите о ваших играх и вообще о ваших достижениях в области разработки — это будет полезно для личного прогресса. Если вы заведете блог или хотя бы блокнот, ваши наблюдения пригодятся вам в будущем. Кроме того, это отличный способ оттачивания своих навыков, в том числе в сотрудничестве с коллегами. Излагайте свои идеи, чтобы получить обратную связь и узнать мнение других.

Доступные ресурсы

Существует множество ресурсов, с помощью которых можно продолжить обучение работе с игровым движком Unity и разработке игр в целом. В первую очередь это документация Unity, а именно — официальный ресурс по ссылке **docs.unity3d.com**. Важно знать, что документ требует технического подхода к Unity. Это скорее справочник, а не средство обучения.

Unity также предоставляет большой ассортимент онлайн-курсов на сайте Learn по адресу **unity3d.com**. Там вы найдете много видеороликов, проектов и других ресурсов, которые помогут вам улучшить навыки.

Если у вас появятся вопросы, на которые вы не сможете ответить с помощью этих двух ресурсов, попробуйте обратиться к сообществу Unity. На сайте Unity Answers по адресу **answers.unity3d.com** вы можете задать конкретные вопросы и получить ответы профессионалов в сфере разработки с помощью Unity.

Помимо официальных ресурсов Unity, есть несколько сайтов по разработке игр. Два наиболее популярных сайта: **www.gamasutra.com** и **www.gamedev.net**. На обоих обитает большое сообщество, постоянно публикуются статьи. Их тематика не ограничивается Unity, так что информация будет разносторонней и непредвзятой.

Резюме

В этом часе мы вспомнили все, что вы делали в Unity, изучая эту книгу. Мы также рассмотрели перспективы. В начале мы вспомнили каждую главу книги. Затем мы подумали, что можно сделать, чтобы продолжать совершенствовать свои навыки. Наконец, вы рассмотрели некоторые доступные для вас ресурсы по разработке игр.

Вопросы и ответы

Вопрос: **После прочтения этого часа у меня сложилось впечатление, что вы рекомендуете мне делать игры. Это так?**

Ответ: Да. Кажется, я уже говорил это несколько раз. Я не могу не подчеркнуть, насколько важно продолжать оттачивать свои навыки на практике и в творчестве.

Семинар

Уделите некоторое время проработке приведенных ниже вопросов, чтобы убедиться, что вы хорошо усвоили материал.

Контрольные вопросы

1. Может ли Unity использоваться для создания 2D и 3D-игр?
2. Можно ли гордиться результатами, которых мы достигли до сих пор?
3. Как лучше всего поступить, чтобы точно успешно наращивать свои навыки в разработке игр?
4. Мы узнали о Unity все?

Ответы

1. Несомненно!
2. Опять же, несомненно!
3. Нужно делать игры и делиться ими с людьми.
4. Нет, учеба никогда не прекращается!

Упражнение

В этом часе мы оглянулись назад и вспомнили все, что вы узнали. Последнее упражнение книги продолжает тему. Обычно в игровой индустрии пишут что-то вроде последней статьи о созданной игре, чтобы другие почитали о ней. В этой статье вы анализируете вещи, с которыми справились в процессе разработки, а также те, которые вы не сделали. Вы рассказываете другим о подводных камнях, на которые вы наткнулись, чтобы коллеги не повторяли ваших ошибок.

В качестве задания для этого упражнения напишите такую статью об одной из игр, которые вы сделали. Не обязательно показывать ее кому-нибудь, здесь важнее сам процесс. Подойдите к написанию ответственно, чтобы не было стыдно перечитывать. Через некоторое время вы будете читать ее и удивляться, что казалось вам трудным, а что приятным.

Написав статью, распечатайте ее и вложите в эту книгу. Позже, наткнувшись на книгу снова, не забудьте открыть статью и прочитать ее.

Предметный указатель

Научно-популярное издание

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР. ГЕЙМДИЗАЙН

Майк Гейг

РАЗРАБОТКА ИГР НА UNITY 2018 ЗА 24 ЧАСА

Главный редактор *Р. Фасхутдинов*
Руководитель направления *В. Обручев*
Ответственный редактор *Е. Горанская*
Научный редактор *С. Клименко*
Литературный редактор *С. Позднякова*
Младший редактор *Д. Атакишиева*
Художественный редактор *Е. Пуговкина*
Компьютерная верстка *Э. Брегис*
Корректор *Р. Болдинова*

ООО «Издательство «Эксмо»
123308, Москва, ул. Зорге, д. 1. Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru
Өндіруші: «ЭКСМО» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.
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-дукен : 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

Подписано в печать 11.12.2019. Формат 70x100$^{1}/_{16}$.
Печать офсетная. Усл. печ. л. 37,59.
Тираж экз. Заказ

12+

ISBN 978-5-04-105963-7

9 785041 059637 >

Москва. ООО «Торговый Дом «Эксмо»
Адрес: 123308, г. Москва, ул. Зорге, д.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**

КОГДА ВЫ ДАРИТЕ КНИГУ, ВЫ ДАРИТЕ ЦЕЛЫЙ МИР

ХОТИТЕ ЗНАТЬ БОЛЬШЕ?

Заходите на сайт:
https://eksmo.ru/b2b/

Звоните по телефону:
+7 495 411-68-59, доб. 2261

ВАШ ЛОГОТИП НА ОБЛОЖКЕ

ВАШ ЛОГОТИП НА КОРЕШКЕ

ОБРАЩЕНИЕ К КЛИЕНТАМ НА ОБЛОЖКЕ

Разработка игр на Unity® 2018

за 24 Часа

Всего за 24 урока, каждый продолжительностью 1 час или меньше, вы освоите игровой движок Unity 2018, на котором были созданы такие замечательные игры, как Ori and the Blind Forest, Firewatch, Monument Valley и многие другие известные хиты!

Пошаговая структура этой книги познакомит вас со всеми особенностями работы с Unity. Вы начнете с самых основ и в процессе обучения освоите такие замысловатые темы, как внутриигровая физика, анимация и развертывание игр на мобильные устройства. Каждый новый урок углубляет и закрепляет уже полученные вами знания, создавая прочную основу для успешной работы с реальными задачами.

Пошаговые инструкции подготовят к наиболее часто встречающимся задачам разработки на Unity.

Практические, реальные примеры продемонстрируют, как применять навыки.

Контрольные вопросы и упражнения помогут протестировать полученные знания и расширить навыки.

Примечания и советы подскажут комбинации клавиш и решения.

Майк Гейг – евангелист Unity Technologies, где он руководит информационно-образовательным направлением. Гейг имеет опыт независимого игрового разработчика, университетского преподавателя и даже успел отметиться в качестве свадебного диджея. Будучи страстным игроком, Майк вкладывает все силы в то, чтобы сделать разработку игр увлекательной и доступной для всех, вне зависимости от их исходных знаний и навыков. И кстати... Привет, мам!

- Быстро начать работать с игровым движком и редактором Unity 2018
- Эффективно использовать графические ресурсы Unity
- Понимать, как наиболее удачно настроить источники света и камеры
- Проектировать потрясающие миры с уникальным ландшафтом и средой при помощи инструментов Unity
- Писать скрипты, начиная с самых основ и переходя к более сложным задачам
- Создавать и редактировать префабы для работы с одинаковыми игровыми объектами
- Разрабатывать интуитивно понятные и простые игровые интерфейсы
- Взаимодействовать со встроенными и настраиваемыми контроллерами персонажей
- Создавать реалистичные физические и триггерные столкновения
- Использовать полную мощь инструментов работы с анимацией и новой системы Timeline
- Создавать звуковые эффекты окружающей среды
- Разрабатывать игры для мобильных устройств, используя встроенные в них акселерометры и сенсорные дисплеи
- Создавать захватывающие 2D-игры с помощью 2D-инструментов Unity и тайлмапов
- Наносить финальные штрихи и подготавливать игру к развертыванию

Идеальная книга для тех, кто хочет быстро начать работать в Unity! Коротко и ясно рассказывается о самых необходимых системах движка, а полученные знания сразу же применяются на практических примерах.

ВАЛЕНТИН СИМОНОВ,
Solutions Engineer, Unity

БОМБОРА

Бомбора — это новое название Эксмо Non-fiction, лидера 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
www.bombora.ru

Для доступа к сопроводительным файлам перейдите по адресу:
http://addons.eksmo.ru/it/Unity.zip

ISBN 978-5-04-105963-7