Алан Торн

Основы анимации в Unity

Alan Thorn

Unity Animation Essentials

Bring your characters to life with the latest features of Unity and Mecanim

[PACKT] PUBLISHING open source community experience distilled

Алан Торн

Основы анимации в Unity

Оживите свои персонажи с помощью последних достижений Unity и системы Mecanim

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО
Москва, 2016

УДК 004.4'2Unity3D
ББК 32.972
Т60

Торн А.
Т60 Основы анимации в Unity / пер. с англ. Р. Рагимова. – М.: ДМК Пресс, 2016. – 176 с.: ил.

ISBN 978-5-97060-377-2

Unity является самодостаточным, интегрированным движком для разработки игр, который предоставляет готовые к применению функции для создания интерактивного 3D-контента. Это игровой движок, соединенный с многофункциональной и сложной системой анимации Mecanim. Данная книга содержит описание мощных инструментов анимации и способов их применения в Unity. Сначала рассматриваются основные идеи, а затем примеры их практического использования для создания анимации в режиме реального времени в играх. В издании собрано множество полезных советов для создания анимации профессионального качества, а также для разработки быстрых интерактивных сцен. Каждая глава посвящена одной из областей анимации, от установки и событий до анимации персонажей и систем частиц.

Прочтя эту книгу, вы сможете в полной мере использовать возможности системы Mecanim и Unity.

УДК 004.4'2Unity3D
ББК 32.972

ISBN 978-1-78217-481-3 (анг.)
ISBN 978-5-97060-377-2 (рус.)

Содержание

Об авторе

Алан Торн (Alan Thorn) – независимый разработчик игр и писатель, более 13 лет занимающийся промышленной разработкой игр. Он основатель компании Wax Lyrical Games и создатель игры *Baron Wittard: Nemesis of Ragnarok*, отмеченной многочисленными призами. Он также автор более десятка видеокурсов и пятнадцати книг о разработке игр, среди которых – *Mastering Unity Scripting*, *How to Cheat in Unity 5* и *UDK Game Development*. Кроме того, он является приглашенным лектором магистерского курса дизайна и разработки игр в Национальной школе кино и телевидения в Лондоне.

Алан участвовал как независимый разработчик в более чем 500 проектах по созданию игр, симуляторов, игровых киосков, сложных игр, программ расширения реальности, программного обеспечения для игровых студий, музеев и тематических парков по всему миру. В настоящее время он работает в двух игровых проектах. Он также увлекается графикой, философией, йогой и пешими походами. Больше информации о нем можно найти на сайте http://www.alanthorn.net.

О технических рецензентах

Адам Сингл (Adam Single) – муж, отец, профессиональный разработчик, инди-разработчик, любитель музыки и геймер. Он программист игры **7bit Hero**; программист в команде Real Serious Games в Брисбене, Австралия; соучредитель, программист и содизайнер Sly Budgie; соорганизатор компании Game Technology Brisbane meetup.

В области профессиональной разработки игр он с 2011 года, он работал над многими мобильными играми, в том числе над хитом для андроида *Photon* и предустановленными играми для мобильных телефонов Disney Japan's. Адам был программистом в команде, которая создала огромный интерактивный дисплей в технологическом университете Квинсленда (Queensland University of Technology's), поразительную мультитач-инсталляцию, которая была названа The Cube. Это было сделано в рамках Australia's first Digital Writing Residency. Он также работал в команде Real Serious Games над созданием масштабных интерактивных демонстраций для горнодобывающей и строительной промышленности, некоторые из которых являются виртуальными реальностями и основаны на Oculus Rift. Все это было сделано с помощью игрового движка Unity.

У Адама страсть к уникальным возможностям современных технологий. Когда он не работает над новой перспективной игровой механикой для компании Sly Budgie, он экспериментирует с домашней виртуальной реальностью с использованием технологии мобильного телефона, создает мобильные приложения, применяя самые современные методы веб-разработки и выдвигая захватывающие идеи.

Брайан Р. Вензен (Bryan R. Venzen) – художник визуальных эффектов и аниматор из Тампы, штат Флорида. После окончания Florida Interactive Entertainment Academy (FIEA) в 2011 году он увлекся спецэффектами и анимацией для мобильный телефонов, ПК и игровых приставок. Благодаря своим способностям и опыту он вносит свой вклад во многие проекты, создавая спецэффекты, включая спецэффекты для производства, работающие на физическом движке NVIDIA (PhysX), анимацию, и выполняя моделирование в пакетах Autodesk 3ds Max и Maya, разрабатывая уровни на Unity и UDK, активно участвуя в обсуждениях дизайна, чтобы улучшить общее качество продукции. Брайан быстро обучается и целеустремленно работает в команде, владея многими 3D-пакетами, движками и инструментами для игр. Он является специалистом в вопросах поиска и устранения неисправностей и хорошо себя чувствует в быстро развивающейся рабочей среде. Создание развлекательного контента – его страсть.

Предисловие

Ничто не сравнится с ней! Анимация играет в играх главную роль практически везде, от простого применения, такого как, например, полет космического корабля, до сложного, как мимика лица. Вы просто не сможете разработать полноценную игру и по-настоящему интерактивное приложение без хоть какого-то использования анимации. В этой книге мы шаг за шагом обойдем всю обширную область, предлагаемую Unity, для создания анимации в реальном времени. Мы начнем свой путь с самого простого. Эта книга не предполагает предварительного наличия каких-либо знаний об анимации или опыта применения анимации в Unity. Однако книга предполагает хотя бы некоторое знакомство с разработкой в Unity, умение пользоваться ее основным интерфейсом и хотя бы начальные навыки написания кода на языке программирования C#. Используя все это в качестве отправной точки, вы с помощью данной книги научитесь создавать множество удивительных анимаций и применять их для отображения реального мира, сможете достичь ощутимых результатов быстро и легко. А сейчас давайте посмотрим, что ждет нас впереди...

Что охватывает эта книга

Глава 1 «*Основы анимации*» начинается с формального описания анимации как понятия, раскрывается сущность анимации. Затем в этой главе рассматриваются применяемые в Unity типы анимаций и написание связанных с ними скриптов. Сюда включены описания работы с deltaTime, анимационными кривыми и даже рельефная анимация мешей. В общем, эта глава начинает наше путешествие по анимации и охватывает все, что нам необходимо знать для продвижения вперед.

Глава 2 «*Анимация спрайтами*» исследует анимацию в мире 2D, свойства спрайтов, используемых для создания 2D-игр, текстуры книг с двигающимися картинками и плоскую анимацию. Рассматриваются редактор спрайтов Unity, кадры анимации, частота кадров и исправление общих проблем анимации.

Глава 3 «*Встроенная анимация*» позволит вам ознакомиться с анимацией и для 2D, и для 3D в Unity, научит вас использованию окна Animation и системы частиц. Мы рассмотрим два реальных и полезных проекта. Во-первых, мы заставим камеру облететь уровень. Во-вторых, мы создадим систему частиц светящейся пыли, которая нередко встречается в фантастических играх.

Глава 4 «*Анимация предметов с помощью системы Mecanim*» научит вас обращению с флагманским инструментом Unity, носящим имя Mecanim. Mecanim включает в себя набор функций, совместное использование которых делает возможным создание современной плавной анимации, особенно для персонажей. В этой главе мы рассмотрим несколько нетрадиционные примеры использования системы Mecanim для оживления дверей и других интерактивных реквизитов уровней.

Глава 5 «*Анимация персонажей: основы*» начинается с анализа оснащенной анимации персонажей. В этой главе мы рассмотрим, как анимировать гуманоидный скелет и персонаж в реальном времени. Мы также рассмотрим, как импортировать готовые персонажи и оптимально настроить их для анимации.

Глава 6 «*Продвинутая анимация персонажей*» логически вытекает из предыдущей главы. Итак, возьмем импортированный и настроенный в Unity персонаж и анимируем его как управляемый игроком. Этот персонаж будет поддерживать анимацию ожидания, анимацию ходьбы и бега, а также анимацию поворотов влево и вправо.

Глава 7 «*Смешивание форм, инверсные кинематики и анимированные текстуры*» последняя в этой книге. В этой главе рассматриваются следующие темы: смешивание форм для анимации лица и других телодвижений, обратная кинематика для точного позиционирования рук и ног персонажа в режиме реального времени, анимированные текстуры для воспроизведения видеофайлов в виде проецирования на геометрических поверхностях.

Что вам необходимо при чтении этой книги

Эта книга содержит почти все, что вам нужно при работе с ней. Каждая глава рассматривает практические, реальные проекты анимации и включает сопутствующие файлы, которые можно загрузить и использовать. Одно только вам нужно, не считая этой книги и вашего внимания к ней, – это копия самой последней версии Unity; на время написания данной книги это Unity 5. Приложение Unity 5 бесплатно для личного использования и может быть загружено с сайта Unity https://unity3d.com/. В дополнение к Unity, если вы захотите создать движущиеся модели персонажей и прочие анимированные 3D-активы, вам понадобится программное обеспечение для 3D-моделирования и анимации, такое как 3ds Max, Maya или Blender. Blender можно бес-

платно использовать, загрузив его с сайта http://www.blender.org/. Однако в этой книге описана только анимация в Unity.

Для кого эта книга

Если вы имеете базовые знания о Unity и собираетесь их расширить, хотите узнать больше об анимации в реальном времени, тогда эта книга для вас. Автор книги предполагает, что вы уже разрабатываете простые приложения, пишите несложные скрипты и желаете научиться чему-то большему, например понимать наиболее продвинутые концепции анимации. Особенно вы, возможно, хотите узнать, как инструменты, такие как Mecanim, помогут вам не только создать невероятную анимацию, но и сделать вашу работу проще и эффективнее.

Соглашения

В этой книге использовано несколько стилей текста для выделения разных видов информации. Здесь приведены примеры этих стилей с объяснением их назначения.

Тексты программ, имена таблиц баз данных, имена папок, файлов, расширения файлов, параметры, просто URL, ввод пользователя и заголовки в Twitter будут выглядеть так: «По умолчанию все новые скрипты создаются с двумя функциями `Start` и `Update`».

Блок программного кода будет приведен в следующем виде:

```
using UnityEngine;
using System.Collections;
public class MoviePlay : MonoBehaviour
{
  //Reference to movie to play
  public MovieTexture Movie = null;
  // Use this for initialization
  void Start ()
  {
    //Get Mesh Renderer Component
    MeshRenderer MeshR = GetComponent<MeshRenderer>();
    //Assign movie texture
    MeshR.material.mainTexture = Movie;
    GetComponent<AudioSource>().clip = Movie.audioClip;
    GetComponent<AudioSource>().spatialBlend=0;
    Movie.Play();
    GetComponent<AudioSource>().Play();
  }
}
```

Когда мы захотим обратить ваше внимание на отдельные части программного кода, соответствующие строки или слова будут выделены полужирным шрифтом:

```
using UnityEngine;
using System.Collections;
public class Mover : MonoBehaviour
{
  // Use this for initialization
  void Start () {
  }
  // Update is called once per frame
  void Update ()
  {
    //Transform component on this object
    Transform ThisTransform = GetComponent<Transform>();
    //Add 1 to x axis position
    ThisTransform.position += new Vector3(1f,0f,0f);
  }
}
```

Новые термины и **важные слова** будут выделены полужирным шрифтом. Слова, которые вы увидите на экране, например в меню или в диалоговых окнах, будут отображены в нашем тексте так: «Как только кривая будет закончена, для тестирования кода запустите проект в Unity, и вы увидите эффект на закладке **Game**».


Так будут оформляться предупреждения и важные примечания.


Так будут оформляться советы и рекомендации.

Обратная связь с читателями

Для нас важны отзывы наших читателей. Дайте нам знать, что вы думаете об этой книге, что вам понравилось или, может быть, не понравилось. Отзывы читателей важны для определения наиболее нужных вам направлений.

Чтобы связаться с нами, просто отправьте электронное письмо по адресу feedback@packtpub.com, указав название книги в теме вашего сообщения.

Если есть область, которую вы хорошо знаете, и хотите написать книгу или посодействовать выходу книги, ознакомьтесь с нашим руководством для авторов на www.packtpub.com/authors.

Поддержка пользователей

Если вы гордый владелец книги, изданной в Packt, мы готовы предоставить вам дополнительные услуги, чтобы ваша покупка принесла вам максимальную пользу.

Загрузка активов к книге

Вы сможете загрузить файлы с примерами кода из вашего аккаунта на http://www.packtpub.com для всех купленных вами книг издательства Packt Publishing. Где бы вы не купили эту книгу, вы можете посетить страницу сайта http://www.packtpub.com/support и зарегистрироваться для получения файлов по электронной почте.

Скачивание цветных изображений из этой книги

Мы также предоставляем в ваше распоряжение файл формата PDF, который содержит цветные изображения скриншотов и диаграмм, используемых в этой книге. Цветные изображения помогут вам лучше понять содержание книги. Вы можете загрузить этот файл по ссылке https://www.packtpub.com/sites/default/files/downloads/4813OT_ColoredImages.pdf.

Опечатки

Хотя мы тщательно проверили содержание нашей книги, ошибки все же случаются. Если вы нашли ошибку в любой из наших книг, в тексте или программном коде, мы будем благодарны, если вы сообщите об этом нам. Сделав это, вы оградите других читателей от разочарования и поможете нам в улучшении следующих изданий данной книги. Если вы нашли опечатку, пожалуйста, сообщите о ней, для этого посетите сайт http://www.packtpub.com/support, выберите вашу книгу, щелкните по ссылке let us know и введите описание опечатки. Как только ваше сообщение об опечатке будет проверено, ваше обращение будет принято, и опечатка будет добавлена в общий список опечаток. Общий список опечаток можно просмотреть, выбрав нужную книгу, на http://www.packtpub.com/support.

Пиратство

Пиратство в отношении авторского права в Интернете является общей проблемой всех издателей.

В Packt мы относимся к защите авторских прав очень серьезно. Если вы столкнетесь с любыми нелегальными копиями в Интернете, сообщите нам адрес или название веб-сайта, чтобы мы могли принять необходимые меры.

Пожалуйста, сообщите нам о пиратских копиях по электронному адресу copyright@packtpub.

Мы ценим вашу помощь в защите наших авторов и наших возможностей в предоставлении вам ценной информации.

Вопросы

Если у вас возникнут любые вопросы по книге, вы можете связаться с нами по адресу questions@packtpub.com.

Глава 1

Основные принципы анимации

Привет и добро пожаловать в путешествие по множеству функций анимации в мире Unity. Важность анимации нельзя переоценить. Без анимации все в игре будет статичным, безжизненным и даже скучным. Это справедливо абсолютно для всего в играх: двери должны открываться, персонажи – двигаться, листья – колебаться ветром, искорки и частицы – вспыхивать и гаснуть и т. д. Следовательно, обучение анимации, безусловно, даст вам как разработчику перспективы карьерного роста. Так как анимация присутствует на игровом поле большинства игр, то она имеет отношение ко всем членам команды, это очевидно для художников и аниматоров, но касается и программистов, дизайнеров звука и дизайнеров оформления уровней. Эта книга одновременно ценна и актуальна для большинства разработчиков, она направлена на быстрое и эффективное внедрение основных концепций и практики анимации реального времени в игры, в частности в Unity. К концу этой книги, если вы прочтете каждую главу внимательно, вы получите прочные знания и навыки в области анимации. Вы сможете добиться того качества анимации, которое отражает ваше художественное видение, а также ознакомиться с тем, как и где вы можете расширить свои знания до следующего уровня. Но для того, чтобы достигнуть этой стадии, мы начнем здесь, в первой главе, с самых основных понятий анимации – основ для понимания любой анимации.

Основные понятия анимации

На фундаментальном уровне анимация основывается на отношении между двумя специфичными и независимыми свойствами, а именно **изменениями**, с одной стороны, и **временем** – с другой. Технически анимация определяет изменение во времени, то есть как свойство или величина изменяется во времени, например как положение автомо-

биля меняется с течением времени или как цвет светофора переходит с течением времени с красного в зеленый. Таким образом, каждая анимация внутри промежутка времени (продолжительности) или на протяжении всей своей жизни меняет свойства объектов в определенные моменты (кадры), помещенные в любом месте от начала до конца анимации.

Это определение само по себе техническое и несколько сухое, но актуально и важно. Хотя и не позволяет должным образом охватить эстетические и художественные свойства анимации. Через анимацию и через креативное изменение свойств с течением времени можно эффективно передать настроение, атмосферу, вселенные и идеи. Невероятно, но эмоциональные и художественные образы, которые обеспечивает анимация, в конечном счете являются продуктами главного соотношения изменений с течением времени. В рамках этой же смены в течение долгого времени мы можем определить дальнейшие ключевые термины, в частности термины компьютерной анимации. Вы, возможно, уже знакомы с этими понятиями, но давайте определим их более формально.

Кадры

В анимации время должно обязательно быть разделено на отдельные и дискретные единицы, где могут произойти изменения. Эти моменты называются **кадрами** (frames). Время, по существу, есть непрерывное и неуничтожимое понятие, поскольку всегда можно поделить единицу времени (например, секунды), чтобы получить еще меньшую единицу времени (например, миллисекунды), и т. д. В теории этот процесс деления, по существу, может продолжаться до бесконечности, в результате чего будут получены все меньшие и меньшие доли времени. Понятие **момента** времени или **события** во времени, в отличие от этого, человек сделал дискретным и автономным объектом. Оно является дискретным, чтобы соответствовать нашему восприятию, согласоваться с нашим реальным опытом. В отличие от времени, момент – это нечто, что не может быть разбито на что-то меньшее, не переставая существовать вообще. Внутри момента или кадра что-то происходит. Кадр предоставляет возможность свойствам измениться: двери – открыться, персонажу – передвинуться, цвету – поменяться и т. д. В видеоанимации, в частности, каждая секунда содержит определенное количество кадров. Количество кадров в одной секунде будет варьироваться от компьютера к компьютеру, в зависимости от количественных характеристик аппаратуры, установленного про-

граммного обеспечения и других факторов. Емкость кадров в секунду называется частотой кадров в секунду (FPS). Она часто используется в качестве меры производительности игр, более низкая частота кадров, как правило, связана с подергиванием и низкой производительностью. Рассмотрим рис. 1.1, показывающий, как кадры делят время на периоды.

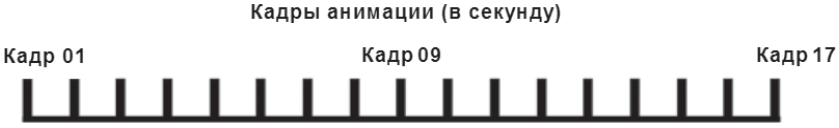

Рис. 1.1. Разделение времени кадрами

Ключевые кадры

Хотя кадр и предоставляет возможность для изменений, но это не обязательно значит, что изменение *произойдет*. Секунда может включать в себя множество кадров, но не в каждом кадре нужны будут изменения. Более того, даже если изменения и необходимы в каждом кадре, аниматорам было бы очень утомительно отдельно определять каждый кадр действия. Одним из преимуществ компьютерного метода анимации, в отличие от ручного, или «старого», метода анимации, является то, что он может сделать нашу жизнь проще. Аниматоры могут определить только ключевые или важные кадры в анимационной последовательности, а затем компьютер автоматически сгенерирует промежуточные кадры. Рассмотрим простую анимацию открытия стандартной межкомнатной двери наружу путем поворота ее на петлях на 90 градусов. Анимация начинается с двери в закрытом положении и заканчивается в открытом положении. Здесь мы определили два основных состояния двери (открыта и закрыта), и эти положения отмечают начало и конец последовательности анимации. Они называются **ключевыми кадрами** (key frames), потому что они ключевые моменты в течение анимации. На основании ключевых кадров Unity (как мы увидим) может осуществлять **автоматическую генерацию кадрами анимации** (tweens), плавно поворачивая дверь от его положения в начальном кадре до положения в конечном кадре. Математический процесс генерации отрезков называется интерполяцией. На рис. 1.2 показано, как кадры генерируются между ключевыми кадрами.

Рис. 1.2. Автоматическая генерация между ключевыми кадрами с использованием интерполяции

Типы анимации

В предыдущем разделе были определены основные понятия, лежащие в основе анимации в целом. В частности, она охватывает изменения, время, кадры, ключевые кадры анимации и интерполяцию. На основании этого мы можем выделить несколько типов анимации в видеоиграх с технической точки зрения, в отличие от взгляда с художественной точки зрения. Все эти варианты основаны на концепции, которую мы рассмотрели, но они осуществляют ее разными способами. Разделение анимаций на типы важно для Unity, потому что различие в их природе требует, чтобы мы управляли и работали с ними по-разному, с использованием конкретных рабочих процессов и методов, которые мы рассмотрим в следующих главах. Типы анимации перечислены ниже в этом разделе.

Анимация твердого тела

Анимация твердого тела используется для создания готовых последовательностей анимации, которые перемещают или изменяют свойства объектов, с учетом того, что объекты являются единым целым, в отличие от объектов, содержащих движущиеся относительно самих объектов части. Некоторыми из примеров этого типа анимации являются движение гоночного автомобиля по дороге, открытие двери на петлях, полет космического корабля сквозь пространство по своей траектории и полет пианино, падающего из окна здания. Несмотря на различия между этими примерами, у всех из них есть важное общее свойство. В частности, хотя и происходят изменения объекта в ключевых кадрах, но они производятся для единого и цельного объекта. Другими словами, хотя дверь и поворачивается на петлях из закрытого состояния в открытое состояние, она заканчивает анимацию по-прежнему как дверь, с той же внутренней структурой и тем же составом, что и раньше. Она не превращается в тигра или льва. Она не взрывается или не превращается в желе. Она не растворяется

в каплях дождя. В течение анимации дверь сохраняет свою физическую структуру. Она изменяется только с точки зрения ее положения, скорости вращения и масштаба. Таким образом, при анимации твердого тела изменения в ключевых кадрах распространяются только на целые объекты и их свойства высокого уровня. Они не касаются подсвойств и внутренних компонентов и не меняют сущности или собственной формы объектов. Эти виды анимации могут быть определены либо непосредственно в редакторе анимации Unity, как мы это увидим в последующих главах, либо в программах 3D-анимации (таких как Maya, Max или Blender), а затем импортированы в Unity посредством файлов мешей. В *главе 3* «*Встроенная анимация*» будет продолжено знакомство с анимацией твердого тела.

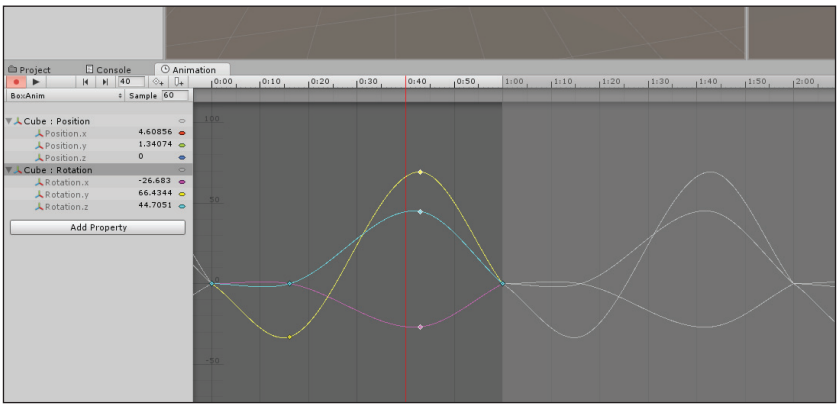

Рис. 1.3. Ключевые кадры для анимации твердого тела

Оснащенная анимация, или анимация на основе скелета

Если вам нужно оживить персонажи человека, животного, питающуюся плотью слизь, взрыв или деформацию объектов, то анимации жесткого тела, скорее всего, будет недостаточно. Вы должны будете использовать анимацию на основе скелета (также называемую изменяющей анимацией). Этот тип анимации изменяет в ключевых кадрах не положение, вращение или масштаб объекта, а движение и деформацию его внутренних частей. Вот как это работает: анимационный художник создает набор специальных объектов костей, подгоняя основной скелет к мешу, что позволит независимо и просто

управлять как внешней, так и внутренней геометрией. Это подходит для создания анимации рук, ног, поворота головы, движений рта, шелеста листьев дерева и многого другого. Как правило, анимация на основе скелета создается как полная последовательность анимации в программе 3D-моделирования и импортируется в Unity как часть файла меша, который может быть обработан и доступен с помощью системы анимации Mecanim Unity. В главах 5, 6 и 7 анимация на основе скелета будет рассмотрена более подробно.

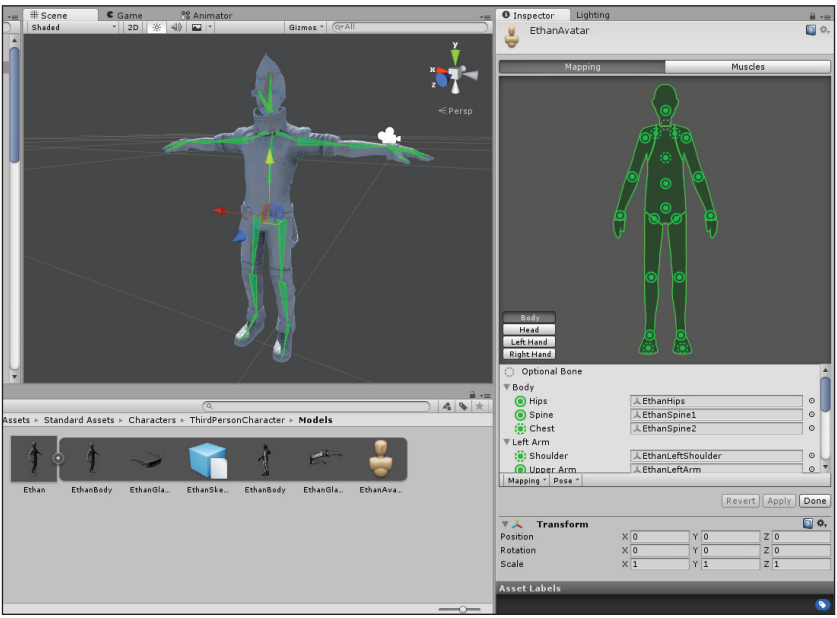

Рис. 1.4. Основанная на скелете анимация, применяемая для персонажа с мешами

Анимация спрайтами

Для 2D-игр, графических пользовательских интерфейсов, а также ряда специальных эффектов в 3D (таких как текстура воды) вам иногда понадобятся стандартные меши с анимированными текстурами. В этом случае объект движется, используя методы анимации твердого тела, но его внутренние части изменяются, как при изменяющей анимации. Это больше похоже на анимацию собственно текстур. Этот тип анимации называется **анимацией спрайтами** (sprite

animation). Анимация состоит из последовательности изображений или кадров и проигрывает их в нужном порядке и с заданной скоростью для получения непрерывной анимации, например цикл ходьбы персонажа в 2D-игре с горизонтальной прокруткой. Более подробная информация об анимации спрайтами приведена в следующей главе.

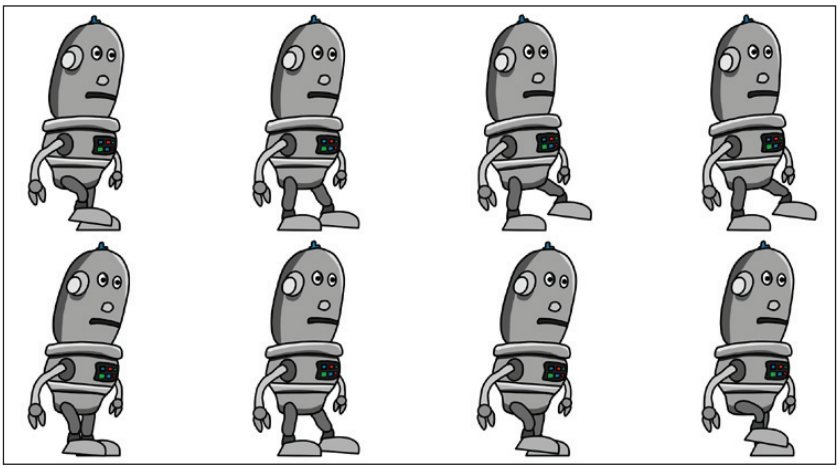

Рис. 1.5. Анимация спрайтами

Анимация, основанная на физике

Во многих случаях вы можете заранее полностью определить вашу анимацию. То есть вы можете ее полностью спланировать и создавать анимационные последовательности для объектов, которые будут проигрываться заданным образом во время выполнения, например цикл ходьбы, открытие двери, взрыв и др. Но иногда вам понадобится анимация, которая выглядит реалистично и, кроме того, еще и реагирует динамично на внешние воздействия, основываясь на решениях, принятых игроком, и других переменных факторах внешнего мира, которые не могут быть предсказаны заранее. Есть разные способы управлять таким поведением, но один из них – это использование физической системы Unity, которая включает в себя компоненты и другие данные, при присоединении их к объектам они заставляют объекты вести себя реалистично. Примерами могут послужить падение объекта на землю под действием силы тяжести или изгиб и кручение объекта, подобно ткани на ветру.

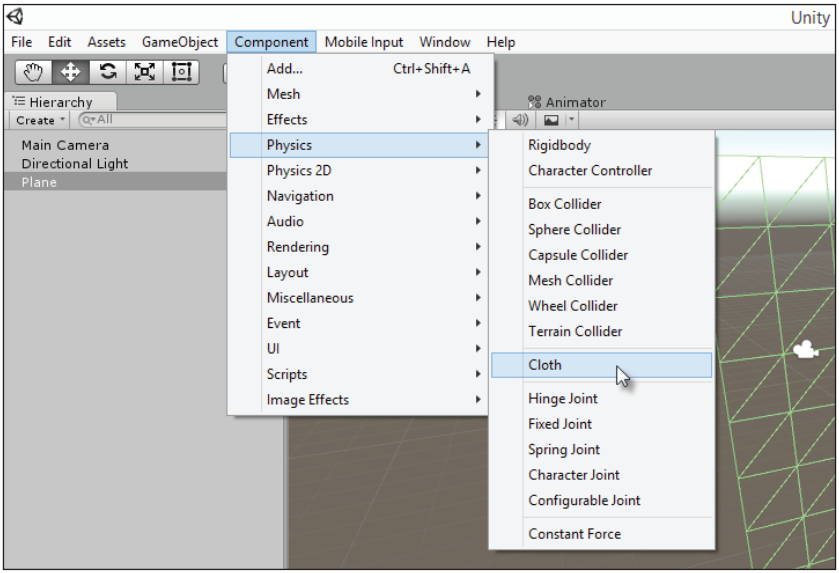

Рис. 1.6. Анимация, основанная на физике

Вы сможете загрузить файлы с примерами кода из вашего аккаунта на http://www.packtpub.com для всех купленных вами книг издательства Packt Publishing. Где бы вы не купили эту книгу, вы можете посетить страницу сайта http://www.packtpub.com/support и зарегистрироваться для получения файлов по электронной почте.

Анимации превращения

Иногда ни один из методов анимации, о которых вы уже прочли, – ни анимация твердого тела, ни анимация, основанная на физике, ни изменяющая анимация, ни анимация спрайтами – не может предоставить вам того, что вам необходимо. Может быть, вы должны превратить что-то одно в совершенно другое, например человека в оборотня, жабу в принцессу или плитку шоколада в замок. В некоторых случаях вам нужно смешать или плавно объединить состояние меша в одном кадре и другое его состояние в другом кадре. Это называется **анимацией превращения** (Morph), или смешиванием форм. По сути, этот метод опирается на фиксацию положения вершин меша в ключевых кадрах анимации и смешивает их состояния с помощью анимации. Недостатком этого метода являются затраты на вычисления. Его применение отрицательно сказывается на производительности, но

результаты могут быть впечатляющими и очень реалистичными. Мы рассмотрим смешивание форм подробно позднее в *главе 7 «Смешивание форм, инверсные кинематики и текстуры с видео»*. На рис. 1.7 показан эффект смешивания форм.

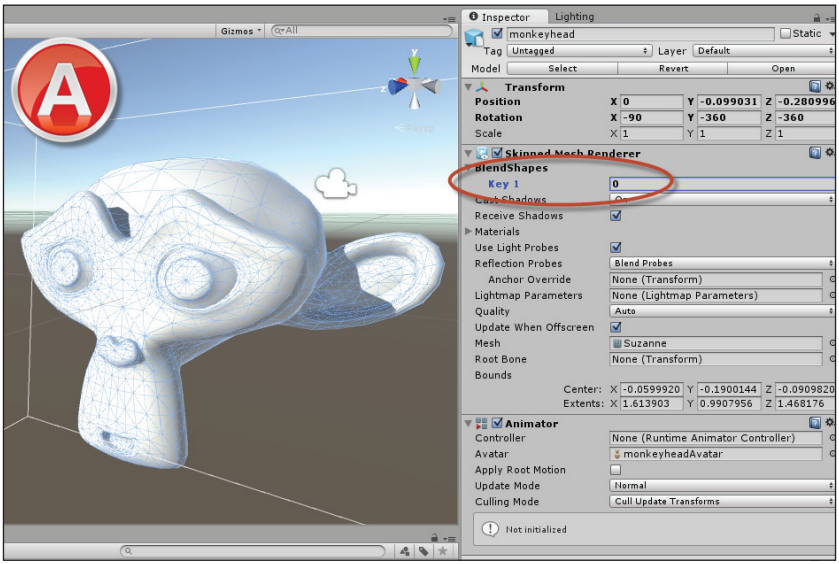

Рис. 1.7. Начальный этап анимации превращения

Смешивание форм переводит модель из одного состояния в другое. Посмотрите на изображение формы для конечного состояния (рис. 1.8).

Видеоанимация

Наверное, одной из наименее известных особенностей анимации в Unity является возможность воспроизводить видеофайлы в виде анимированных текстур как на настольных платформах, так и на мобильных устройствах, таких как iOS и Android. Unity обработает файлы видеоформата OGV (Ogg Theora video) как активы и может воспроизводить видео и звук из этих файлов как анимированные текстуры на меш-объектах в сцене. Это позволяет разработчикам использовать предварительно подготовленные в любом из анимационных пакетов видеофайлы в своих играх (рис. 1.9).

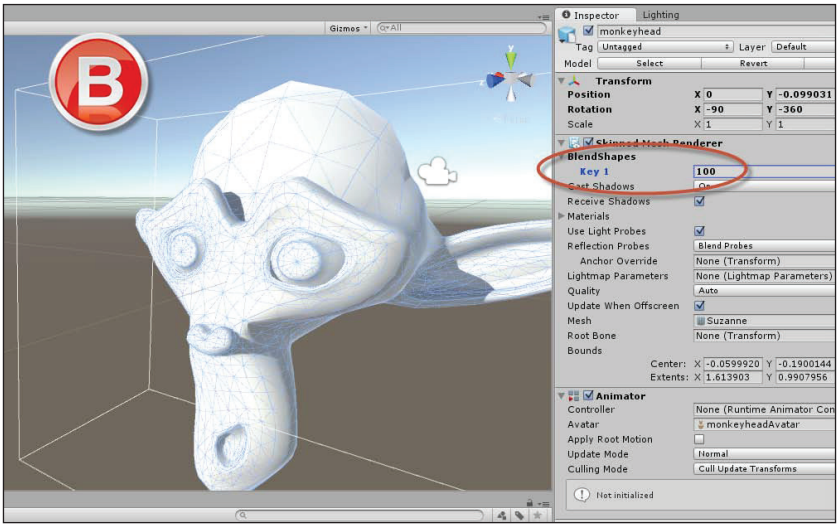

Рис. 1.8. Конечный этап анимации превращения

Рис. 1.9. Видеоанимация

Анимация частицами

До сих пор мы рассматривали методы анимации для четко определенных, осязаемых предметов сцены, таких как спрайты и меши. Это были объекты с четко обозначенными границами, которые отделяют их от других предметов. Но вам достаточно часто будет нужно оживить менее ощутимую, менее прочную и менее физическую материю, такую как дым, огонь, пузыри, искры, туман, рои, фейерверки, облака и др. Для этих целей необходима система частиц. Как мы увидим в главе *3 «Встроенная анимация»*, соответствующие настройки системы частиц позволяют имитировать дождь, снег, стаю птиц и многое другое. На рис. 1.10 представлена система частиц в действии.

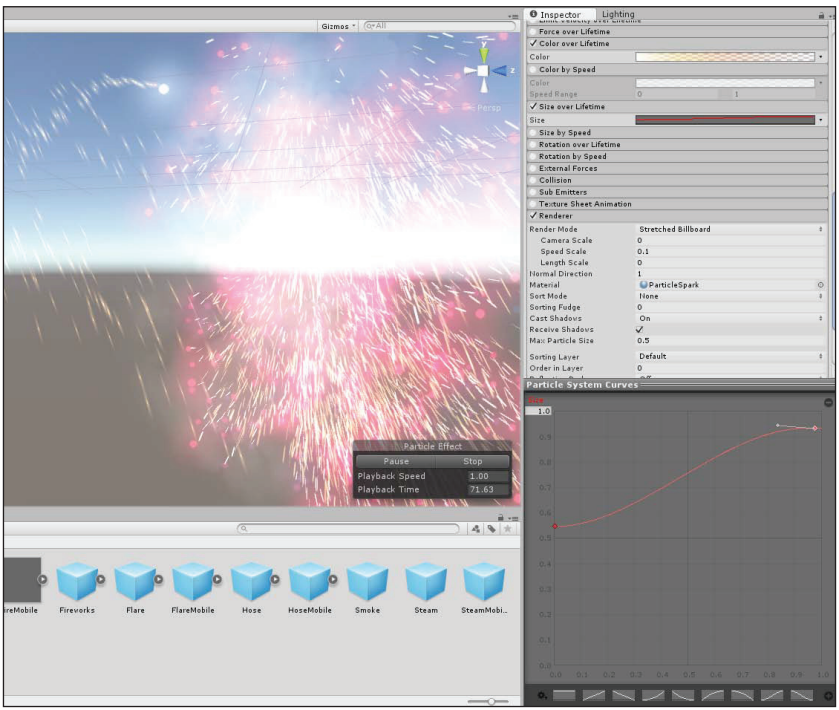

Рис. 1.10. Анимация системой частиц

Программная анимация

Удивительно, но наиболее распространенный тип анимации – это программная анимация, или динамическая анимация. Если вам нуж-

но организовать полет космического корабля по экрану, перемещение по местности управляемого пользователем персонажа или открытие двери при приближении к ней, то вам не обойтись без программной анимации. Все это реализуется с помощью изменений, вносимых в свойства объектов с течением времени программным кодом, который программист специально написал для этой цели. В отличие от других форм анимации, программная анимация не создана заранее дизайнером или аниматором, потому что все ее изменения и комбинации предварительно не известны. Итак, она создается с помощью программного кода и имеет возможность изменяться и корректироваться в соответствии с некими условиями и переменными прямо во время своего выполнения. Конечно, во многих случаях сами анимации подготовлены все же художниками и аниматорами, а код просто переключает или направляет анимации во время выполнения. Вы узнаете больше о программной анимации в следующих разделах этой главы.

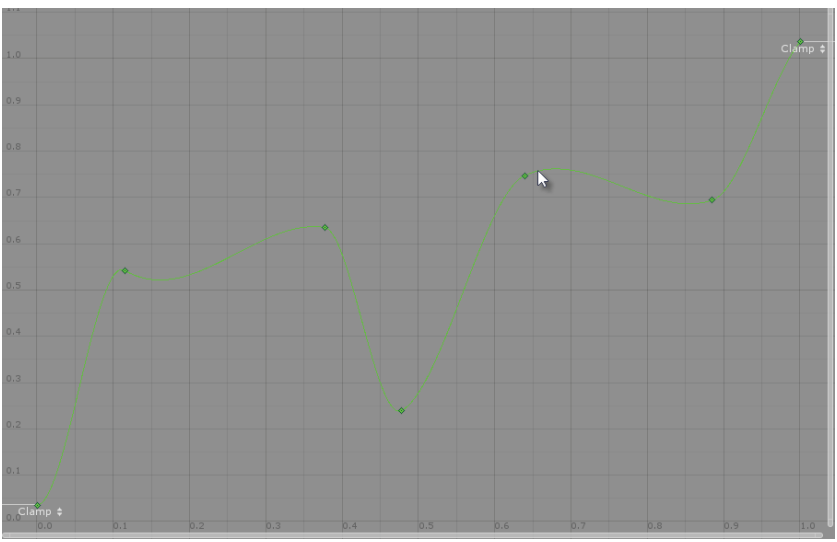

Рис. 1.11. Программная анимация, управляемая скриптом

Анимация с помощью кода – придание предметам движения

Анимация с помощью кода – это отличный способ, чтобы начать работу с анимацией, он демонстрирует все основные понятия, которые мы рассмотрели до сих пор, так что давайте попробуем. В этом разделе и всех последующих мы будем использовать C# для написания скриптов, везде, где это возможно. Однако, поскольку эта книга фокусируется именно на анимации, я не буду здесь объяснять основные понятия программирования (например, переменные, циклы и функции). Вместо этого я предполагаю, что вы уже обладаете хотя бы начальными знаниями о программировании. Если вы хотите научиться программировать, то я рекомендую мой видеокурс *C# для разработчиков Unity (C# For Unity Developers)* (доступен на сайте http://3dmotive.com/) и книгу издательства Packt Publishing *Learning C# by Developing Games with Unity 3D Beginner's Guide*.

К концу нескольких следующих разделов мы создадим объект игры (для примера космический корабль), который сможет путешествовать с постоянной скоростью в одном направлении по уровню игры. Начнем с создания нового файла скрипта, щелкнув правой кнопкой по панели проекта. Выполним **Create → C# Script** в контекстном меню. В качестве альтернативы можно выполнить **Assets → Create → C# Script** из меню приложения, как показано на рис. 1.12. Имя файла Mover.cs. Этот скрипт будет прикреплен к некому объекту, который будет двигаться.

Теперь откройте файл скрипта в редакторе скриптов MonoDevelop двойным щелчком по нему на панели проекта. По умолчанию во всех новых скриптах создаются две функции: `Start` и `Update` (рис. 13). Функция `Update` имеет особое значение для анимации, потому что она связана с кадрами и частотой кадров. В частности, она вызывается во время выполнения на каждом кадре. Это значит, что для игры с частотой кадров 70 функция `Update` вызывается 70 раз в секунду для каждого объекта, к которому прикреплен скрипт, при условии что объект является активным. Именно это делает функцию `Update` особенно важной для анимации, потому что дает нам возможность регулировать свойства объекта с течением времени.

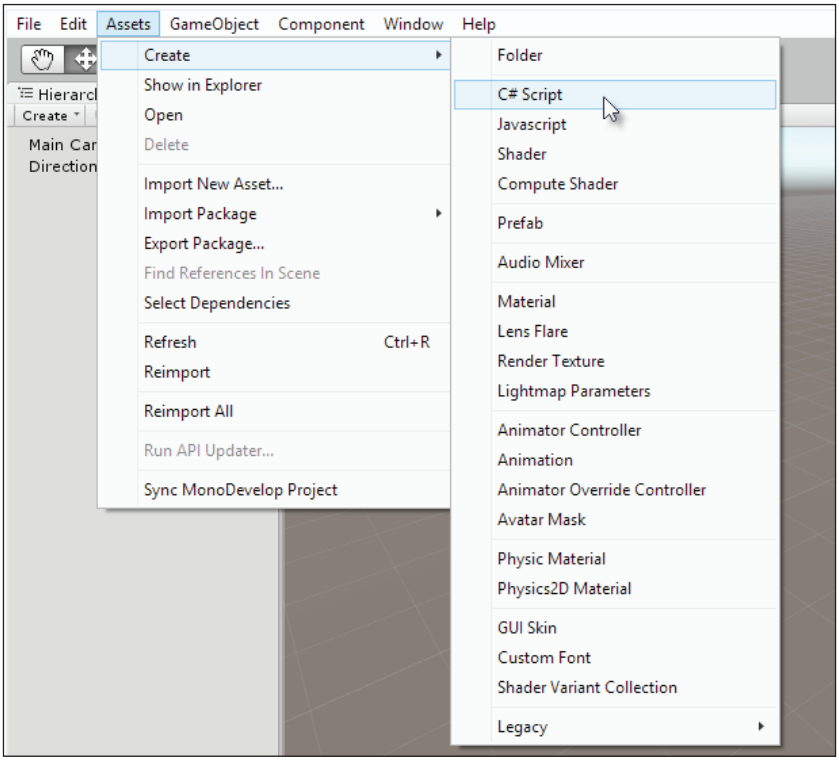

Рис. 1.12

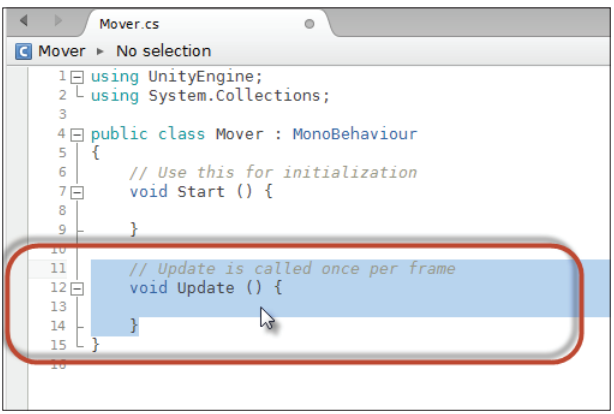

Рис. 1.13. Функция `Update` вызывается на каждом кадре

Для того чтобы вызвать перемещение объекта, давайте напишем эту функцию `Update` так, как это показано в следующем фрагменте. Она будет обращаться к компоненту **transform** объекта и увеличивать его текущую позицию по оси *x* на 1 единицу (метр) на каждом кадре.

Пример кода 1.1. Перемещение объекта

```
using UnityEngine;
using System.Collections;

public class Mover : MonoBehaviour
{
  // Use this for initialization
  void Start () {

  }
  // Update is called once per frame
  void Update ()
  {
    //Transform component on this object
    Transform ThisTransform = GetComponent<Transform>();

    //Add 1 to x axis position
    ThisTransform.position += new Vector3(1f,0f,0f);
  }
}
```

Если вы еще не проверили код, то перетащите файл скрипта на объект в сцене. Затем нажмите кнопку **Play**. В зависимости от выбранной вами точки обзора объект, возможно, будет двигаться слишком быстро, чтобы быть видимым. Удостоверьтесь, что вы расположили камеру так, чтобы обеспечить себе хороший обзор, и вы увидите ваш объект полным жизни при помощи перемещения с постоянной скоростью по оси *x* (рис. 1.14).

Совместимая анимация – скорость, время и deltaTime

Код в примере 1.1 работает, но есть одна важная проблема анимации, и мы должны решать ее прямо сейчас. Как мы видели, объект перемещается вдоль оси *x* на 1 единицу при каждом вызове функции `Update`, то есть на каждом кадре. Это потенциальная проблема, потому что частота кадров отличается на разных компьютерах и даже в разное время на одном и том же компьютере. Это означает, что разные пользователи получат различные картины при использовании нашего кода, потому что объект будет двигаться с разной скоростью. В част-

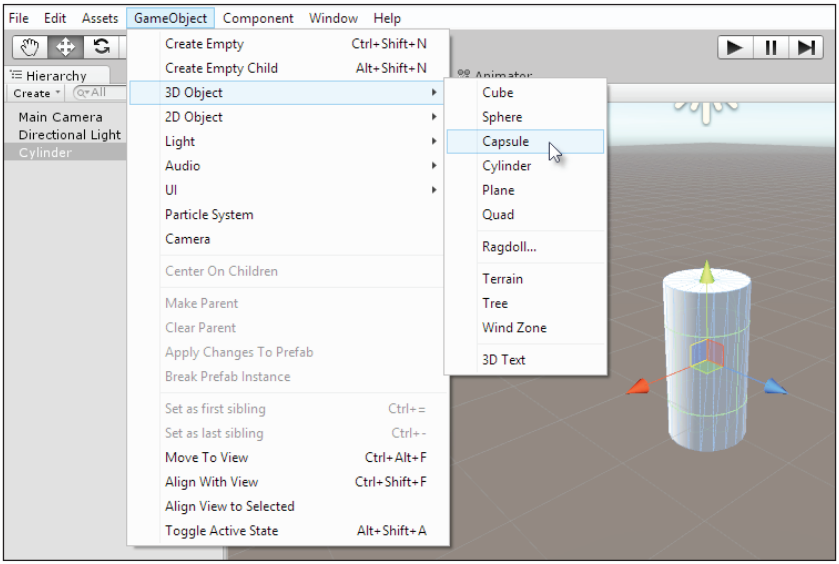

Рис. 1.14. Добавление скрипта Mover к объекту в сцене

ности, в системе с частотой кадров 70 объект сместится на 70 единиц за секунду. Но на другой системе, работающей с частотой кадров 90, объект переместится за то же время на 90 единиц. Это плохо, потому что мы хотим, чтобы все пользователи, которые взаимодействуют с игрой, ощущали скорость одинаковым образом. Этот вопрос особенно критичен для многопользовательских игр, где все пользователи должны действовать синхронно. Чтобы решить эту проблему, мы должны подойти к задаче по-другому, учитывая скорость и время.

Для расчета, как далеко объект должен переместиться за период времени, мы можем использовать формулу *скорость–расстояние–время*, где *пройденное расстояние* = *скорость* × *время*. Это означает, что объект, перемещающийся на 10 метров за 1 секунду, в течение 5 секунд переместится на 50 метров. Этот способ мышления о движении не учитывает ни частоту кадров, ни сами кадры. Кроме того, он не связывает движение с функцией `Update` и частотой ее вызова. Вместо этого он отображает движение на время напрямую, рассчитывая, что время течет одинаково на всех компьютерах; то есть 1 секунда – это всегда 1 секунда. При написании кода в Unity для решения этой проблемы мы можем использовать переменную `DeltaTime`. Рассмотрим пример кода 1.2, в котором обновлен и улучшен пример кода 1.1.

Пример кода 1.2. Установка скорости объекта

```
using UnityEngine;
using System.Collections;

public class Mover : MonoBehaviour
{
  //Amount to travel (in metres) per second
  public float Speed = 1f;

  // Update is called once per frame
  void Update ()
  {
    //Transform component on this object
    Transform ThisTransform = GetComponent<Transform>();

    //Update X Axis position by 1 metre per second
    ThisTransform.position += new Vector3(Speed * Time.deltaTime,0f,0f);
  }
}
```

Переменная `DeltaTime` является встроенной переменной Unity, и ее значение обновляется в каждом кадре, она является частью класса `Time`. В каждом кадре она выражает (в секундах), сколько времени прошло с момента предыдущего кадра. Поэтому если `DeltaTime` равно 0,5, то это означает, что с момента предыдущего кадра прошло 1/2 секунды, то есть прошло 1/2 секунды с того момента, когда функция `Update` была вызвана прошлый раз. Это очень важная информация, потому что при умножении на значение скорости мы масштабируем значение скорости, приводя его в соответствие с частотой кадров для данного компьютера, обеспечивая одну и ту же скорость для разных компьютеров. Делая это, мы делаем любое значение скорости одинаковым для всех. Теперь попробуйте отредактированный код и ощутите разницу (рис. 1.15). Наши объекты будут двигаться с одинаковой скоростью на всех компьютерах.

Движение в направлении

Используя пример кода 1.2, мы получили объект, который движется по оси x с постоянной скоростью. Но как адаптировать этот код так, чтобы заставить объект двигаться в другом направлении? Если бы мы захотели, чтобы объект стал двигаться в направлении только оси y или лишь оси z, то мы бы смогли легко откорректировать для этого код. Но что нужно сделать для получения движения в любом направлении, в том числе и по диагонали? Для этого нам понадобятся векторы. Вектор является набором трех координат в виде (x, y, z)

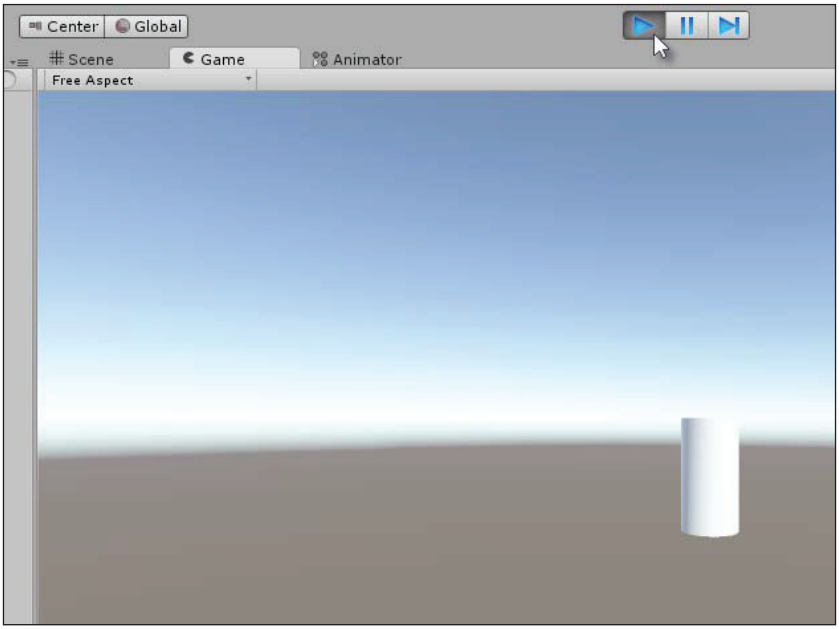

Рис. 1.15. Нажмите кнопку **Play**
и протестируйте обновленный код

и представляет собой направление. Например, (0, 1, 0) означает вверх (так как осью, определяющей движение вверх-вниз, является ось *y*), а (0, 0, 1) означает вперед (так как осью, определяющей движение вперед-назад, является ось *z*).

Пример кода 1.3. Управление направлением

```
using UnityEngine;
using System.Collections;

public class Mover : MonoBehaviour
{
  //Amount to travel (in metres) per second
  public float Speed = 1f;

  //Direction to travel
  public Vector3 Direction = Vector3.zero;

  // Update is called once per frame
  void Update ()
  {
```

```
    //Transform component on this object
    Transform ThisTransform = GetComponent<Transform>();

    //Update position in specified direction by speed
    ThisTransform.position += Direction.normalized * Speed * Time.
deltaTime;
  }
}
```

Теперь вернемся к объекту в сцене. Инспектор объектов предоставляет возможность редактирования значения переменной `Direction` для задания направления, в котором должен переместиться объект. Для задания движений вдоль оси *x* подойдут значения (1, 0, 0) или (–1, 0, 0), заметим, что значения для других двух осей одинаковы. Вы также можете выполнить перемещение по диагоналям, используя значение (1, 1, 1), что будет означать движение по всем трем осям одновременно.

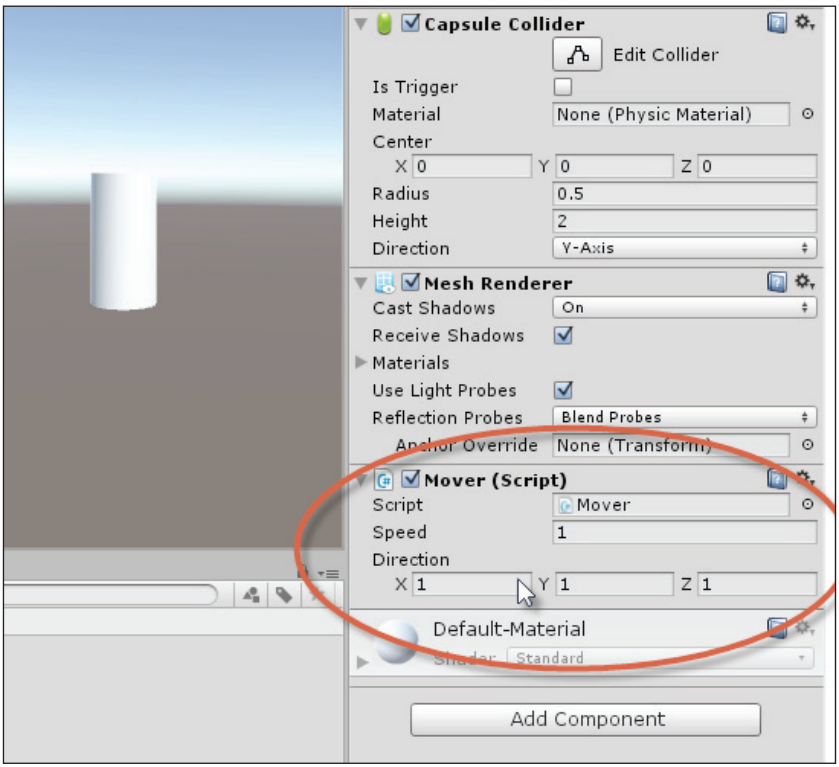

Рис. 1.16. Определение направления для скрипта Mover

Соответствующий проект Unity можно найти в прилагаемых к этой книге файлах в папке Chapter01/Moving Object.

Программирование автогенерации с помощью анимационных кривых

Для объектов, которые перемещаются непрерывно с постоянной скоростью и по прямой, код примера 1.3 работает именно так, как задумано. Но при анимации вам, как правило, требуется, чтобы объекты двигались по изогнутым траекториям, а не только по прямой. Или вам будет нужно, чтобы объекты перемещались с переменной скоростью, а не с постоянной. Чтобы решить эту проблему, мы можем использовать анимационные кривые (animation curves), которые как специальные объекты (доступны только в Unity Pro) позволят нам строить кривые, определяющие автогенерацию для анимации, управляемую изменениями объектов в ключевых кадрах. Рассмотрим пример кода 1.4, который позволит варьировать скорость объекта с течением времени, согласно анимационной кривой.

Более подробную информацию об анимационных кривых можно найти по адресу http://docs.unity3d.com/Manual/animeditor-AnimationCurves.html.

Пример кода 1.4. Наращивание скорости

```
using UnityEngine;
using System.Collections;

public class Mover : MonoBehaviour
{
  //Maximum Speed to travel (in metres) per second
  public float Speed = 1f;

  //Direction to travel
  public Vector3 Direction = Vector3.zero;

  //Curve to adjust speed
  public AnimationCurve AnimCurve;

  // Update is called once per frame
  void Update ()
  {
    //Transform component on this object
    Transform ThisTransform = GetComponent<Transform>();

    //Update position in specified direction by speed
    ThisTransform.position += Direction.normalized * Speed * AnimCurve.
Evaluate(Time.time) * Time.deltaTime;
  }
}
```

Выберите объект анимации с прикрепленным примером кода 1.4 в сцене и просмотрите инспектор объектов. Общедоступная переменная `AnimCurve` будет отображена в виде графика (рис. 1.17).

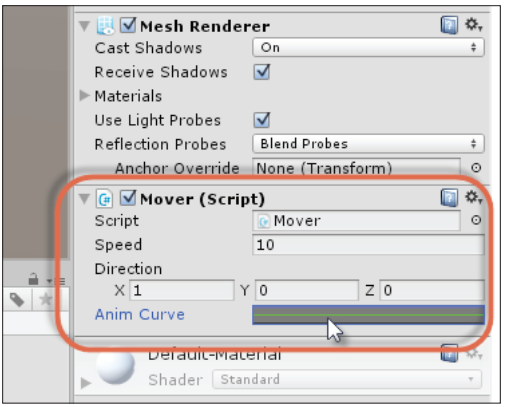

Рис. 1.17. Доступ к редактору анимационных кривых

Щелкните по графику в инспекторе для открытия диалогового окна редактора графика. Этот график позволит вам управлять анимацией применительно к значению скорости. Горизонтальной оси соответствует время (в секундах), а вертикальной – значение скорости (рис. 1.18).

Вы можете щелкнуть по любой из иконок предустановленных кривых в нижнем левом углу окна для создания начальной кривой, управляющей интерполяцией скорости объекта. Вы можете также дважды щелкнуть в любом месте вдоль кривой, чтобы добавить новые контрольные точки, что позволит увеличить контроль над формой кривой. Давайте создадим знаменитый в анимации тип кривой – кривой броска с падением. Эта кривая должна будет постепенно увеличивать скорость объекта в начале анимации (ускорение), а затем в конце уменьшить скорость объекта до его полной остановки (замедление). Для начала, используя колесико прокрутки мыши, можно уменьшить масштаб кривой, чтобы показать горизонтальный отрезок от 5 секунд с начала анимации до 5 секунд до конца анимации. Убедитесь, что первые и последние ключевые кадры покоя моментов начала и окончания анимации, соответственно, находятся на горизонтальной оси. Также убедитесь, что в оба момента точки кривой расположены на от-

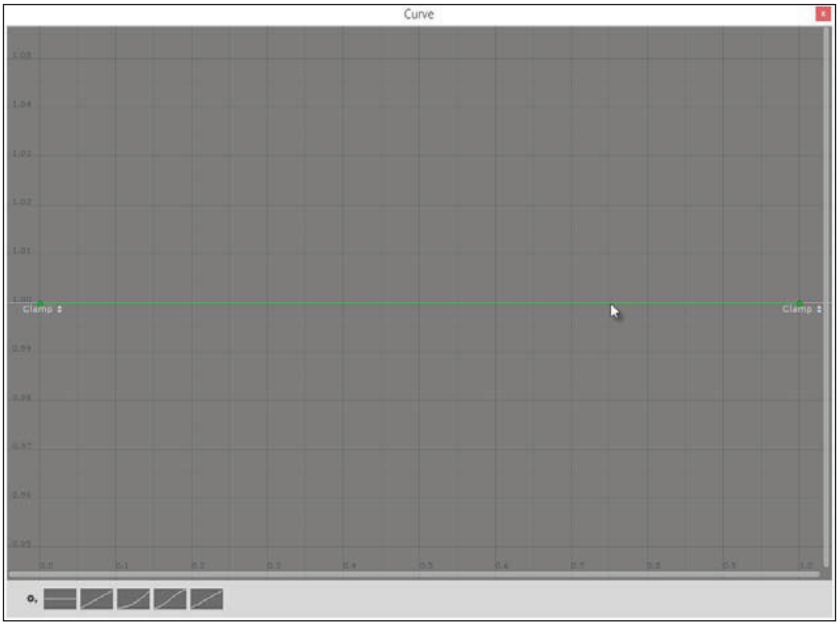

Рис. 1.18. Построение анимационной кривой

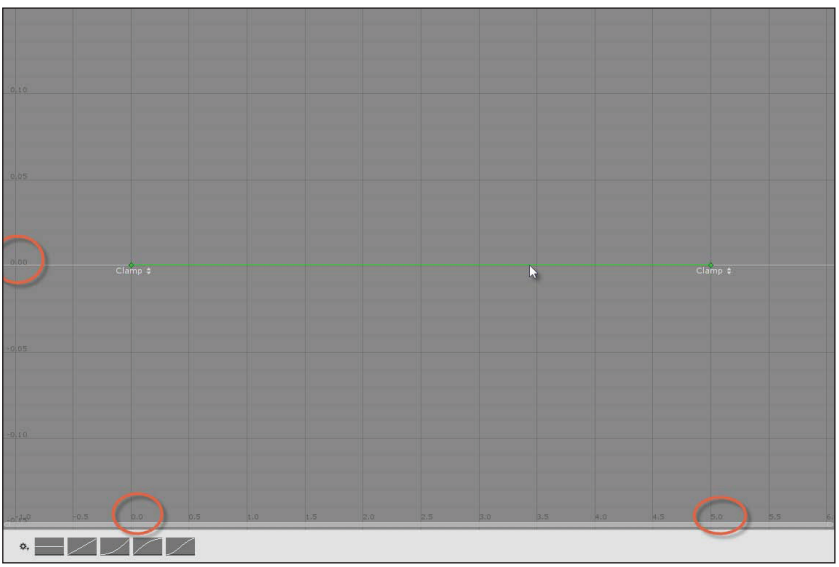

Рис. 1.19. Начальная анимационная кривая

метке 0 вертикальной оси, что означает, что скорость объекта должна быть равна 0 как на начало, так и на конец анимации.

Когда вы перетаскиваете точки, удерживайте клавишу **Ctrl**, чтобы поместить точки на линии сетки.

Чтобы построить кривую броска с падением, добавьте новую точку контроля в центре кривой (для времени, равного 2,5 секунды) и перетащите ее вверх по вертикальной оси до значения 1, которое будет отражать максимальную скорость для объекта. Если при добавлении новой точки контроля на кривой образовался угол, нарушающий ее гладкость, то щелкните правой кнопкой мыши по контрольной точке и выберите опцию сглаживания **Free Smooth** из контекстного меню, чтобы сгладить кривую.

Вы можете нажать клавишу **F** для изменения размеров отображаемой части графика так, чтобы увидеть кривую целиком.

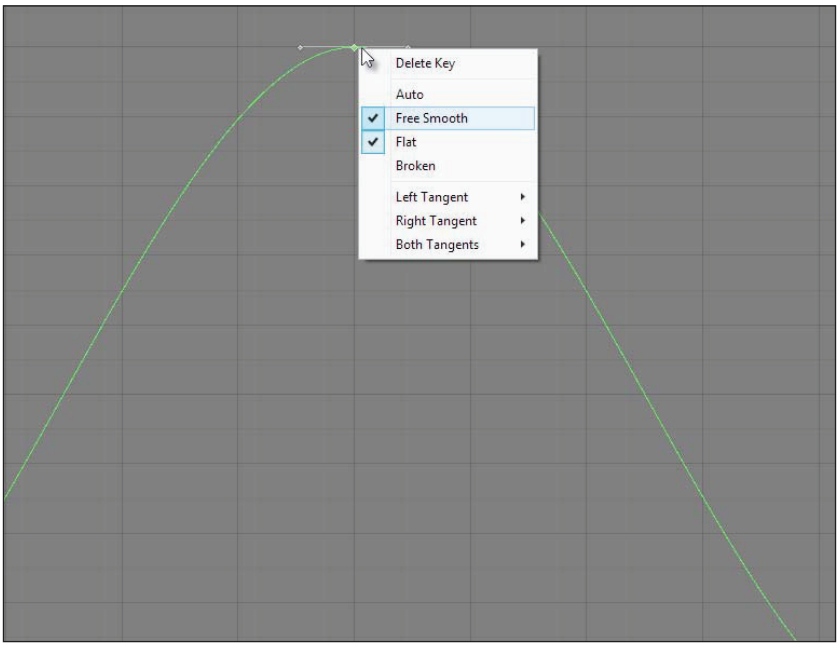

Рис. 1.20. Создание кривой броска с падением для скорости

После построения кривой выполните тестовый запуск проекта Unity и просмотрите эффект на закладке **Game**. С помощью кода из примера 1.4 скорость объекта будет анимирована во времени в соответствии с заданной кривой. Метод `AnimationCurve.Evaluate` принимает значение времени в качестве параметра (по горизонтальной оси) и возвращает соответствующее значение от оси *y* как произведение на скорость. Используя эту функцию, мы сможем оценить любую кривую для программной анимации.

Более подробную информацию о методе оценки (Evaluate) можно найти в официальной документации Unity по адресу http://docs.unity3d.com/ScriptReference/AnimationCurve.html.
Соответствующий проект Unity можно найти в прилагаемых к этой книге файлах в папке Chapter01/animation_curves folder.

Вращение объектов – анимация с помощью сопрограмм

Теперь давайте попробуем другой программный скрипт анимации, использующий **сопрограммы** (coroutines), функции особого вида, которые очень полезны для создания типов поведения, которые разворачиваются во времени. В частности, мы создадим скрипт, который медленно и плавно вращает объект, оставляя его всегда повернутым лицом к цели. Это пригодится для создания врагов, которые всегда смотрят на тебя, вращающейся орудийной башни или другого предмета, следящего за целью. Следует подчеркнуть, что нужное нам поведение – это не поведение при использовании стандартной функции `LookAt`, которая при вызове метода `Transform.LookAt` заставляет объекты немедленно сориентироваться на цель. Вместо этого мы запрограммируем такое поведение, при котором объект постоянно вращается с заданной угловой скоростью, держась лицом к цели, как это показано на рис. 1.15. Объект, возможно, и не сможет увидеть свою цель в определенные моменты, но будет всегда пытаться смотреть на нее. Поведение будет включать в себя медленное вращение и повороты, для того чтобы увидеть цель, где бы она не находилась. Рассмотрим пример кода 1.5 из файла `LookAt.cs`.

Более подробную информацию о сопрограммах можно найти в официальной документации Unity по адресу http://docs.unity3d.com/Manual/Coroutines.html.

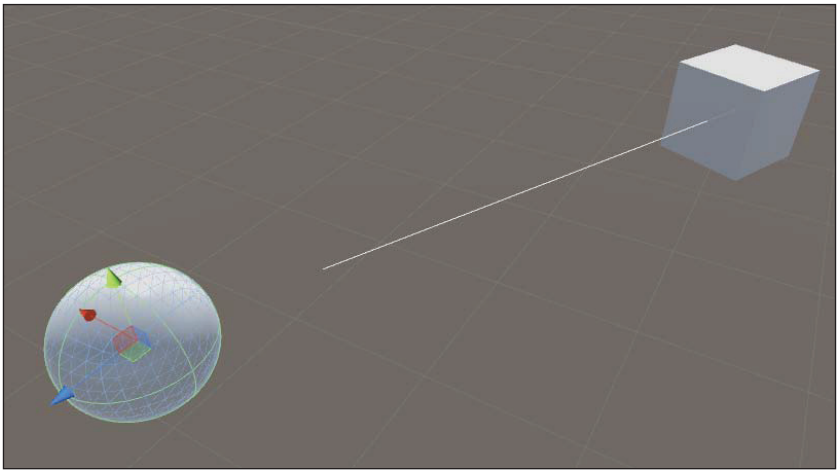

Рис. 1.21. Кубический объект вращается, оставаясь лицом к сфере

Пример кода 1.5. Вращение лицом к цели

```
//-------------------------------------------
using UnityEngine;
using System.Collections;
[ExecuteInEditMode]
//-------------------------------------------
public class LookAt : MonoBehaviour
{
  //Cached transform
  private Transform ThisTransform = null;

  //Target to look at
  public Transform Target = null;

  //Rotate speed
  public float RotateSpeed = 100f;

  //-------------------------------------------
  // Use this for initialization
  void Awake () {
    //Get transform for this object
    ThisTransform = GetComponent<Transform>();
  }
  //-------------------------------------------
  void Start()
  {
    //Start tracking target
```

```
    StartCoroutine(TrackRotation(Target));
  }
  //-----------------------------------------
  //Coroutine for turning to face target
  IEnumerator TrackRotation(Transform Target)
  {
    //Loop forever and track target
    while(true)
    {
      if(ThisTransform != null && Target != null)
      {
        //Get direction to target
        Vector3 relativePos = Target.position - ThisTransform.position;

        //Calculate rotation to target
        Quaternion NewRotation = Quaternion.LookRotation(relativePos);

        //Rotate to target by speed
        ThisTransform.rotation = Quaternion.RotateTowards(ThisTransform.
          rotation, NewRotation, RotateSpeed * Time.deltaTime);
      }

      //wait for next frame
      yield return null;
    }
  }
  //-----------------------------------------
  //Function to draw look direction in viewport
  void OnDrawGizmos()
  {
    Gizmos.DrawLine(ThisTransform.position, ThisTransform.forward.
      normalized * 5f);
  }
  //-----------------------------------------
}
//-------------------------------------------
```

Сопрограммы работают не так, как обычные функции. Они всегда возвращают значение типа `IEnumerator` и содержат, по крайней мере, один оператор `yield`. В отличие от обычных функций, которые выполняются построчно, а затем заканчиваются, после чего выполнение основной программы возобновляется, выполнение сопрограмм выглядит так, как будто они выполняются параллельно с процессом, который их вызвал. Они похожи на потоки или фоновые процессы, хотя и без применения многозадачности, и работают вместе с другими процессами. Это делает их полезными для анимации, сопрограммы позволяют нам оживить объекты, изменив их свойства, в моменты, когда запущены другие процессы.

Соответствующий проект Unity можно найти в прилагаемых к этой книге файлах в папке Chapter01/RotatingObjects.

Материалы и рельефная анимация

Еще один очень полезный метод анимации – это **рельефная анимация** (UV, или mapping animation), пример ее показан на следующем скриншоте. Он заключается в программном изменении координат вершин меша с течением времени с заставлением их скользить по текстуре на поверхности объекта. Этот метод не изменяет пикселей текстуры, а изменяет их расположение на поверхности. С помощью рельефной анимации могут быть созданы такие эффекты, как, например, течение воды, поток лавы, движение облаков, туннельные эффекты и многое другое. Рассмотрим пример кода 1.6 (MatScroller.cs).

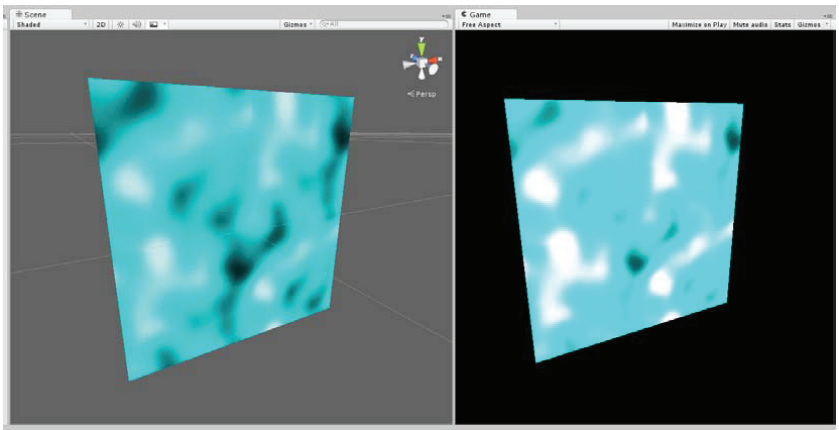

Рис. 1.22. Рельефная анимация текстуры поверхности создает движение облаков, воды или лавы

Пример кода 1.6. Прокрутка материала

```
//CLASS TO SCROLL TEXTURE ON PLANE. CAN BE USED FOR MOVING SKY
//-------------------------------------------------
using UnityEngine;
using System.Collections;
//-------------------------------------------------
[RequireComponent (typeof (MeshRenderer))] //Requires Renderer Filter
Component
public class MatScroller : MonoBehaviour
{
```

```
  //Public variables
  //-----------------------------------------------
  //Reference to Horizontal Scroll Speed
  public float HorizSpeed = 1.0f;

  //Reference to Vertical Scroll Speed
  public float VertSpeed = 1.0f;

  //Reference to Min and Max Horiz and Vertical UVs to scroll between
  public float HorizUVMin = 1.0f;
  public float HorizUVMax = 2.0f;

  public float VertUVMin = 1.0f;
  public float VertUVMax = 2.0f;

  //Private variables
  //-----------------------------------------------
  //Reference to Mesh Renderer Component
  private MeshRenderer MeshR = null;

  //Methods
  //-----------------------------------------------
  // Use this for initialization
  void Awake ()
  {
    //Get Mesh Renderer Component
    MeshR = GetComponent<MeshRenderer>();
  }
  //-----------------------------------------------
  // Update is called once per frame
  void Update ()
  {
    //Scrolls texture between min and max
    Vector2 Offset = new Vector2((MeshR.material.mainTextureOffset.x >
HorizUVMax) ? HorizUVMin : MeshR.material.mainTextureOffset.x + Time.
deltaTime * HorizSpeed,
            (MeshR.material.mainTextureOffset.y > VertUVMax) ? VertUVMin
: MeshR.material.mainTextureOffset.y + Time.deltaTime * VertSpeed);
    //Update UV coordinates
    MeshR.material.mainTextureOffset = Offset;
  }
  //-----------------------------------------------
}
//-------------------------------------------------
```

Этот код может быть присоединен к мешу объекта для анимации его материала. Для этого установите значения переменных `HorizSpeed` и `VertSpeed` в инспекторе объектов для управления горизонтальной и вертикальной скоростями прокрутки материала.

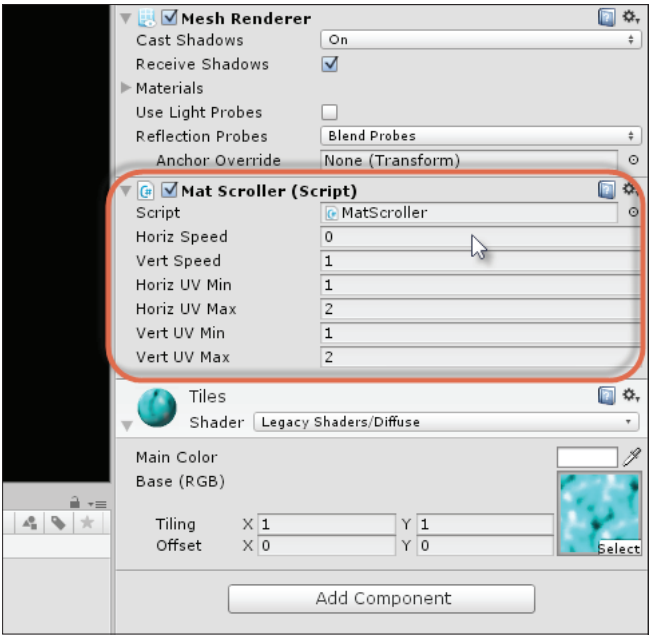

Рис. 1.23. Установка скорости прокрутки материала

Проект Unity для этого задания можно найти в прилагаемых к этой книге файлах в папке Chapter01/texture_animator.

Дрожание камеры – анимационные эффекты

Если вы играли в игры, содержащие нелицеприятные действия, такие как побои или стрельба, то должны были часто видеть эффект дрожания камеры, когда персонажу больно. Эффект дрожания добавляет драматизма и динамики в действие. Этого эффекта анимации легко достичь, основываясь на принципах и идеях, рассмотренных в этой главе. Рассмотрим пример кода 1.7, который может быть добавлен к любой камере сцены для создания эффекта дрожания:

Пример кода 1.7. Дрожание камеры

```
using UnityEngine;
using System.Collections;
//---------------------
public class CameraShake : MonoBehaviour
{
  private Transform ThisTransform = null;
```

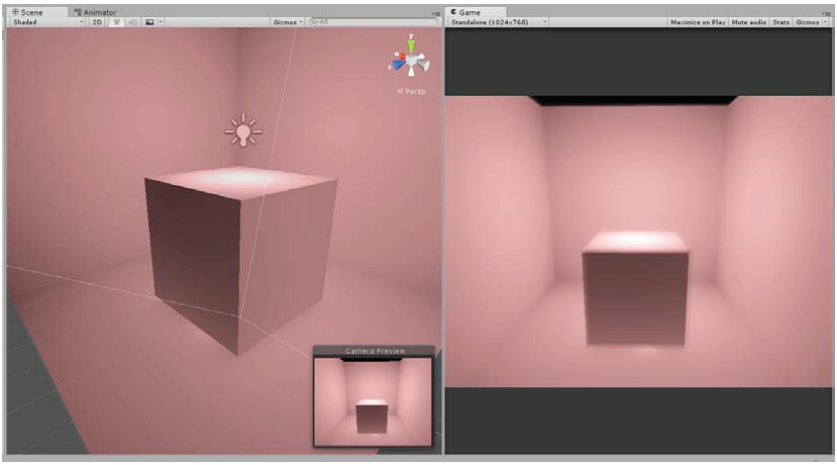

Рис. 1.24. Эффект дрожания камеры

```
//Total time for shaking in seconds
public float ShakeTime = 2.0f;

//Shake amount - distance to offset in any direction
public float ShakeAmount = 3.0f;

//Speed of camera moving to shake points
public float ShakeSpeed = 2.0f;

//----------------------
// Use this for initialization
void Start ()
{
  //Get transform component
  ThisTransform = GetComponent<Transform>();

  //Start shaking
  StartCoroutine(Shake());
}
//----------------------
//Shake camera
public IEnumerator Shake()
{
  //Store original camera position
  Vector3 OrigPosition = ThisTransform.localPosition;

  //Count elapsed time (in seconds)
  float ElapsedTime = 0.0f;

  //Repeat for total shake time
```

```
    while(ElapsedTime < ShakeTime)
    {
      //Pick random point on unit sphere
      Vector3 RandomPoint = OrigPosition + Random.insideUnitSphere *
ShakeAmount;

      //Update Position
      ThisTransform.localPosition = Vector3.Lerp(ThisTransform.
localPosition, RandomPoint, Time.deltaTime * ShakeSpeed);

      //Break for next frame
      yield return null;

      //Update time
      ElapsedTime += Time.deltaTime;
    }

    //Restore camera position
    ThisTransform.localPosition = OrigPosition;
  }
  //---------------------
}
//---------------------
```

Этот пример кода использует сопрограммы для случайного колебания камеры в воображаемой сфере, определяемой с помощью переменной `Random.insideUnitSphere`. Чтобы использовать этот код, просто перетащите скрипт на объект камеры, и вперед!

Проект Unity для этого задания можно найти в прилагаемых к этой книге файлах в папке Chapter01/camera_shake.

Итоги

В этой главе анимация была рассмотрена абстрактно, с точки зрения искусства и с научной точки зрения. Мы познакомились с наиболее распространёнными в играх на Unity типами анимации. Кроме того, мы рассмотрели некоторые основные задачи и идеи программной анимации, в том числе возможность динамического изменения объектов с помощью кода, что позволяет отойти от предопределённых заранее анимаций, которым будет посвящена большая часть этой книги. Хотя в этой главе заканчивается рассмотрение программной анимации как таковой, но тем не менее кодирование и скрипты будут присутствовать на протяжении большей части последующих глав этой книги. В следующей главе, продолжив свое путешествие, мы окунемся в мир анимации 2D-спрайтов.

Глава 2

Анимация спрайтами

В этой главе мы входим в мир анимации для 2D-игр в Unity, а конкретнее в мир анимаций спрайтами и связанного с ними обширного набора функций. Если вы планируете создавать 2D-игры, такие как игры с горизонтальным слайдингом или игры для мобильных телефонов, или если вы создаете анимированные интерфейсы пользователя (GUI) и системы меню, то 2D-анимация займет важное место в ваших проектах.

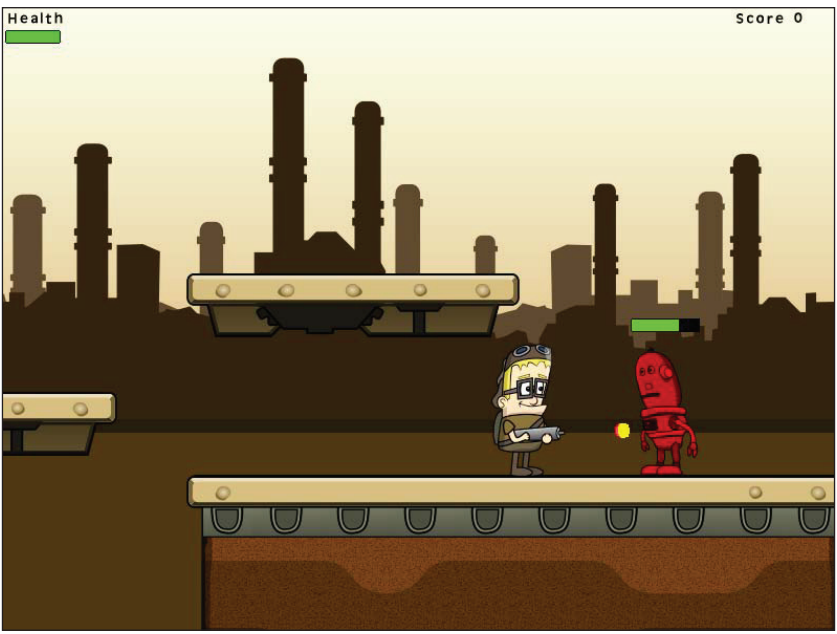

Рис. 2.1. Игра с горизонтальным слайдингом в действии

Прежде чем начать, позвольте мне остановиться на важности 2D-анимации, ее отличиях от 3D-анимации, чтобы определить значимость

того, с чем мы будем иметь дело в этой главе. Под 2D (двумя измерениями) я имею в виду только определенный вид представления, то есть вид обзора любой сцены с использованием либо ортогональной камеры, либо камеры, показывающей объекты плоскими, не позволяющей пользователю сместить точку обзора и взглянуть на сцену под другим углом. В этом смысле даже 3D-сцена может быть частью 2D-игры, потому что 2D относится только к режиму обзора сцены через камеру и не касается каких-либо внутренних свойств сцены. Тем не менее большая часть нашей работы в 2D будет связана со спрайт-объектами, особыми видами объектов для 2D-игр, поддерживаемыми Unity. Итак, давайте начнем. Нашей целью в этой главе будут создание анимированного спрайтами персонажа, который бегает по уровню, и воспроизведение анимации его бега.

Спрайты, их импорт и настройка

Спрайт (sprite) – это специальный 2D-объект Unity. Он позволяет отображать в сцене импортированную плоскую, регулярную текстуру как бы на картонной опоре или в виде рекламного щита. В играх с горизонтальным слайдингом спрайты используются и для анимированных персонажей, и для реквизитов. Применение 2D-спрайтов можно проиллюстрировать на **примере книги с движущимися картинками** (flip-book), где на каждой странице изображен эскиз одного ключевого кадра и читатель, быстро перелистывая страницы, наблюдает эффект анимации. Использованные здесь объекты спрайтов включены в прилагаемые к этой книге файлы, их можно найти в папке Chapter02/assets. Есть два способа импорта и настройки анимированных спрайтов, и в этом разделе мы рассмотрим их оба. Итак, откройте активы и повторяйте мои действия. Давайте начнем с рассмотрения двух способов импорта спрайтов: импорта отдельных спрайтов и импорта атласа спрайтов.

Более подробную информацию о сопрограммах можно найти в онлайн-документации по Unity по адресу http://docs.unity3d.com/ScriptReference/Sprite.html.

Отдельные спрайты

Если вы создали анимационный спрайт, используя отдельные файлы изображений для каждого кадра анимации, то для их импорта вы можете просто перетащить их все сразу в панель проекта Unity.

Рис. 2.2. Импорт анимационных слайдов из отдельных файлов, содержащих по одному кадру

При этом каждый файл будет импортирован как обычная текстура (рис. 2.3).

Для преобразования их в спрайты выберите все импортированные текстуры в панели проекта. В инспекторе объектов выберите опцию **Sprite** для параметра **Texture Type**. Убедитесь, что режим спрайтов **Sprite Mode** установлен в **Single**, так как каждое отдельное изображение представляет собой один кадр для общего спрайта (одного и того же персонажа). Кроме того, настройка **Generate Mip Maps** должна быть сброшена для улучшения качества текстур. И наконец, опция **Pivot** для спрайта должна быть установлена в **Bottom**, низ – место расположения ног большинства спрайтовых персонажей. Это потому, что ноги – это то, чем персонаж касается горизонтальной плоскости, и такой выбор позволит правильно расположить его в пределах сцены. После этого нажмите на кнопку **Apply** для подтверждения.

Помните, что вы не должны применять настройки спрайта отдельно для каждой текстуры. Это было бы утомительно! Вместо этого вы можете выбрать несколько текстур в панели проекта и применить настройки сразу для всех.

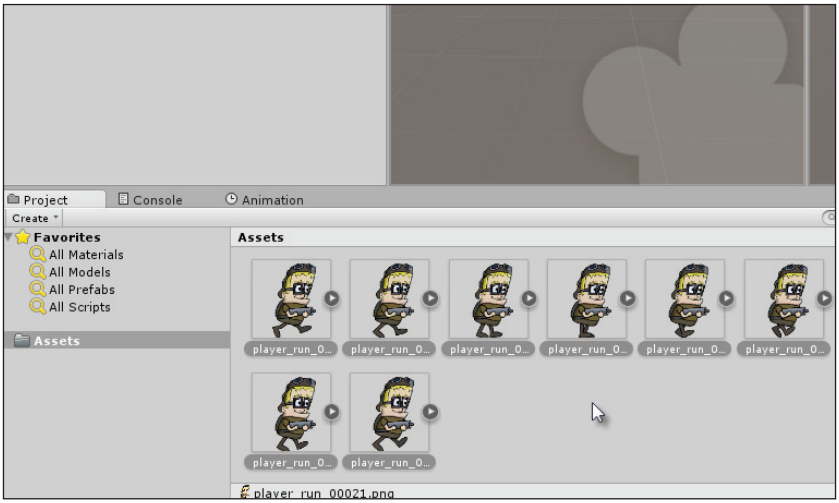

Рис. 2.3. Импорт анимации бега
для персонажа игрока в панель проекта Unity

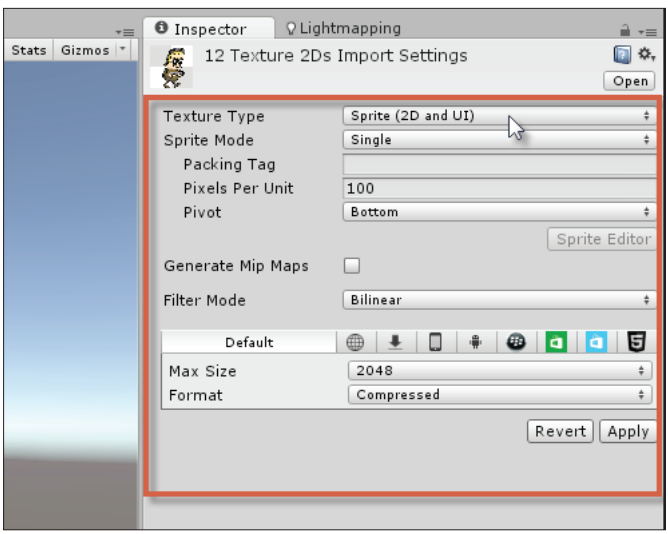

Рис. 2.4. Настройка импортированных текстур спрайтов

Вот и все! Импортированные кадры теперь настроены и готовы к анимации спрайтами в сцене. Как запустить анимацию, мы рассмотрим в этой же главе позднее. Теперь посмотрим, как импортировать несколько иной вид спрайта.

Атлас спрайтов

Иногда спрайты уже собраны вместе на одном листе спрайтов, который называется **текстурой атласа** (texture atlas). Другими словами, все кадры в анимации могут быть помещены в строки и столбцы одного файла текстуры, а не размещаться в отдельных файлах. Этот вид листов спрайтов также поддерживается Unity, но он требует некоторой дополнительной настройки, в том числе и на этапе импорта.

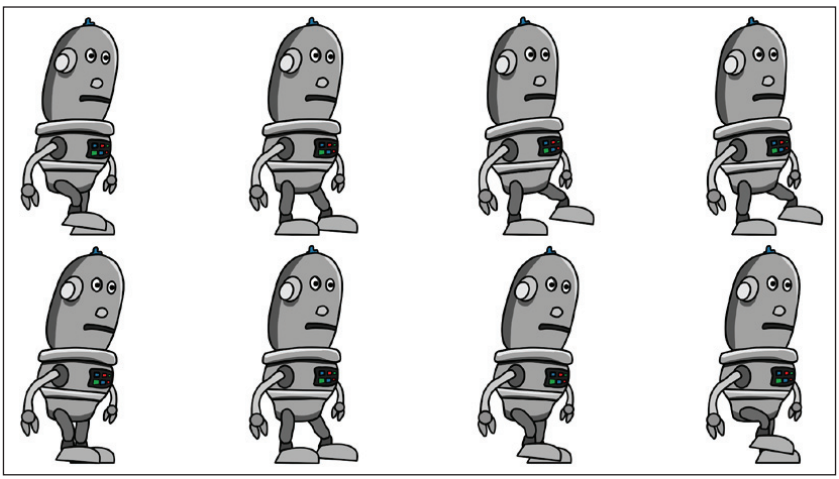

Рис. 2.5. Текстура атласа спрайтов

Чтобы импортировать атлас спрайтов, просто перетащите текстуру в панель проекта Unity. Атлас будет импортирован как обычный файл текстуры. После импорта он появится в панели проекта (рис. 2.6).

После импорта вы должны будете настроить текстуру для работы в качестве спрайта. Чтобы сделать это, выберите текстуру в панели проекта. В инспекторе объектов выберите **Sprite** для опции типа текстуры **Texture Type**. Затем выберите **Multiple** для опции режимов спрайта **Sprite Mode**. Это соответствует тому, что файл текстуры содержит несколько кадров, а не один. Отключите флажок **Generate MIP Maps** для оптимального качества текстур, а затем нажмите на кнопку **Apply**.

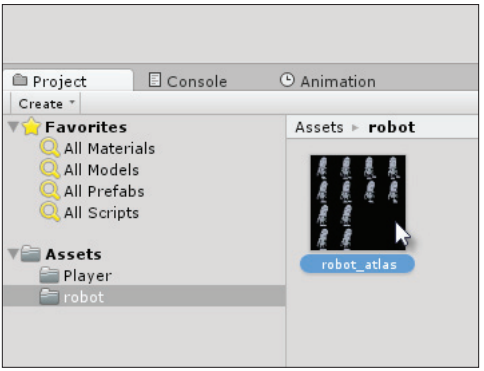

Рис. 2.6. По умолчанию атлас спрайтов импортируется как обычная текстура

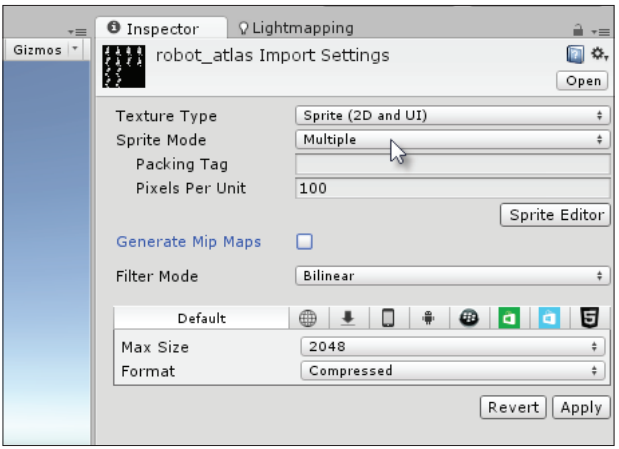

Рис. 2.7. Подготовка текстуры атласа в инспекторе объектов

Выберите **Multiple** для опции **Sprite Mode**, дав понять Unity, что в текстуре присутствует несколько кадров, но Unity еще не знает, где именно они находятся в текстуре. Чтобы определить это, можно использовать **редактор спрайтов** (Sprite Editor). Этот инструмент доступен через щелчок на кнопке **Sprite Editor** в инспекторе объектов, когда текстура выбрана в панели проекта. В диалоговом окне редактора спрайтов **Sprite Editor** вы сможете вручную прорисовать грани-

цы для каждого спрайта, просто перетаскивая рамку вокруг каждого из них в изображении. Вы можете воспользоваться альтернативным подходом, автоматически сгенерировать фрагменты спрайта, задав размеры плитки. В текстуре из файлов, прилагаемых к этой книге, каждый спрайт имеет размеры 512×512. Для их разделения нажмите на кнопку **Slice** в верхнем левом углу окна редактора спрайтов **Sprite Editor**.

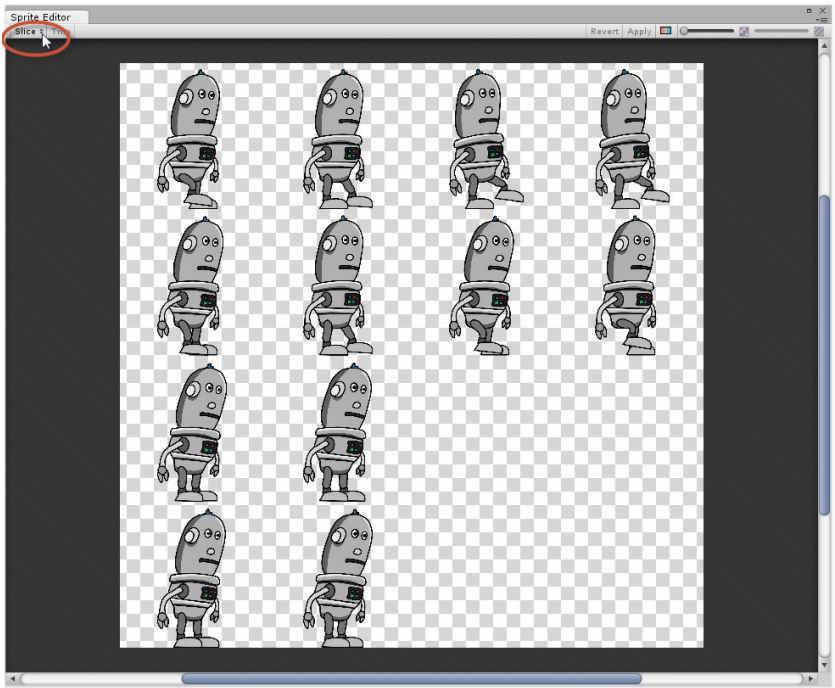

Рис. 2.8. Щелкните по кнопке Slice в редакторе спрайтов для генерации фрагментов спрайта

После нажатия на кнопку **Slice** появляется всплывающее окно конфигурирования. Заполните параметры разреза для генерации фрагментов кадров из атласа. Для текстуры робота, представленной здесь, тип **Slice** должен быть установлен на **Grid** (а не **Automatic**), так как спрайты расположены в изображении в виде сетки. Размеры **512×512** задаются в поле **Pixel**, а опция **Pivot** должна быть установлена в положение **Bottom** на уровень ног персонажа.

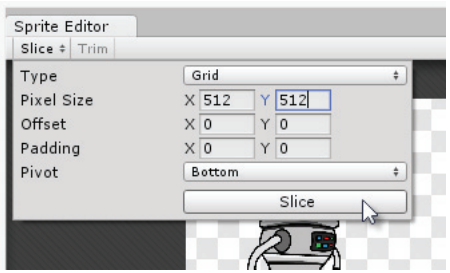

Рис. 2.9. Генерация слайдов из сетки

После нажатия на кнопку **Slice** во всплывающем окне **Slice** Unity поделит текстуру атласа на отдельные спрайты, каждый из которых будет окаймлен выбранной границей. При необходимости вы можете выбрать фрагмент и изменить его настройки. Для текстуры робота сделанное разбиение подходит, поэтому нажмите кнопку **Apply** в верхней панели окна редактора спрайтов **Sprite Editor**.

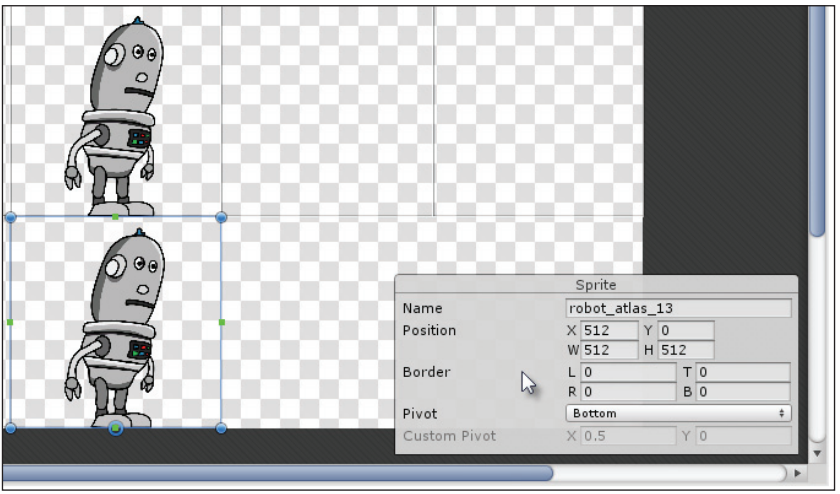

Рис. 2.10. Фрагменты созданы

Нажатие кнопки **Apply** в окне редактора спрайтов приведет к генерации последовательности объектов спрайтов, которые появятся в панели проекта в качестве независимых субъектов, готовых к анимации.

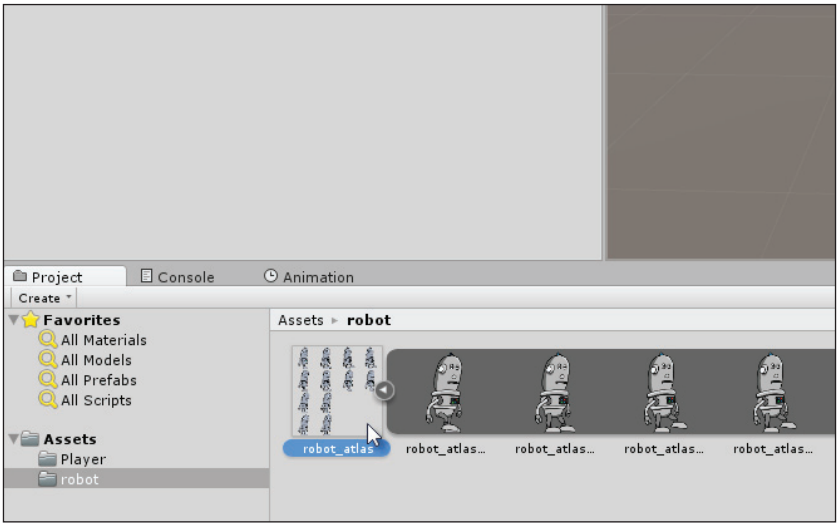

Рис. 2.11. Сгенерированные из текстуры атласа спрайты готовы к анимации

Анимация с помощью спрайтов

После того как вы импортировали набор объектов спрайтов (не имеет значения, как отдельные изображения или как текстуру атласа), вы готовы запустить анимацию с их участием. Сделать это очень просто, по крайней мере на начальном этапе. На панели проекта Unity выберите все спрайты, входящие в полную последовательность анимации, а затем перетащите их все вместе на панели иерархии сцены. Перетаскивание их на закладку сцены или закладку игры не сработает; их нужно перетащить именно на панель. Например, перетащите последовательность анимации бега игрока (из файлов этой книги) в сцену. Если вы сделаете это, Unity отобразит диалоговое окно сохранения **Save**, предлагая вам создать новую последовательность анимации актива (файл с расширением .anim). Задайте имя файла анимации (такое, например, как PlayerRun.anim) и щелкните по кнопке **Save**.

В сцене будет создан объект спрайта, он должен стать видимым на вкладках сцены и игры. Если это не так, убедитесь, что объект расположен в поле зрения камеры. И это все, что нужно для создания начальной последовательности анимации. Если вы протестируете вашу сцену, то ваш персонаж будет анимированным, вы должны уви-

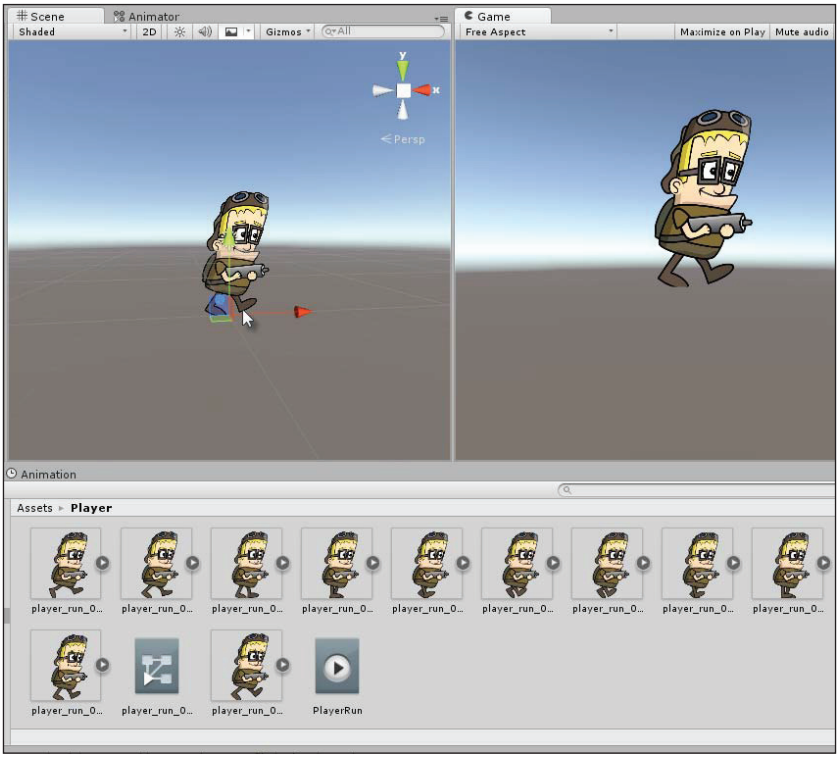

Рис. 2.12. Персонаж игры из набора кадров

деть все изображения последовательности. Это работает потому, что Unity выполнил ряд шагов конфигурирования за кулисами. В частности, Unity создало новый актив Animation Clip (.anim), который определяет ключевую последовательность кадров. Во-вторых, Unity создало контроллер Mecanim для запуска воспроизведения анимации при входе в уровень и для управления скоростью воспроизведения. В-третьих, Unity добавило компонент аниматора к объекту спрайта в сцене, чтобы связать объект с его данными по анимации. Однако созданная по умолчанию анимация, возможно, работает не так, как вы хотели. В последующих разделах мы увидим, как мы можем настроить анимацию.

Анимация спрайтами слишком медленная или слишком быстрая

Если ваша анимация спрайтами воспроизводится слишком быстро или слишком медленно, то вы должны будете отредактировать диаграмму спрайта инструмента Mecanim (инструмент Mecanim будет подробно описан в последующих главах). Скорость анимации для спрайтов может быть изменена быстро и легко. Для достижения этой цели выберите объект спрайта в сцене. В инспекторе объектов дважды щелкните на актив **Animation Controller** внутри слота **Controller** компонента **Animator**.

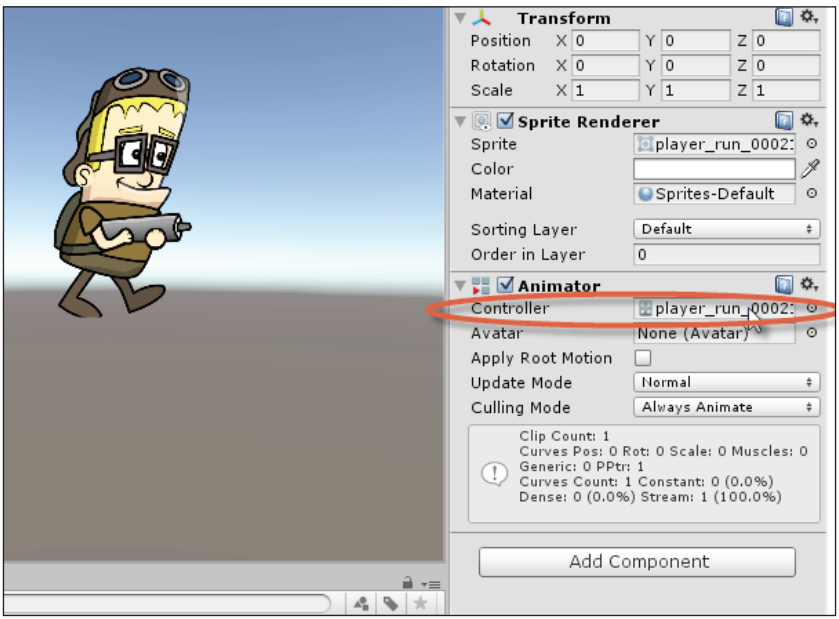

Рис. 2.13. Доступ к активу **Animation Controller** спрайта

Двойной щелчок по активу **Animation Controller** спрайта вызывает диаграмму инструмента **Mecanim** для спрайта. Из нее можно управлять скоростью анимации. Эта диаграмма содержит несколько узлов, связанных вместе.

В диаграмме нажмите на узел **PlayerRun** (по умолчанию), который представляет в диаграмме спрайт анимации, чтобы выбрать его и просмотреть его свойства в инспекторе объектов. В диаграмме спрайта для

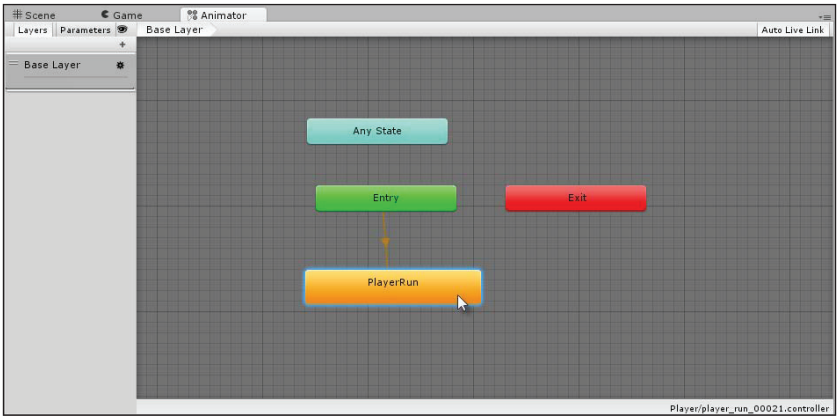

Рис. 2.14. Диаграмма инструмента **Mecanim** для анимированного спрайта

игрока спрайт графикой (входит в число сопроводительных файлов для этой книги) представляет собой полный цикл анимации ходьбы. Скорость анимации регулируется с помощью параметра **Speed**. Значение 0 – остановка, значение 1 – скорость по умолчанию, значение 0,5 – половинная скорость, 2 – двойная скорость и т. д. Если анимация слишком медленная, то увеличьте значение скорости, и если она слишком быстрая, то уменьшите значение скорости. Для моего игрока анимации персонажей я установил скорость до 2 – с удвоенной скоростью. После того как вы закончите, просто запустите игру, чтобы увидеть эффект.

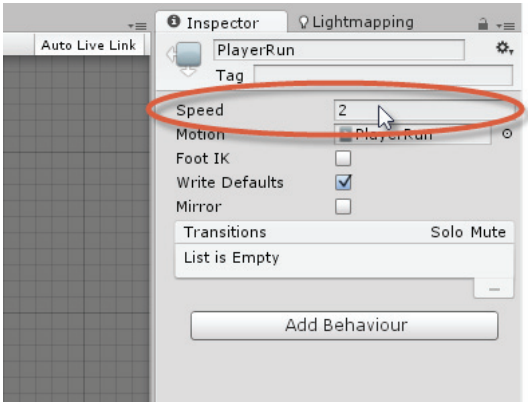

Рис. 2.15. Изменение скорости анимации

Анимация должна быть цикличной

По умолчанию спрайт анимации зациклен, то есть воспроизводится снова и снова без конца. Когда воспроизведение анимации завершает цикл, оно просто возвращается к началу, и происходит новое воспроизведение. Это именно то, что вам нужно в некоторых случаях, но не всегда. Иногда вам нужно воспроизвести анимацию только один раз, а затем остановить. Если это так, вам нужно получить доступ к данным анимации (они находятся внутри актива .anim) и откорректировать их свойства. Чтобы сделать это, выберите актив спрайта анимации в панели проекта. Активы анимации помечены иконкой **Play** и имеют имя, которое вы присвоили им при их создании.

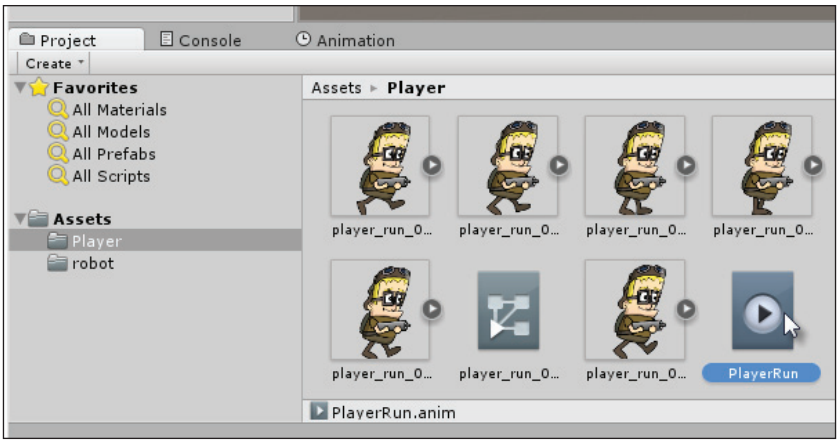

Рис. 2.16. Выбор актива анимации

После выбора актива сбросьте флажок **Loop Time** в инспекторе объектов (рис. 2.17). Теперь запустите игру, и анимация будет воспроизведена только один раз.

Кадры воспроизводятся в неправильном порядке

Если ваш спрайт анимации имеет много кадров, то, возможно, Unity расположит их при генерации анимации в неправильном порядке, в результате чего некоторые кадры станут появляться раньше или позже нужного момента. Если это произойдет, вы должны будете отредактировать сами данные анимации в окне **Animation**. Это окно можно получить, перейдя к **Window → Animation** в главном меню (рис. 2.18).

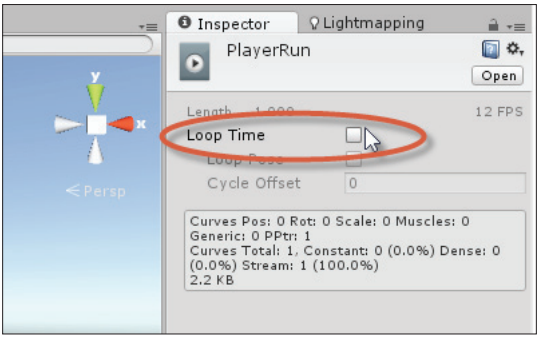

Рис. 2.17. Сброс флажка зацикливания при однократном проигрывании анимации

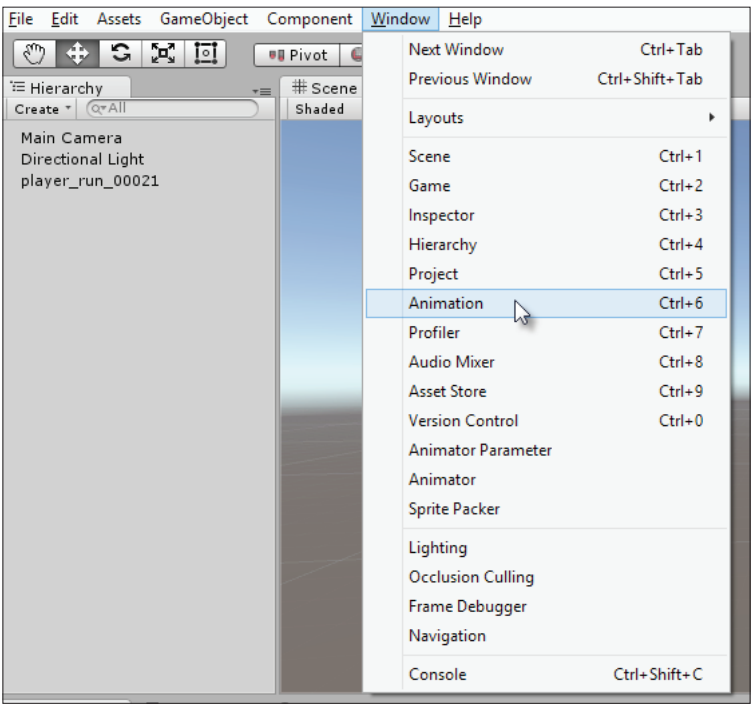

Рис. 2.18. Доступ к окну **Animation**

При открытом окне **Animation** выберите объект спрайта в сцене, и его анимационные данные автоматически будут отображены на временной шкале. График содержит в себе весь период анимации, от начала и до конца. Ромбовидные символы, равномерно распределенные по шкале, представляют собой ключевые кадры, в которых сменяются изображения спрайтов. Вы можете выбрать конкретный ключевой кадр на временной шкале, щелкнув по нему. Затем вы можете просмотреть его спрайтовые свойства в инспекторе объектов. На рис. 2.19 для примера показано поле **Sprite** со значением **player_run_00013** для момента времени, равного **0:06**. Поле **Sprite** выделено красным в инспекторе объектов, чтобы подчеркнуть, что поле анимированное и изменяется с течением времени.

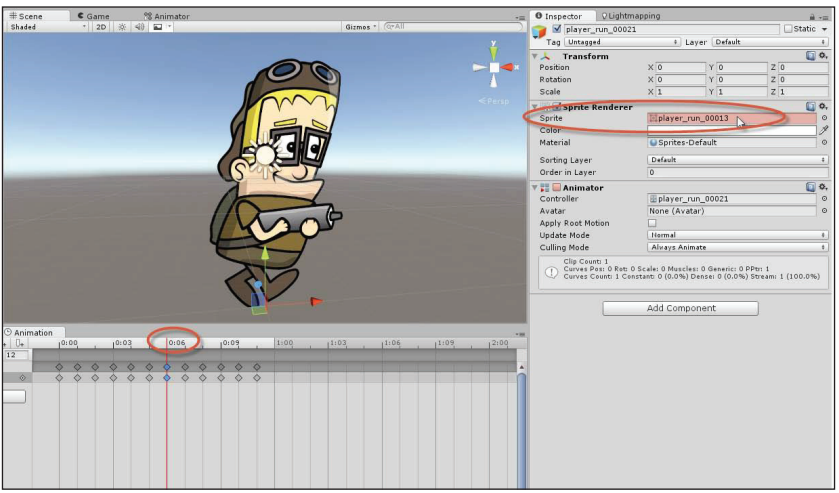

Рис. 2.19. Просмотр ключевых кадров для анимации спрайтами

Выбрав ключевой кадр графика, в котором находится то изображение, которое не должно быть показано в это время, вы можете это легко исправить, нажав на поле **Sprite** в инспекторе объектов и выбрав новый спрайт в браузере спрайтов. Unity автоматически внесет нужные изменения, и выбранный вами спрайт будет отображаться в этом ключевом кадре.

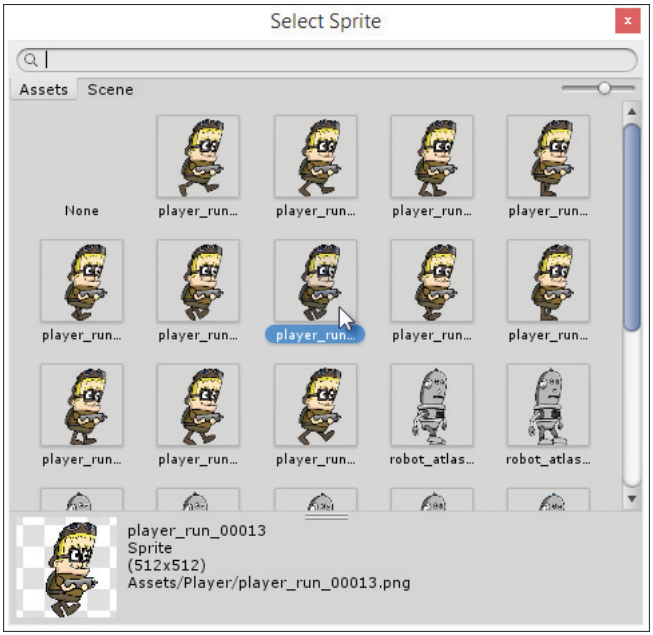

Рис. 2.20. Использование браузера спрайтов для корректировки кадров анимации

Итоги

В этой главе рассмотрен обширный набор 2D-функций Unity, используемый для создания анимации спрайтами. Спрайты могут быть импортированы как кадр за кадром из отдельных файлов, так и как текстура атласа, содержащая набор кадров. В любом случае, кадры легко могут быть собраны в анимационную последовательность, а инструменты создания анимации Unity (такие как система Mecanim и окно анимации) могут быть использованы для решения наиболее распространенных проблем, таких как настройка скорости анимации, зацикливание и перепутанные кадры. Используя активы, описанные в этой главе (в том числе актив персонажа игрока), вы теперь сможете быстро и эффективно создавать анимированный персонаж игрока. Кроме того, вы можете использовать скрипт Mover, созданный в пре-

дыдущей главе, для перемещения анимированного спрайтами объекта вдоль сцены, одновременно воспроизводя анимацию бега, что придаст процессу бега законченность. Завершенный проект, который делает все это, находится в приложенных к книге файлах в папке Chapter02\AnimatedChar. В следующей главе окно анимации Unity, а также и другие инструменты будут рассмотрены более детально.

Глава 3

Встроенная анимация

Приложение Unity не рассматривается как инструмент для создания контента, а скорее как композиционный инструмент. Unity знают как **игровой движок** (game engine), в котором предварительно созданные активы (активы, сделанные в других приложениях) импортируются и настраиваются при создании игр. Однако, несмотря на то что это в значительной степени верно, Unity предлагают и широкий спектр инструментов создания активов, особенно для анимации. Эти инструменты называются встроенными функциями анимации. Они включают в себя следующее:

- **Редактор анимации Unity** (Unity animation editor) для анимации твердых тел, например для создания открывающейся двери, летающей камеры, платформы лифта и прочего;
- **Система частиц Сюрикэн** (Shuriken particle system) для создания дождя, снега, фейерверков, искр и в другой нематериальной анимации со множеством движущихся частиц.

Оба этих инструмента будут рассмотрены в этой главе.

Окно анимации – создание полета

Окно анимации (Animation window) – это полнофункциональный редактор анимации, предназначенный для создания предварительного сценария анимации, содержащего в себе набор числовых значений свойств объекта игры, таких как положение, поворот и масштаб, в ключевых кадрах. Короче говоря, он позволяет выполнять анимацию объекта игры во времени, храня данные об анимации в виде отдельного независимого актива, который называется **клипом анимации (animation clip)**, в панели проекта. Если вы хотите создать открывающиеся и закрывающиеся поворачивающиеся на петлях двери, движущуюся вверх и вниз платформу лифта, вращающиеся полы, едущие автомобили, летящие вражеские космические корабли или что-то еще подобное, то окно **Анимация** – это именно то, что вам нужно.

Для демонстрации работы окна анимации мы создадим летающую над ландшафтом камеру. Важно отметить, что набор инструментов, который мы продемонстрируем, универсален и может быть использован не только при создании летающей камеры. Вы можете либо использовать свой собственный ландшафт, если он у вас есть, либо открыть начальный проект для этого раздела, который входит в сопровождающие эту книгу файлы и находится в папке Chapter03/FlyThroughStart. Если у вас есть ландшафт и камера, то вы готовы идти дальше.

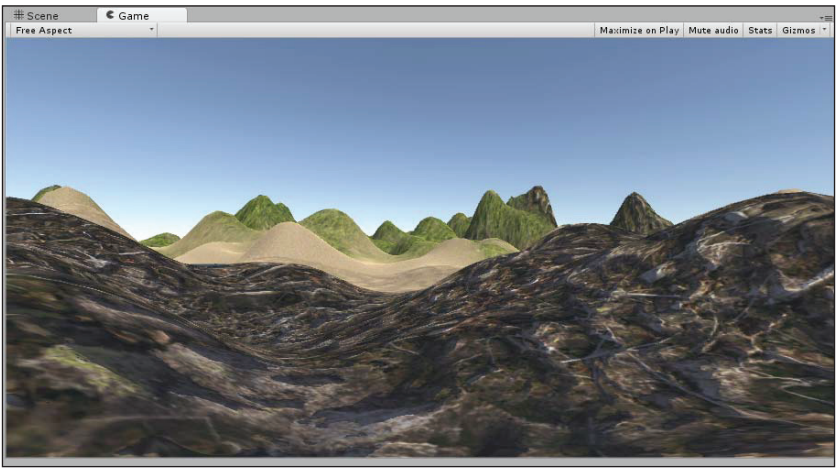

Рис. 3.1. Создание ландшафта для полета камеры

Законченный проект для этого раздела можно найти в прилагаемых к этой книге файлах в папке Chapter03/FlyThroughEnd.

Чтобы открыть окно анимации для создания ключевых кадров анимации, перейдите в **Window → Animation** из главного меню (или нажмите **Ctrl+6** на клавиатуре), рис. 3.2. Удостоверьтесь, что вы выбрали **Animation**, а *не* **Animator**.

С окном анимации лучше всего работать в горизонтальном выравнивании, поэтому я, как правило, закрепляю окно в нижней части интерфейса, под вкладками игры и сцены. Чтобы сделать это, просто перетащите окно анимации на заголовки вкладок панели проекта или области консоли.

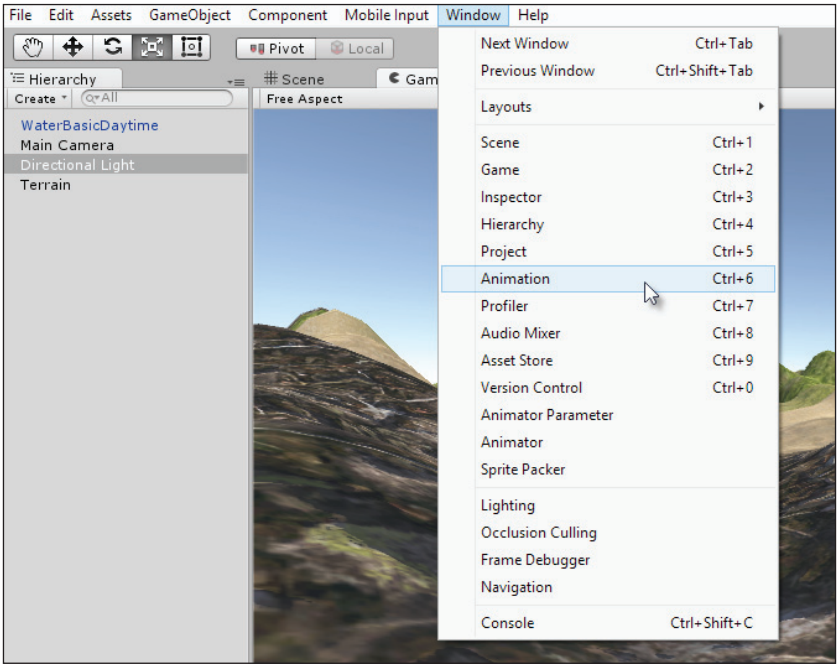

Рис. 3.2. Доступ к окну анимации из основного меню

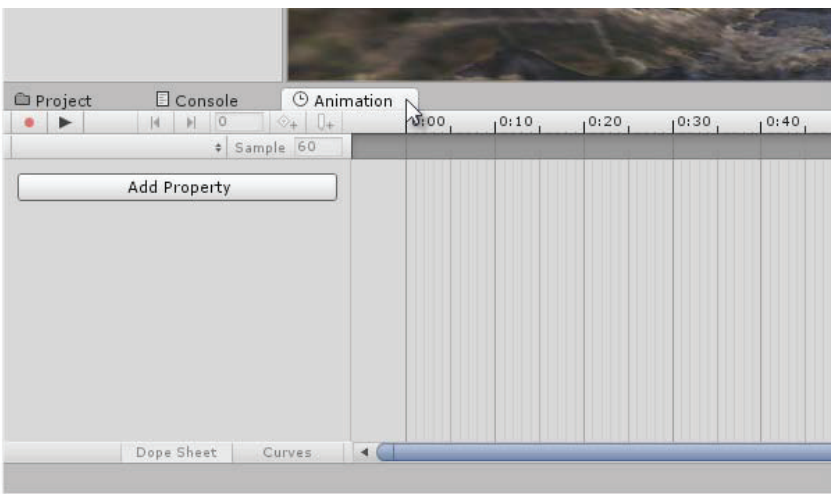

Рис. 3.3. Закрепление окна анимации в нижней части интерфейса

Для создания ключевых кадров анимации объекта игры выберите объект на сцене или по его имени в панели иерархии. Затем нажмите на кнопку **Add Property** в окне анимации. Для создания анимированной летающей камеры выберите камеру сцены, а затем нажмите **Add Property** из окна анимации. После того как вы нажали на эту кнопку, Unity выведет диалоговое окно для указания месторасположения в папке проекта данных анимации (клипа анимации), задания его имени и подтверждения. Задайте имя анимации CameraFlyAnim.anim и выберите папку **Assets**, не сохраняя этот файл в корневую папку проекта.

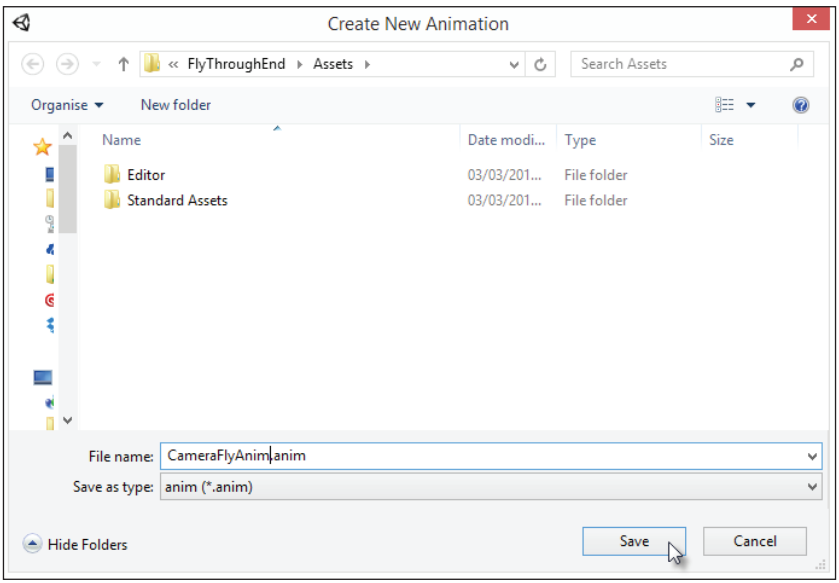

Рис. 3.4. Создание актива нового анимационного клипа

При создании нового анимационного клипа Unity произведет несколько действий за кулисами. Во-первых, Unity создаст два актива: актив анимационного клипа (содержащий данные всех ключевые кадров) и актив контроллера анимации (Animation Controller), являющийся привязкой к системе Mecanim. Кроме того, к объекту камеры будет добавлен компонент **Аниматор** (Animator), который связан с активом контроллера анимации, который, в свою очередь, отвечает за воспроизведение данных клипа анимации автоматически при загрузке сцены. Подробнее система Mecanim будет рассмотрена

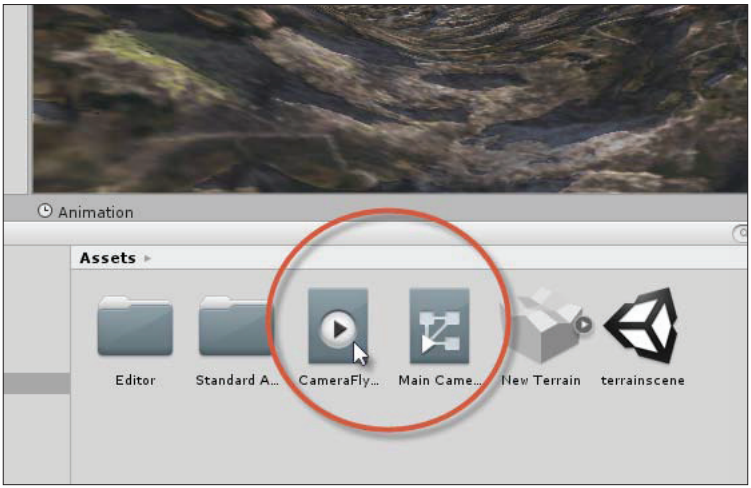

Рис. 3.5. Создание нового анимационного клипа с авгогенерацией контроллера анимации

в следующей главе. Сейчас достаточно знать только, что контроллер анимации – это активная сила, которая будет инициировать воспроизведение анимации.

После создания актива анимационного клипа мы должны определить анимацию. Unity может сохранять данные ключевых кадров для любого числового свойства. Другими словами, любое поле объекта, имеющее числовое значение, например положение или вращение, можно анимировать или изменять во времени. Чтобы анимировать полет камеры, мы должны будем определить изменения положения и вращения камеры во времени. Чтобы сделать это, нажмите на кнопку **Add Property** в окне анимации и из появившегося контекстного меню выберите каналы для анимации. В частности, нажмите кнопку **+** и перейдите к **Transform**. Затем вы можете перейти либо к каналу положения (**Position**), либо к каналу вращения (**Rotation**). При этом в окне анимации будут добавлены два канала (положение и вращение), которые теперь могут быть анимированы. Канал – это просто свойство, значения которого мы можем привязать к ключевым кадрам.

По умолчанию для обоих каналов положения и поворота автоматически генерируются два ключевых кадра: один кадр в начале анимации (время: 0 секунд) и один в конце (время: 1 секунда). Графически

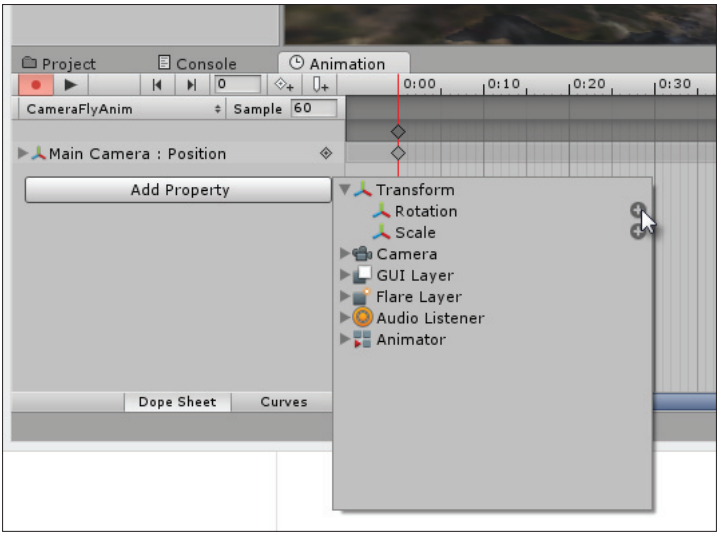

Рис. 3.6. Добавление каналов анимации для положения и вращения

ключевые кадры представлены серыми ромбовидными значками на временной шкале. Вы можете перетащить мышкой ползунок времени проигрывателя анимации в полосе единиц времени (в верхней части шкалы времени), поместив ее на любой момент времени для предварительного просмотра анимации. По умолчанию, однако, начальный и конечный ключевые кадры для камеры идентичны, что означает, что ничего не будет меняться при анимации.

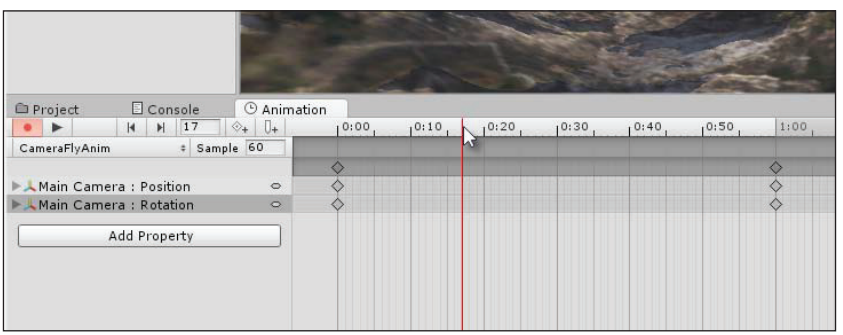

Рис. 3.7. Установка головки проигрывания анимации для просмотра кадров анимации

Вы можете вручную создать ключевой кадр для записи состояния любого канала (или всех каналов) для камеры в любой момент в диапазоне анимации. Это можно сделать, щелкнув по каналу для записи в левой колонке окна анимации (**Список каналов**). Затем переместите ползунок времени (щелкните и перетащите) на соответствующее время и, наконец, нажмите на кнопку добавления ключевого кадра **Add Key Frame** из панели инструментов окна анимации. Это приведет к вставке нового ключевого кадра в шкале времени на том месте, где находится ползунок времени. Запомните, что несколько ключевых кадров можно добавить сразу для нескольких каналов, нажав клавишу **Shift** и выбирая каналы в списке каналов; то есть удерживая нажатой клавишу **Shift** на клавиатуре при щелчке по каналам. Вы можете перетаскивать ромбовидные значки ключевых кадров на шкале времени, меняя для них время, а также продублировать их с помощью **Ctrl**+**C** и **Ctrl**+**V**. И еще вы можете удалить выбранные ключевые кадры с помощью клавиш **Backspace** или **Delete**.

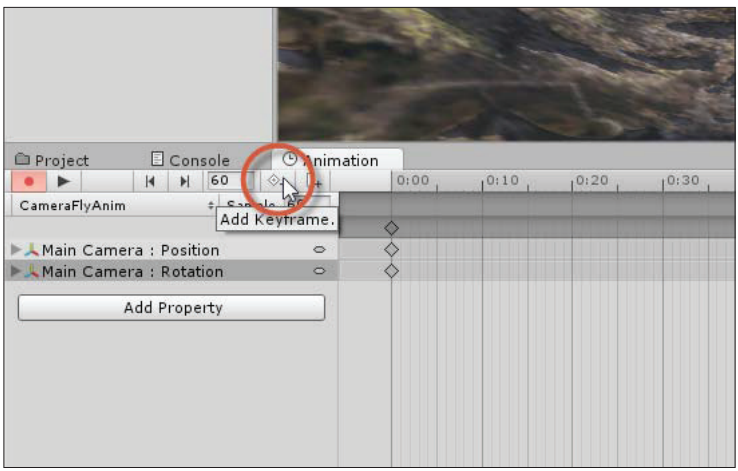

Рис. 3.8. Создание ключевых кадров вручную для выбранных каналов с помощью кнопки **Add Keyframe**

Создание ключевых кадров вручную, как мы видели, является легитимным методом определения анимации, но проще воспользоваться возможностью, предоставляемой Unity, – генерировать ключевые кадры автоматически, основываясь на движениях объектов в сцене. Создание полета камеры при использовании этого метода выполня-

ется просто: поместите ползунок анимации в окне анимации в положение 0, а затем переместите и поверните объект камеры в сцене в ее желаемое исходное положение и ориентацию. Unity автоматически запишет положение и поворот камеры для каналов на выбранный вами момент времени. Затем перетащите ползунок на момент времени 0:15 в окне анимации, переместите и поверните камеру в сцене в новое положение и ориентацию для этого момента времени. Unity сгенерирует новый набор ключевых кадров на вновь выбранный момент времени. Повторите этот процесс генерации ключей кадров, определяющих положение и поворот для камеры, для каждого из моментов времени: 0:30, 0:45, 1:00.

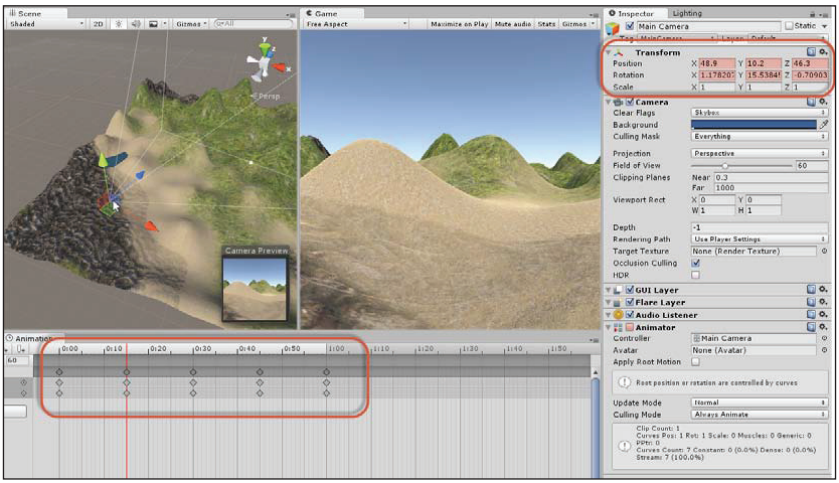

Рис. 3.9. Генерация ключевых кадров для анимации полета камеры

Попробуйте сделать первые и последние ключевые кадры одинаковыми (с помощью комбинаций клавиш **Ctrl**+**C** и **Ctrl**+**V**), копирования и вставки первого ключевого кадра в момент окончания, время 1:00. Это значит, что анимация будет заканчиваться так же, как и началась, что позволит зациклить ее, то есть она станет анимацией вида, который можно воспроизводить снова и снова в цикле. Когда анимация придет в то состояние, к которому вы стремились, протестируйте ее. Нажмите кнопку **Play** в панели инструментов Unity, и анимация из-за компонента **Аниматор** начнет воспроизводиться автоматиче-

ски. Если анимация воспроизводится слишком быстро или слишком медленно, или не повторяется многократно, обратитесь к предыдущей главе (*глава 2* «*Анимация спрайтами*»), чтобы проконтролировать скорость анимации и зацикленность. Поздравляю! Вы только что создали анимацию полета камеры.

Анимация нескольких объектов сразу

Анимация полета камеры, созданная в предыдущем разделе, анимировала только один объект, а именно камеру. Рано или поздно наступит время, когда вы должны будете анимировать сразу несколько объектов, движущихся синхронно, таких, например, как вращающиеся колеса едущего автомобиля. В этом случае автомобиль движется вперед, колеса вращаются и одновременно перемещаются вперед как часть автомобиля. Для создания такого рода составной анимации вы должны первым делом соединить все составляющие объекты вместе в один родительский объект. В примере для автомобиля все колеса, шасси и двигатель будут принадлежать одному родительскому объекту автомобиля с именем **Car**.

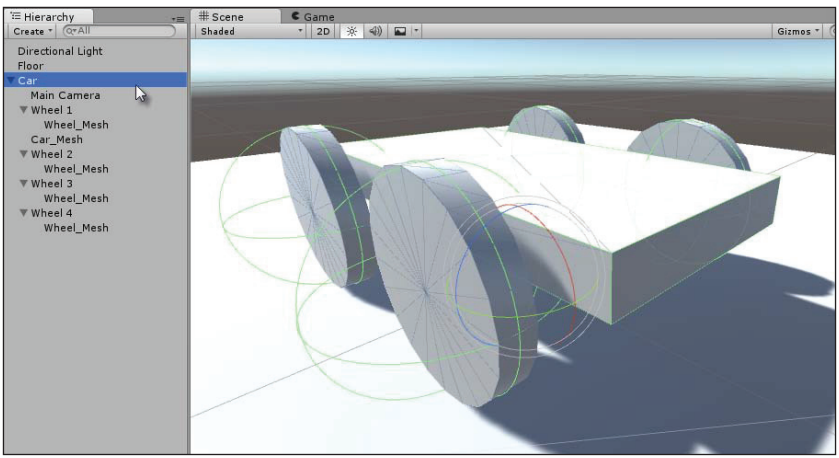

Рис. 3.10. Подготовка составного объекта для анимации

Проект автомобиля, пример для этого раздела, можно найти в прилагаемых к этой книге файлах в папке Chapter03/CarAnim.

Когда анимационный клип присоединяется к родительскому объекту, а не к дочернему объекту, вы можете анимировать и каналы любого (или всех) его дочернего объекта, что дает вам возможность управлять анимацией нескольких объектов сразу. В окне редактора анимации нажмите на кнопку **Add Property** и выберите добавляемые каналы из появившегося меню. Меню включает в себя и все каналы дочерних объектов.

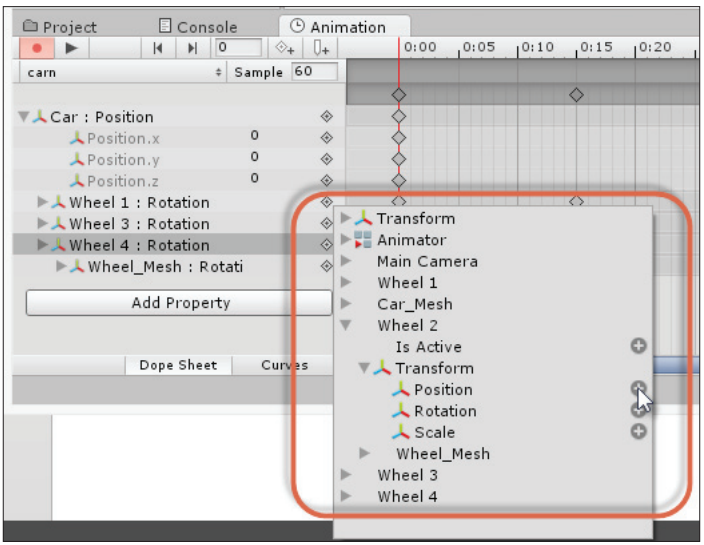

Рис. 3.11. Добавление каналов анимации для дочерних объектов

Вызовы функций из анимации

Как правило, анимацию необходимо связать со скриптами и с логикой игры. В частности, при достижении анимацией определенного кадра или момента времени, или при старте и завершении анимации, вы захотите выполнить некоторые другие действия, возможно, даже с другим объектом игры. В анимации полета камеры мы хотим, например, показать на экране сообщение «Добро пожаловать!» в момент завершения полета. Есть много способов реализации этого. Один из способов – это запуск функции скрипта, когда анимация достигнет определенного кадра. Чтобы создать этот вид связи между анимацией и скриптом, мы должны использовать события анимации.

Законченный проект Unity для событий анимации можно найти в прилагаемых к этой книге файлах в папке Chapter03/AnimationEvents.

Для начала работы с вызовом функций из анимации создайте новый файл скрипта C# или откройте для редактирования существующий файл. Добавьте функцию с любым именем, которая возвращает void и/или не имеет аргументов или принимает строку, вещественное число, целое число или GameObject. Ниже приведен пример кода, функция в котором не содержит параметров и при ее вызове отображает канву объекта пользовательского интерфейса GUI для вывода приветствия при завершении полета камеры. Файл скрипта необходимо прикрепить к объекту камеры в сцене.

Более подробную информацию о событиях анимации можно найти в официальной документации Unity по адресу docs.unity3d.com/Manual/animeditor-AnimationEvents.html.

```
//-------------------------------------------
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
//-------------------------------------------
public class ShowMessage : MonoBehaviour
{
  //Reference to GUI canvas to show on event
  public Canvas UICanvas = null;

  //Function to be called from an Animation Event
  void ShowWelcomeMessage()
  {
    //Enable canvas to show message
    UICanvas.gameObject.SetActive(true);
  }
}
//-------------------------------------------
```

Чтобы создать связь между анимацией и функцией `ShowWelcomeMessage`, откройте окно анимации и щелкните правой кнопкой мыши по серой панели над завершающим ключевым кадром момента времени 1:00, но ниже шкалы времени. При этом появится контекстное меню с опцией для создания события анимации для выбранного ключевого кадра, как это показано на рис. 3.12.

Щелкните по опции **Add Animation Event** из контекстного меню. Это приведет к появлению диалогового окна редактирования события анимации **Edit Animation Event** с элементом управления в виде выпадающего списка. Он служит для выбора вызываемой в момент

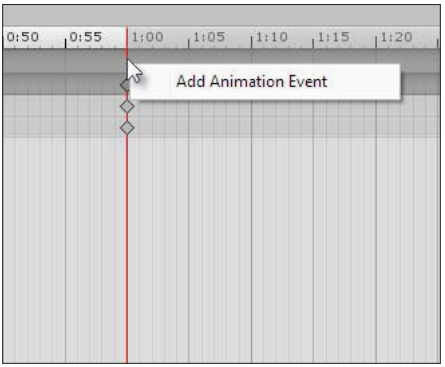

Рис. 3.12. Добавление
к анимации события

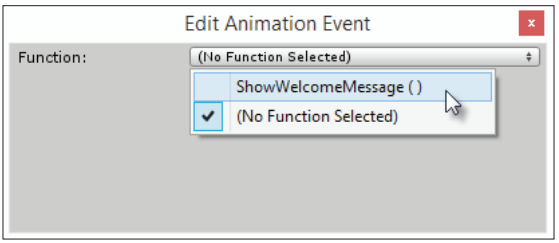

Рис. 3.13. Выбор функции для вызова
при наступлении события анимации

наступления события функции. Если нужная вам функция не отображается в списке в качестве опции, убедитесь, что все необходимые файлы скриптов прикреплены к анимированному объекту игры.

Закройте диалоговое окно редактирования события анимации **Edit Animation Event**, и вы увидите белый маркер на шкале времени, отмечающий момент времени вызова функции. Маркер может быть отредактирован, удален и перемещен так же, как и обычный ключевой кадр.

Вот и все! Теперь событие анимации будет срабатывать автоматически при воспроизведении анимации объекта, когда ползунок времени коснется последнего кадра. Нажмите на кнопку **Play** на панели инструментов Unity и запустите тестирование проекта.

Не забудьте, что общедоступная переменная UI Canvas скрипта `ShowMessage` должна быть связана с реальным объектом канвы пользовательского интерфейса. Соответствующие файлы расположены в папке Chapter3\Ani-

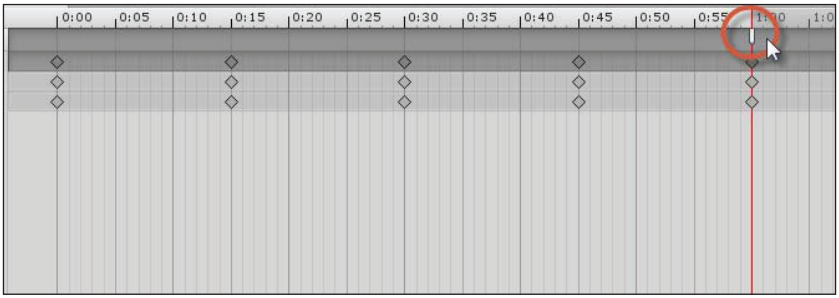

Рис. 3.14. Привязка события анимации на шкале времени анимации

Рис. 3.15. Показ приветственного сообщения при завершении анимации

mationEvents. Более подробную информацию о канве пользовательского интерфейса UI Canvas можно найти в онлайн-документации Unity на http://docs.unity3d.com/ScriptReference/Canvas.html.

Системы частиц

Системы частиц прежде всего предназначены для создания спецэффектов и анимации, где множество частей или предметов должны двигаться вместе, единой сплоченной массой, например дождь, снег, сияние, облака светящейся пыли, стаи птиц, рои пчел и многое другое. Они также используются для имитации нематериальных явлений, таких как световые лучи, пыль, призраки, голограммы и прочее.

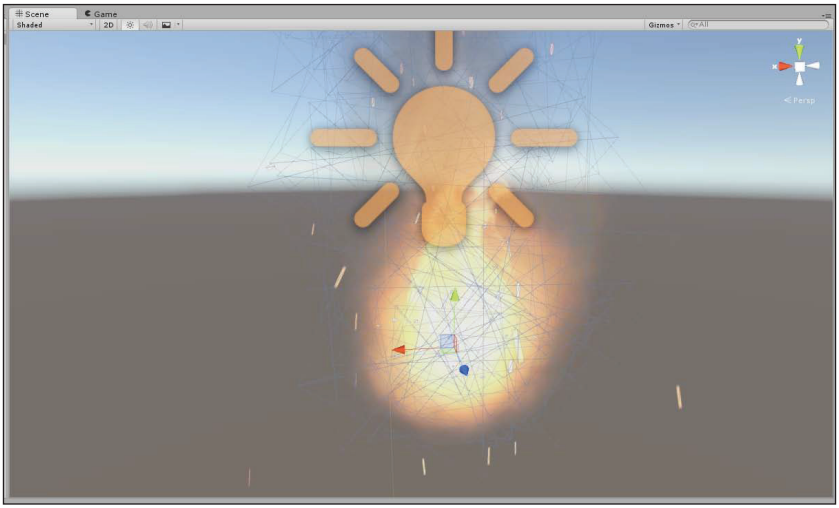

Рис. 3.16. Система частиц огня в действии

Unity 5 обладает широким диапазоном предварительно подготовленных систем частиц, которые достаточно просто поместить в сцену для создания наиболее распространенных эффектов, таких как взрыв, огонь, пар или дым. Для получения доступа к этим эффектам импор-

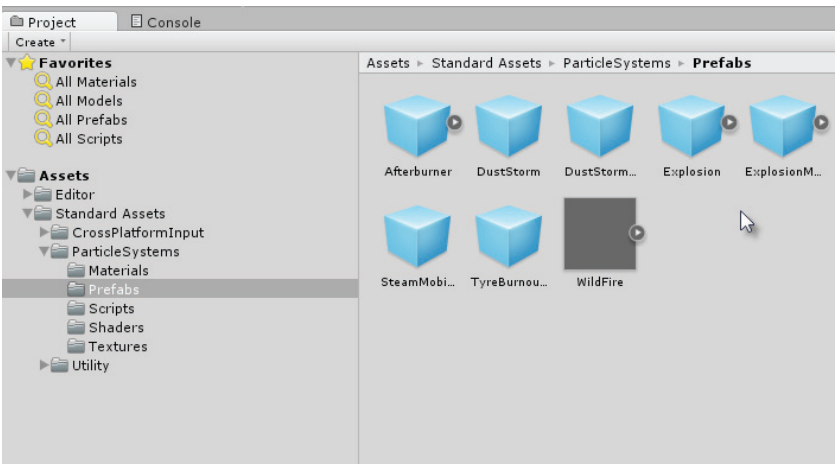

Рис. 3.17. Unity 5 поставляется со множеством предварительно подготовленных систем частиц

тируйте пакет активов системы частиц в проект, выполнив **Assets → Import Package → ParticleSystems** из меню приложения. После выполнения такого импорта все предварительно подготовленные системы будут доступны в **Assets → ParticleSystems → Prefabs** панели проекта Unity.

Несмотря на различия между ними, все системы частиц имеют три общие черты. Во-первых, это наличие **эмиттера** (emitter), который, как поливочный шланг или портал, испускает или извергает частицы. Во-вторых, сами частицы – мелкие предметы или кусочки, генерируемые эмиттером. Для дождя частицами служат капли, для снега – снежинки, для стаи птиц – отдельные птицы и т. д. И наконец, в-третьих, испускаемые частицы имеют срок службы – период времени, в течение которого они существуют, а также траекторию или скорость, которая определяет, что произойдет с частицами в течение их срока службы (как они будут двигаться, как быстро перемещаться и т. д.). Таким образом, для того чтобы создать систему частиц, надо определить, как эти три элемента будут работать вместе. В следующем разделе мы рассмотрим проект создания с нуля системы частиц с помощью системы Сюрикэн (Shuriken System).

Система частиц для имитации светлячков

Часто встречающаяся в играх система частиц, особенно в ролевых играх (RPG) на темы фэнтази, – это система частиц для имитации светлячков. Если вы бредете по темному лесу в ночное время или пробираетесь через предательские болота в тумане, вокруг вас будут кружиться маленькие, яркие существа. Они не несут в себе никакой функциональной нагрузки с точки зрения логики игры, то есть не наносят вам вреда и не приносят пользы. Их предназначение, скорее, косметическое и эмоциональное. Они передают атмосферу и настрой. Группа ярких светлячков перемещается медленно и размеренно, каждый летит со своей индивидуальной скоростью (рис. 3.18).

Законченный проект Unity для этой системы частиц можно найти в прилагаемых к этой книге файлах в папке Chapter03/ParticleSystems.

Стандартный пакет систем частиц Unity не включает в себя эффекта имитации светлячков, поэтому мы создадим его сами. Для этого мы используем систему Сюрикэн Unity. Начнем с создания новой системы частиц, выполнив **GameObject → ParticleSystem** из меню приложения (рис. 3.19). Выбрав ее, создадим в сцене новый объект системы частиц по умолчанию. При ее выборе система частиц автоматически начнет функционировать в окне сцены.

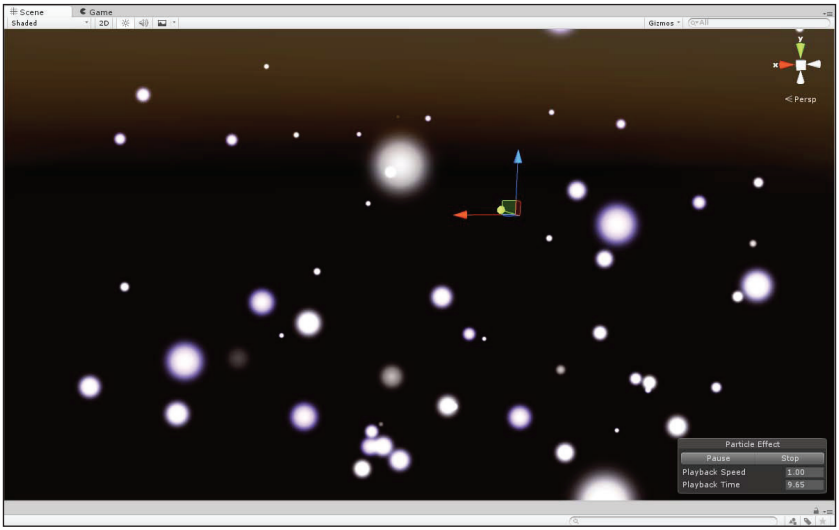

Рис. 3.18. Создание системы частиц имитации светлячков

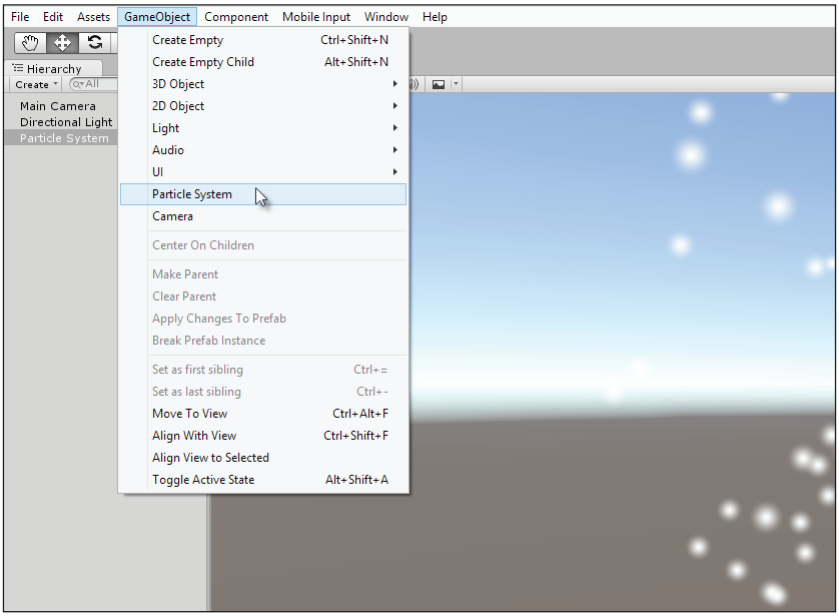

Рис. 3.19. Создание системы частиц с помощью Сюрикэн

Для этого проекта нам понадобится освещение сцены для ночного времени или освещение сразу после заката солнца. Добиться этого просто, создайте новую сцену и поверните источник по умолчанию направленного света, направив его вверх. Скайбокс по умолчанию Unity 5 автоматически отреагирует на переориентацию света.

Общие свойства системы частиц

Когда система частиц выбрана в сцене, вы можете просмотреть ее свойства в редакторе Сюрикэн, который доступен как компонент системы частиц в инспекторе объектов. Этот редактор делится на несколько различных категорий. Основные общие свойства управляют общими атрибутами и поведением выбранной системы частиц.

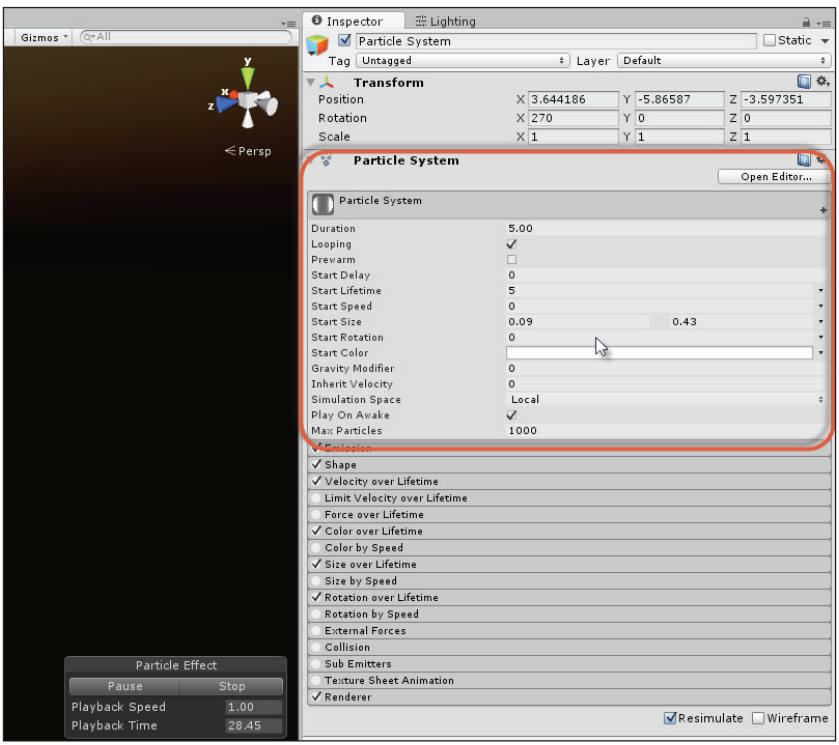

Рис. 3.20. Настройка общих свойств системы частиц светлячков

Значение поля длительности **Duration** определяет общую продолжительность анимации системы частиц, прежде чем она начнется сначала и зациклится. Для светлячка значение поля начальной скорости **Start Speed** установлено в 0, так как управление скоростью и перемещением будет перехвачено и контролироваться с помощью более продвинутых настроек в редакторе Сюрикэн. Значение поля начального размера **Start Size** управляет размером сгенерированных частиц. Изменим режим задания значения этому полю с выбора одной постоянной величины (все частицы одного размера) на случайное значение для каждой частицы, в диапазоне между минимальным и максимальным значениями (с учетом этого каждая частица получит свой размер). При этом светлячки будут разного размера. Для доступа к параметру смены режима нажмите на стрелку вниз справа от поля начального размера **Start Size** и выберите опцию случайной величины между двумя константами **Random Between Two Constants** из контекстного меню.

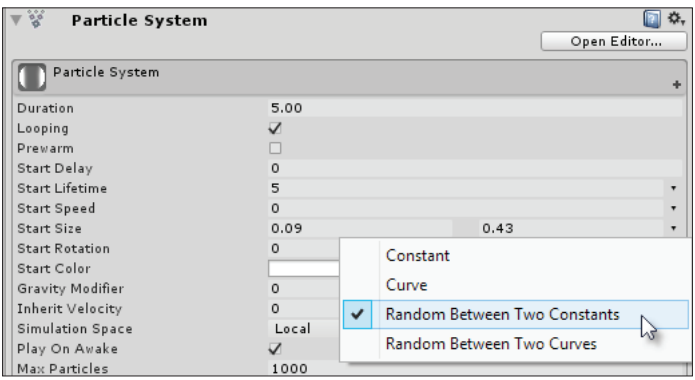

Рис. 3.21. Выбор случайной величины между минимальным и максимальным значениями

Форма эмиттера и скорость эмиссии

Как уже упоминалось, каждая система частиц имеет эмиттер – место выброса генерируемых частиц. Сам эмиттер имеет форму. Эта форма может быть плоскостью, точкой, кубом, шаром или обычным мешем, и она представляет собой поверхность или тело, внутри которого генерируются частицы. Точное место на поверхности или внутри тела случайным образом выбирается для каждой частицы. Для управления формой эмиттера раскройте закладку **Shape** в редакторе Сюрикэн и убедитесь, что флажок в поле **Shape** установлен. Для систе-

мы имитации светлячков все частицы должны быть сгенерированы в пределах сферы. Я установил радиус сферы 2,5 (в метрах). Точность задания значения поля радиуса **Radius** не существенна, но это значение должно быть выбрано так, чтобы система частиц выглядела правильно и подходила для сцены.

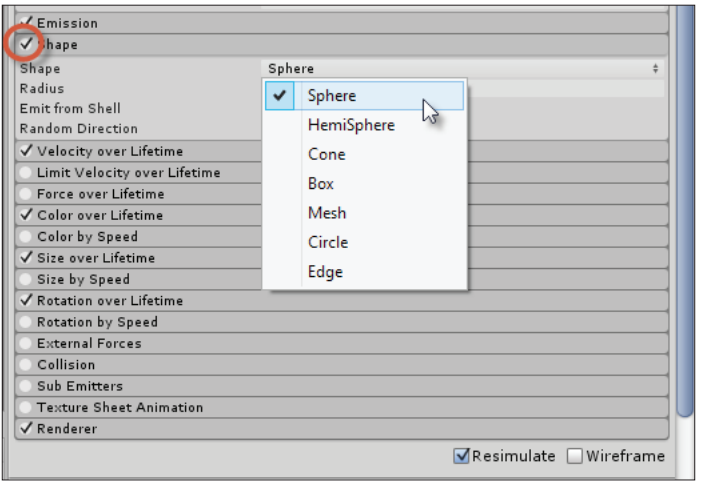

Рис. 3.22. Установка формы эмиттера

Затем раскроем секцию эмиссии Emission редактора Сюрикэн. Она отвечает за частоту или скорость генерации новых частиц из эмиттера в секунду. При создании светлячков не нужен плотный или насыщенный выброс частиц. Я выбрал для поля **Emission Rate** значение **18,5**.

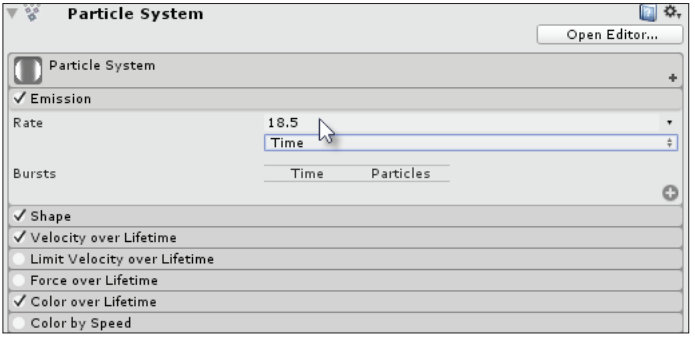

Рис. 3.23. Установка скорости эмиссии

Внешний вид частиц

Каждая частица в системе одинаково рендерится и имеет одинаковый внешний вид. Параметры, управляющие внешним видом частиц, находятся в секции рендерера **Renderer** редактора Сюрикэн. Поле **Render Mode** определяет, как визуализируются частицы, режимов всего два: режим билборда **Billboard** и режим мешей **Mesh**. Частицы, визуализируемые в виде билборда, представляют собой меш квадрата с нанесенной на него текстурой. Частицы, визуализируемые в виде меша, имеют более сложную форму, и вы можете использовать для них любой меш, импортированный в проект. Кроме настройки режима рендера, частицы нуждаются в материале, который будет назначен мешу. Для частиц вы можете использовать любой материал, но в системе имитации светлячков мы будем использовать предварительно подготовленные материалы для систем частиц, поставляемые в составе активов Unity (которые можно импортировать в проект, выполнив **Assets → Import Package → ParticleSystems** из меню приложения). Я выбрал материал **ParticleAfterburner**. Вы можете просто перетащить этот материал в слот материала **Material** в инспекторе объектов.

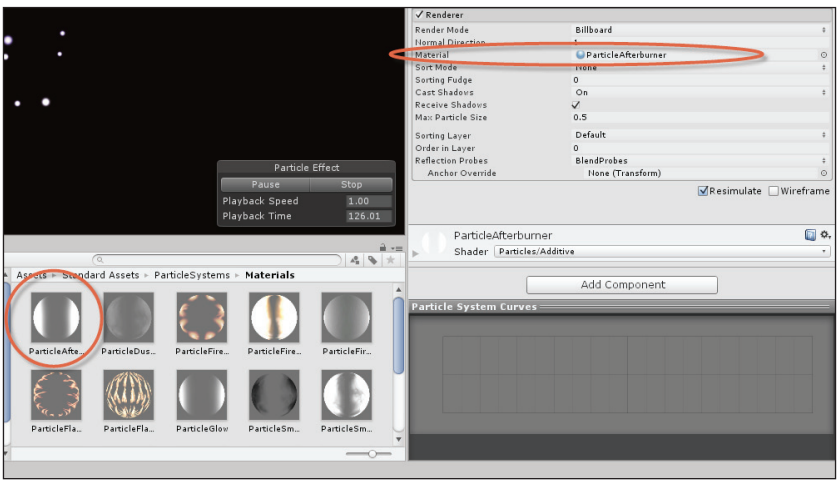

Рис. 3.24. Установка рендера частиц

Скорость частиц

Одной из наиболее важных особенностей поведения системы частиц имитации светлячков является то, что каждая частица должна двигаться по своей траектории. Это определяется тем, что каждая частица – это отдельный и независимый светлячок, которого связывает с основной массой светлячков лишь близость расположения. Скорость частицы в период ее существования определяется в секции **Velocity over Lifetime**. Если вы раскроете эту секцию в инспекторе объектов, то увидите три поля для постоянных значений для осей *x*, *y* и *z*. Вместе они определяют трехкомпонентный вектор, задающий направление и скорость перемещения всех частиц. Эти три постоянных значения не подходят для системы имитации светлячков, потому что они определяют одинаковое направление полета для всех светлячков. Для внесения элемента случайности в направление полета светлячков нам нужно изменить режим скорости с **Constant** на **Random Between Two Curves**. Для внесения такого изменения нажмите на направленную вниз стрелку в правом верхнем углу секции **Velocity over Lifetime**.

Если опции исчезли, то убедитесь, что флажок **Velocity over Lifetime** включен.

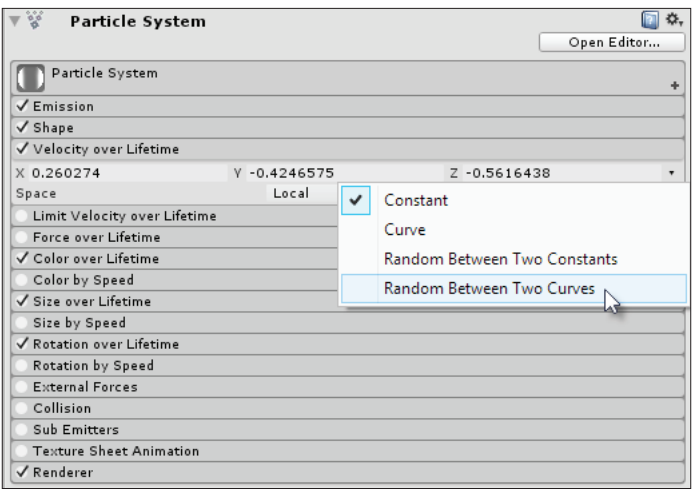

Рис. 3.25. Подготовка полей секции **Velocity over Lifetime**

При использовании двух кривых целый диапазон минимальных и максимальных значений скорости может быть установлен для каждой из частиц, что определит ее движение на все время существования. Чтобы задать кривые, щелкните по одному из полей осей в инспекторе, что приведет к появлению ее в виде графика в панели предварительного просмотра в инспекторе объектов. Просмотрите все оси по одной, щелкая по ним. В панели предварительного просмотра дважды щелкните по графику линии и затем, щелкнув по точке графика, перетащите ее для изменения кривой. Горизонтальная ось относится ко времени (левая сторона соответствует рождению частиц, а правая – уничтожению частиц), а вертикальная ось относится к величине скорости на выбранный момент времени. Идея заключается в том, чтобы создать вариации кривой для получения соответствующих изменений скорости частицы на протяжении всего ее существования. Точки вдоль кривой могут быть созданы просто двойным щелчком в том месте, куда вы их хотите добавить.

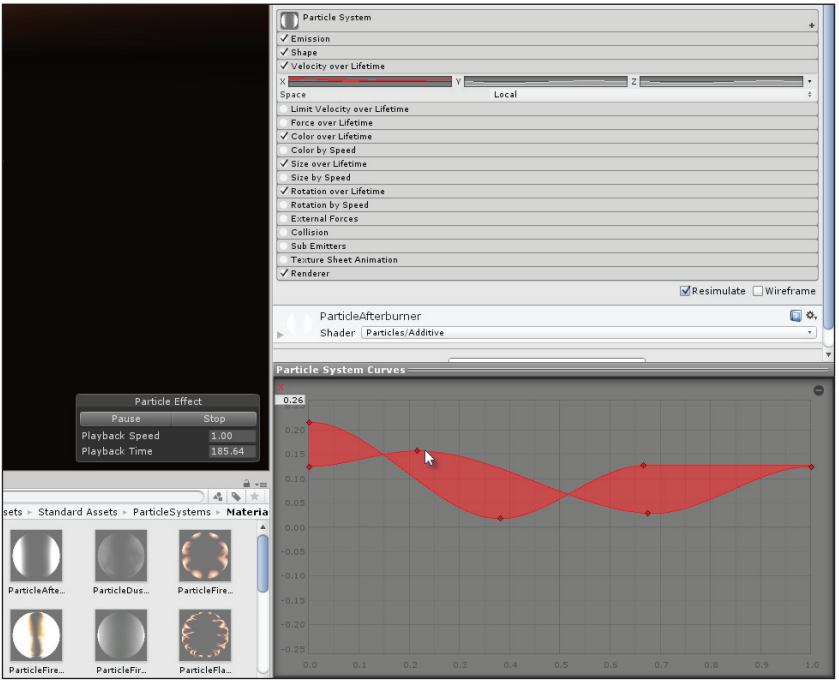

Рис. 3.26. График скорости частиц

После создания кривой для первой оси повторите процедуру для остальных осей. Когда вы закончите, частицы будут двигаться независимо друг от друга.

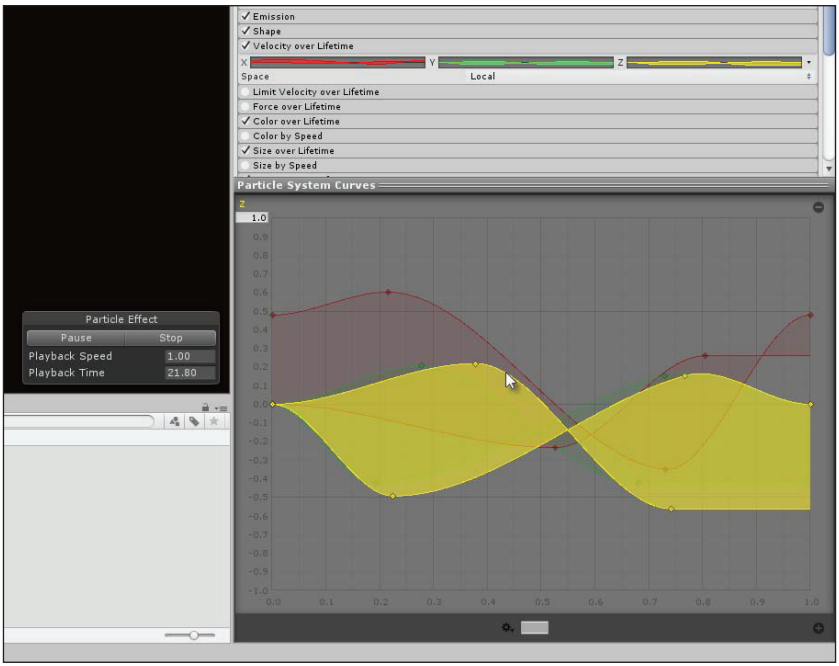

Рис. 3.27. Завершенный график **Velocity over Lifetime**

Цвет частиц и их исчезновение

Для завершения системы имитации светлячков мы должны изменить способ рождения и исчезновения частиц в системе. По умолчанию частицы рождаются, неожиданно появляясь, а затем умирают, мгновенно пропадая. Это приводит к резкому взаимоисключающему переходу между существованием и отсутствием, что придает всей системе в целом неестественный вид. Все это потому, что частицы видны в один момент времени и их уже нет в следующем. Чтобы решить эту проблему, мы можем использовать секцию смены цвета на протяжении существования частиц **Color over Lifetime** для изменения цвета частиц во времени. Важно отметить, что цвет частиц включает в себя альфа-канал или альфа-прозрачность. Следовательно, конт-

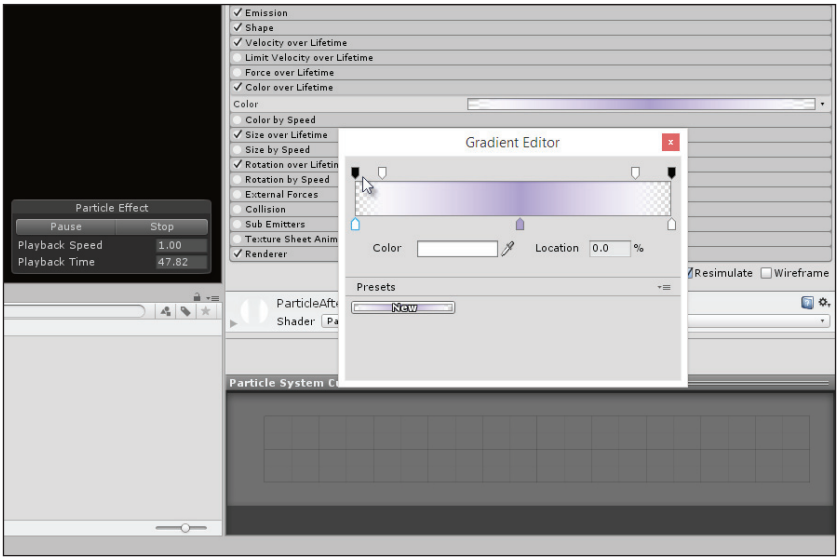

Рис. 3.28. Управление прозрачностью с помощью секции **Color over Lifetime**

ролируя цвет, мы можем заставлять частицы появляться и исчезать с помощью установки нужного уровня их прозрачности.

Для управления цветом на протяжении всей жизни частицы щелкните по образцу цвета градиента в секции **Color over Lifetime** в инспекторе объектов. При этом откроется окно редактора градиентов **Gradient Editor**. Здесь вы можете просмотреть изменение цвета частицы на протяжении ее существования в элементе управления градиентом. Горизонтальная ось полосы управления градиентом охватывает все время жизни частицы; левая сторона соответствует рождению частиц, а правая – уничтожению частиц. Нижняя часть градиентной полосы определяет оттенок (цвет) частицы, а верхняя часть – степень прозрачности градиента. Вы можете управлять прозрачностью, щелкнув по верхней части градиентной полосы для добавления закладки в виде ползунка в положении мыши, а затем щелкнув по самому ползунку, чтобы установить его альфа-значение на выбранный момент времени (0 значит полностью прозрачен, а 255 – полностью виден). Добавьте две закладки в начале и в конце градиентной полосы, чтобы установить прозрачность частиц в 0, отметив рождение и уничтожение частиц. Затем вставьте еще две закладки между 0 и 1 для уве-

личения видимости частиц до 255 – частицы полностью видны. Это приведет к плавному появлению частиц при их рождении и плавному исчезновению при уничтожении. При этом появление и исчезновение частиц при их входе в систему и их выходе из системы станут более гладкими и естественными.

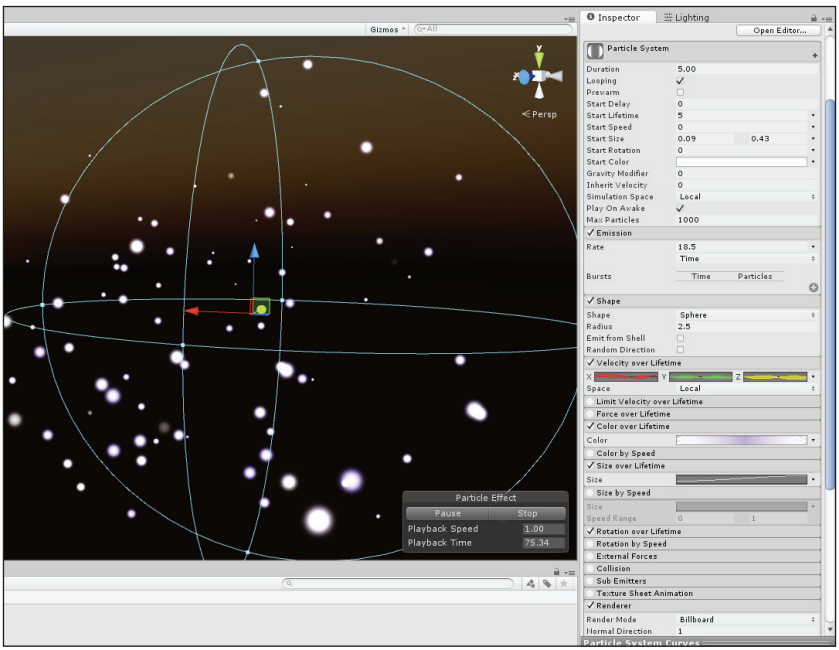

Рис. 3.29. Законченная система имитации светлячков

Поздравляю! Вы только что завершили систему частиц имитации светлячков, которая может быть использована в различных играх. Сейчас подходящее время, чтобы создать префаб системы, просто перетащив ее в любую из ваших сцен.

Итоги

Эта глава охватывает как встроенное в Unity окно анимации **Animation** для разработки анимации в самом приложении Unity, так и систему частиц для создания анимации со множеством движущихся элементов. При помощи редактора анимации создается клип ани-

мации **Animation Clip**, который определяет набор ключевых кадров для хранения изменений во времени, этот клип совместим (как мы увидим позже) с системой Mecanim. Системы частиц, напротив, полностью основаны на самой процедуре анимации и не хранят данные о ключевых кадрах. В системах частиц анимация создается динамически во время выполнения, на основании графиков, заданных с помощью редактора Сюрикэн в инспекторе объектов. Это придает системам частиц впечатляющий уровень гибкости. В следующей главе мы рассмотрим систему Mecanim, изучим различные виды анимации на основании системы Mecanim, не связанные с человекоподобными персонажами.

Глава 4

Анимация предметов с помощью системы Mecanim

В этой главе мы подробно рассмотрим анимацию с помощью системы Mecanim, научимся создавать интерактивные сцены с движущимися объектами. В частности, мы создадим сцену, где игрок (в игре от первого лица) может нажать кнопку или рычаг, чтобы открыть дверь, расположенную в другом месте. Этот простой скрипт будет включать интерактивность (игрок нажимает кнопку), и в ответ воспроизводится анимация (дверь поворачивается на петлях в ответ на нажатие кнопки). Создавая этот проект, мы будем исследовать систему анимации Mecanim в Unity и увидим, как создать интерактивность без традиционных скриптов. Впрыгивайте внутрь, и начнем работу.

Подготовка сцены с помощью прототипирования активов

Начнем рассмотрение анимации в этой главе с предварительно подготовленной сцены и проекта, который включен в приложенные к этой книге файлы и находится в папке Chapter04/DoorAnim. Сцена содержит несколько реквизитов и активы для работы с ними, она была легко собрана из прототипов активов, предоставляемых приложением Unity 5, которое содержит большое количество префабов и объектов (таких как полы и лестницы) для упрощения создания сцен со строениями. Я использую в качестве отправной точки пример такой сцены, показанный на следующем скриншоте. Тем не менее вы можете создать свой собственный уровень для той же цели, если хотите. Чтобы импортировать пакет активов примеров, выполните **Assets → Import Package → Prototyping** из меню приложения.

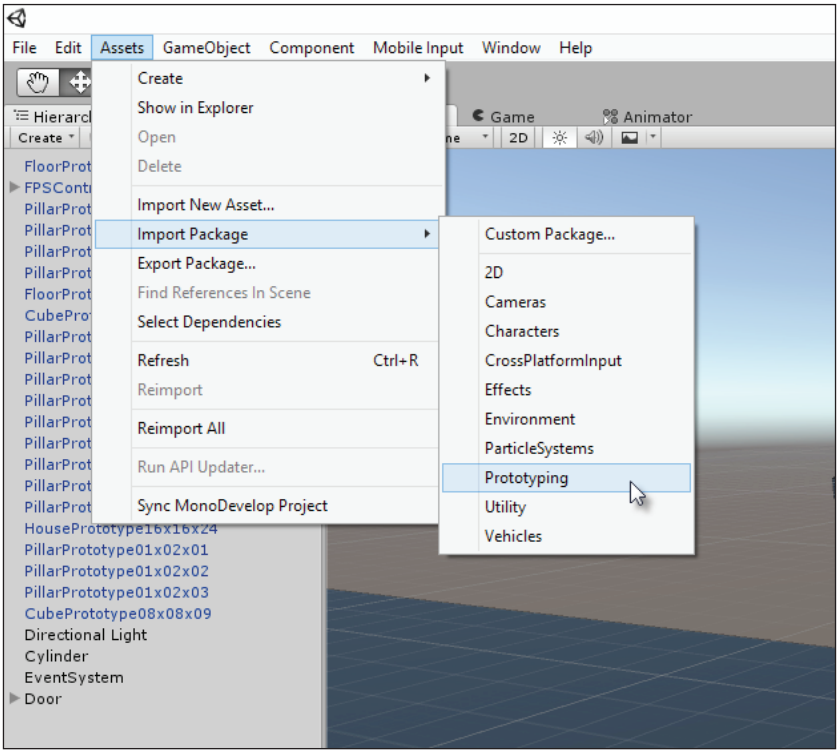

Рис. 4.1. Импорт примеров прототипов активов

После того как прототипы активов импортированы, вы можете перетащить предварительно подготовленные префабы в сцену и начать строить внешнюю среду. Прототипы префабов включены в **Standard Assets → Prototyping → Prefabs** (рис. 4.2).

Вот сцена, которую мы будем использовать для анимационного проекта. Она включает в себя кнопку, которую нужно нажать, чтобы разблокировать и открыть дверь, ведущую в другую часть внешней среды (рис. 4.3).

Создание анимаций для кнопки и двери

Прежде чем мы непосредственно начнем работать с системой Mecanim, мы должны подготовить для дальнейшей работы данные анимации, конкретнее, мы должны будем создать две анимации для кнопки

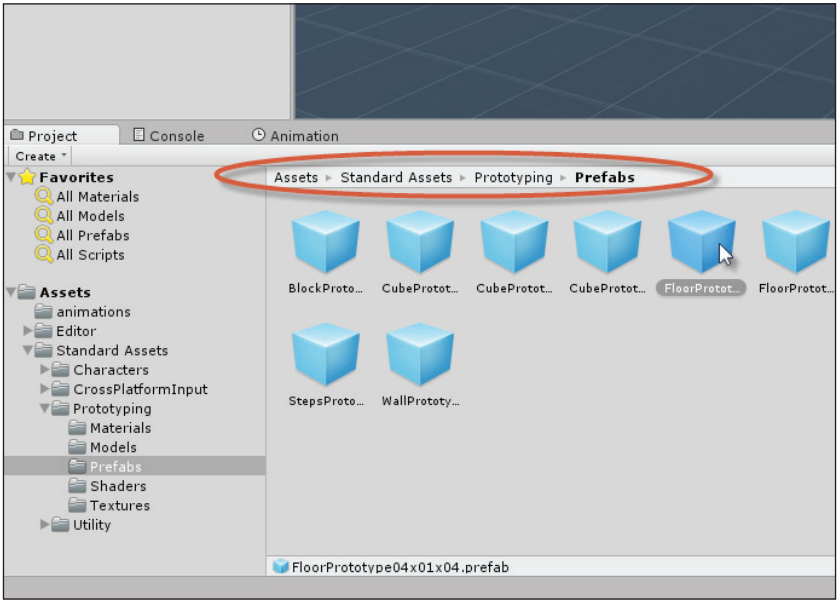

Рис. 4.2. Прототипирование префабов

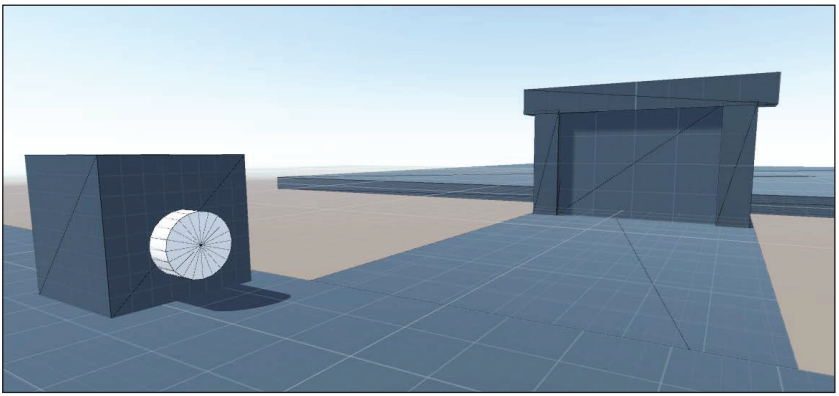

Рис. 4.3. Кнопка и дверь

и для двери. При нажатии на кнопку открывается дверь. Следовательно, нам понадобится анимация для кнопки – для отображения ее движения внутрь гнезда при нажатии, а затем возврата в нейтральное положение. Нам также понадобится анимация открытия двери, то

есть вращение двери на петлях при открывании наружу. Эта анимация должна быть воспроизведена только после нажатия кнопки. Обе эти анимации могут быть созданы непосредственно в редакторе Unity с помощью окна анимации, как это описано в предыдущей главе. По этой причине здесь я опишу процесс создания анимаций лишь вкратце.

Чтобы создать анимацию нажатия кнопки, выберите объект кнопки в сцене, или в окне обзора, или в иерархии, а затем перейдите в окно анимации. Это можно сделать, выполнив **Window → Animation** из главного меню или нажав **Ctrl+6**. В окне анимации добавьте ключевые кадры для канала `Transform.Position` в начале, в середине и в конце шкалы времени, то есть для моментов времени 0 секунд, 0,5 секунды и 1 секунда.

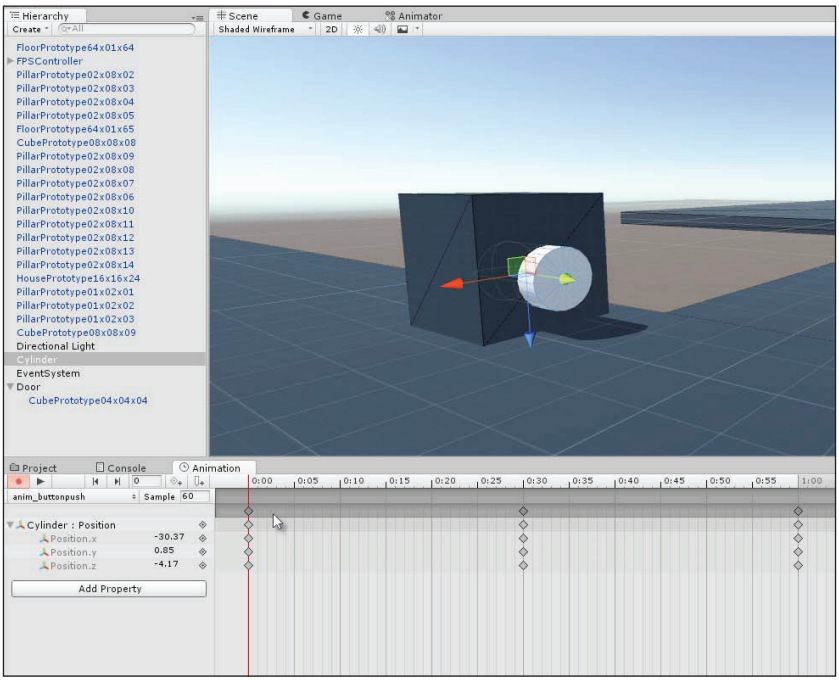

Рис. 4.4. Создание анимации нажатия кнопки

Начальный и конечный ключевые кадры анимации должны быть одинаковыми, на них кнопка должна находиться в нейтральном отпущенном состоянии. Средний ключевой кадр, напротив, должен

соответствовать полностью утопленной в гнезде кнопке (состояние «нажата»). Чтобы создать этот ключевой кадр, перетащите ползунок шкалы времени так, чтобы красная линия находилась в середине шкалы в окне анимации, а затем переместите кнопку в сцене в нужное положение. Такое расположение ключевых кадров гарантирует, что при нажатии кнопки она начинает анимацию с нейтрального состояния, затем смещается внутрь и, наконец, возвращается в нейтральное состояние. Для повышения плавности движения кнопки в начале и в конце анимации вы можете сгладить кривую движения, обработав ее в редакторе кривых, доступном из окна анимации, нажав на кнопку **Curves** в нижнем левом углу окна. Если вы не сможете изменить форму кривой в ручном режиме **Handles**, то щелкните правой кнопкой мыши по нужной точке и выберите опцию **Flat** из контекстного меню.

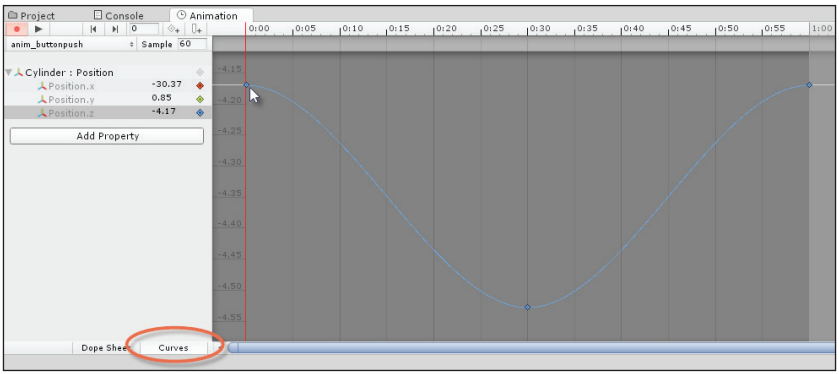

Рис. 4.5. Сглаживание движения кнопки

Анимация открытия двери создается аналогичным образом, хотя и потребуется несколько дополнительных операций конфигурирования. В частности, дверь при ее открытии должна вращаться вокруг точек крепления ее за петли. Ось вращения по умолчанию для куба, короба или меша объекта, представляющего собой нашу дверь, не всегда подходит для создания анимации вращения. Мы, например, не захотим, чтобы дверь при открытии поворачивалась вокруг своего центра. В Unity это может быть легко исправлено с помощью присоединения меша двери к родительскому пустому объекту, расположенному в месте поворота, а затем анимированием пустого объекта, а не меша двери напрямую.

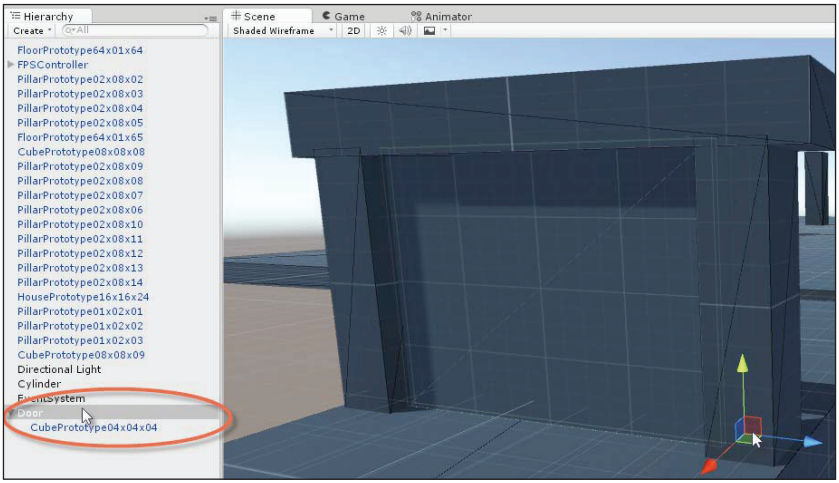

Рис. 4.6. Создание точки поворота для анимации открытия двери

Для анимации открытия двери должны быть созданы два ключевых кадра: один – в начале анимации (закрыта) и один – в конце (открыта). Как и для анимации нажатия кнопки, для сглаживания кривых анимации можно использовать редактор кривых, чтобы создать плавное движение и добиться требуемого эффекта.

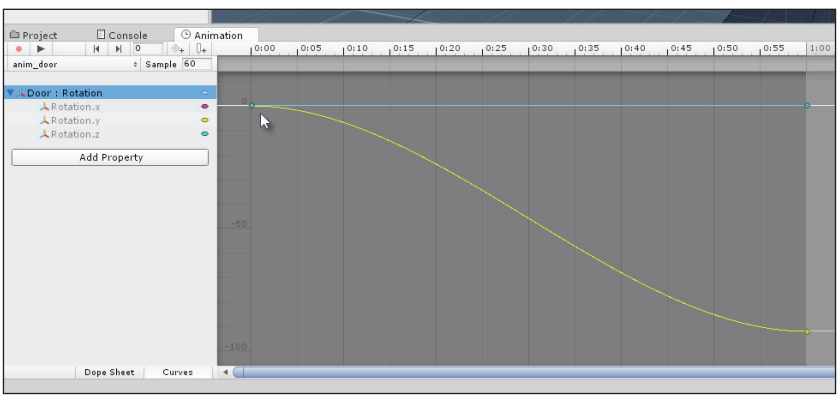

Рис. 4.7. Задание анимационной кривой для анимации открытия двери

Начало работы с системой Mecanim

После создания анимаций для кнопки и двери они будут автоматически воспроизводиться в цикле на вкладке **Game** при каждой активации сцены. Они автоматически запускаются, потому что после создания актива анимационного клипа Unity автоматически генерирует актив контроллера анимации `Animator Controller` и присоединяет его к объекту в сцене. Сам контроллер является частью системы Mecanim и использует автомат состояний **State Machine** для определения момента воспроизведения анимационного клипа для объекта. По умолчанию он воспроизводит анимационный клип при запуске сцены. Анимация воспроизводится в цикле, потому что все новые активы анимационных клипов по умолчанию определяются как зацикленные. Ни автоматическое воспроизведение, ни зацикливание в данном случае не нужны, потому что анимация кнопки должна воспроизводиться только при нажатии ее игроком, а дверь должна открываться лишь в ответ на нажатие кнопки. Давайте сначала отключим зацикливание для каждого анимационного клипа. Чтобы сделать это, выберите по очереди каждый анимационный клип в панели проекта и в инспекторе объектов отключите флажок **Loop Time**. Это отключит зацикливание воспроизведения анимаций. Вместо этого они будут воспроизводиться только один раз, а затем останавливаться.

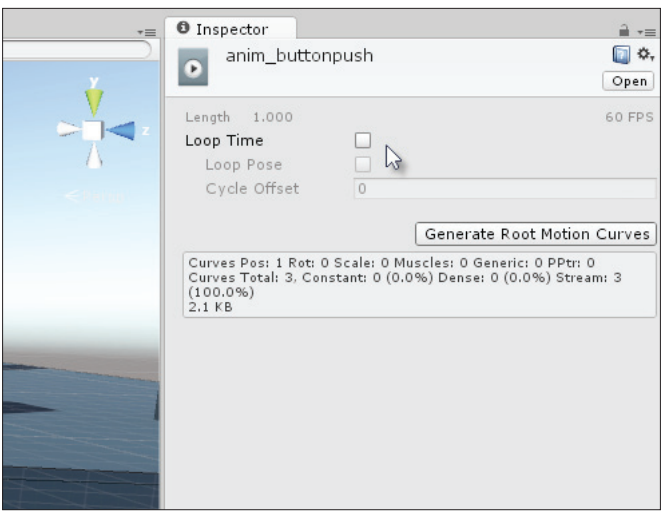

Рис. 4.8. Отключение зацикливания анимаций

Даже если зацикливание воспроизведения отключено, анимация будет по-прежнему воспроизводиться автоматически при запуске уровня, хотя и только один раз. Это происходит потому, что актив контроллера анимации присоединен к каждому из объектов двери и кнопки. Чтобы изменить это, мы должны будем открыть окно аниматора Animator. Вы можете сделать это, выполнив **Window → Animator** в меню приложения, или можете дважды щелкнуть на актив контроллера анимации **Animation Controller** в панели проекта. Активы контроллеров анимации генерируются вместе с активами анимационных клипов **Animation Clip**.

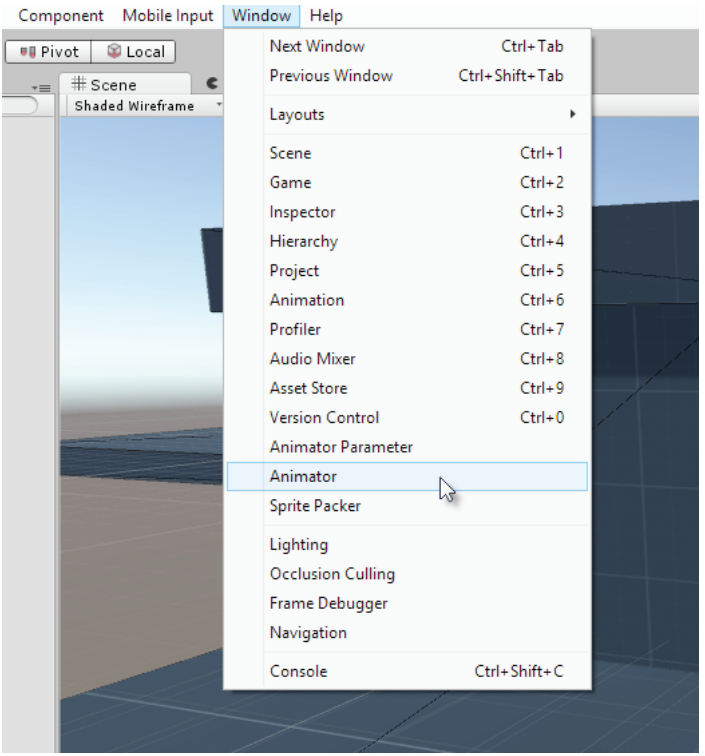

Рис. 4.9. Доступ к контроллеру аниматора

Если вы выберите объект кнопки в сцене и посмотрите на окно аниматора **Animator**, то увидите автомат состояний системы Mecanim, состоящий из соединенных между собой узлов (называемых **состоя-**

ниями (states)). Состояния управляют тем, как и когда должны воспроизводиться анимационные клипы объекта. По умолчанию узел состояния зеленого цвета **Entry** связан с узлом состояния **anim_ ButtonPush** (здесь может стоять и любое другое имя, назначенное анимации кнопки). Узел **Entry** загорается, или активируется, при запуске сцены. В этот момент происходит активация и следующего с ним узла, если к нему, согласно логике графа, идет стрелка соединения. По этой причине анимация кнопки нажатия воспроизводится автоматически при запуске сцены, так как узел **Entry** подключен к узлу клипа анимации нажатия кнопки. Давайте откорректируем этот граф для нужного нам сценария поведения. Для начала щелкните правой кнопкой мыши в любом месте области графа и выполните **Create State → Empty** из контекстного меню. При этом система Mecanim создаст новый пустой узел в графе. Это значит, что связанный объект в сцене не будет делать *ничего* при активации этого узла.

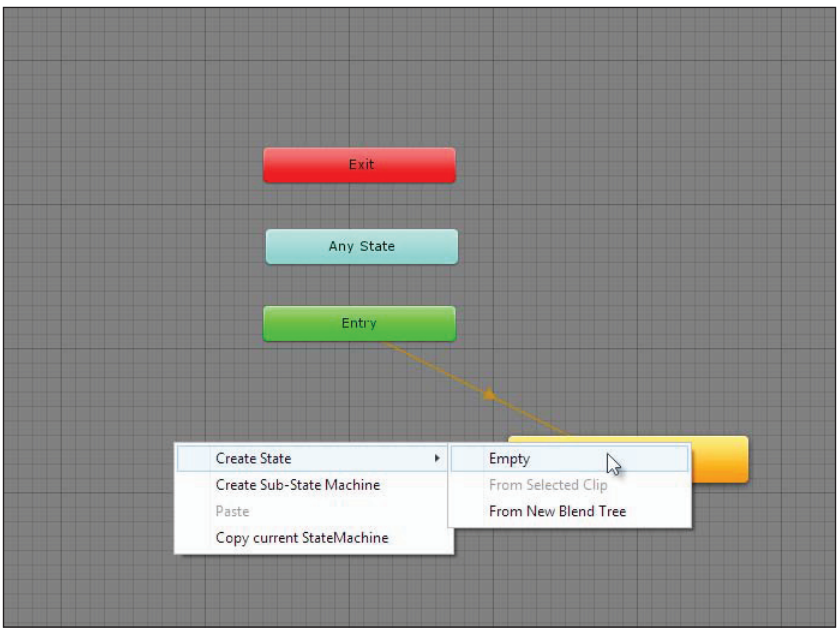

Рис. 4.10. Создание пустого состояния
в графе системы Mecanim

Однако вновь созданный узел ни к чему не подключен. Это означает, что он никогда не станет активным, потому что с ним не связан узел **Entry**, с активации которого всегда все начинается. Узел **Entry** всегда подключается к узлу по умолчанию, который будет выделен оранжевым цветом. А сейчас выберите вновь созданный пустой узел и назовите его `Idle`, набрав его имя в поле **Name** инспектора объектов.

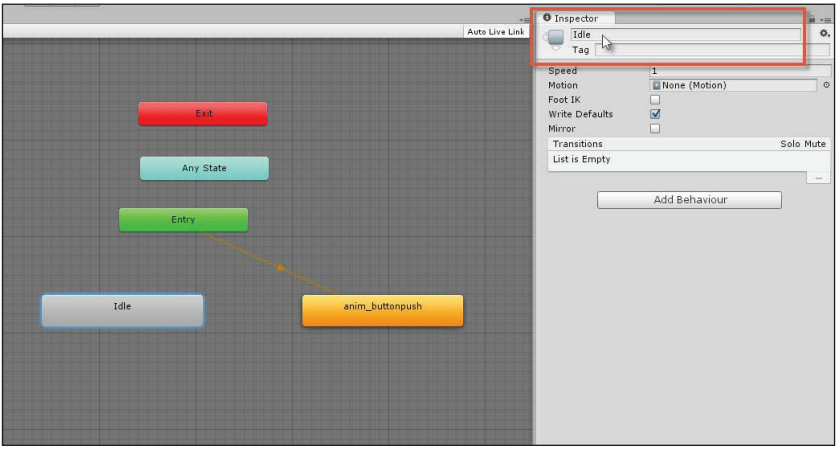

Рис. 4.11. Присвоение имен узлам системы Mecanim

Как уже упоминалось, узел ввода всегда соединяется с узлом по умолчанию, который выделяется оранжевым цветом. Давайте сделаем узел **Idle** узлом по умолчанию. Чтобы сделать это, щелкните правой кнопкой мыши по узлу **Idle** и выберите пункт **Set as Layer Default State**. Теперь узел **Entry** переключился на узел **Idle** (рис. 4.12). Это значит, что анимация нажатия кнопки не будет воспроизводиться автоматически при запуске сцены, это то, что нам было нужно. Но это только половина того, чего мы хотели!

Переходы и параметры системы Mecanim

Хотя проблема графа системы Mecanim, связанная с автоматическим запуском анимации при старте сцены, и решена, но до сих пор не определен способ воспроизведения анимации кнопки в тот момент, когда игрок нажмет на нее. Цель теперь состоит в том, чтобы откорректировать граф системы Mecanim так, чтобы получить кон-

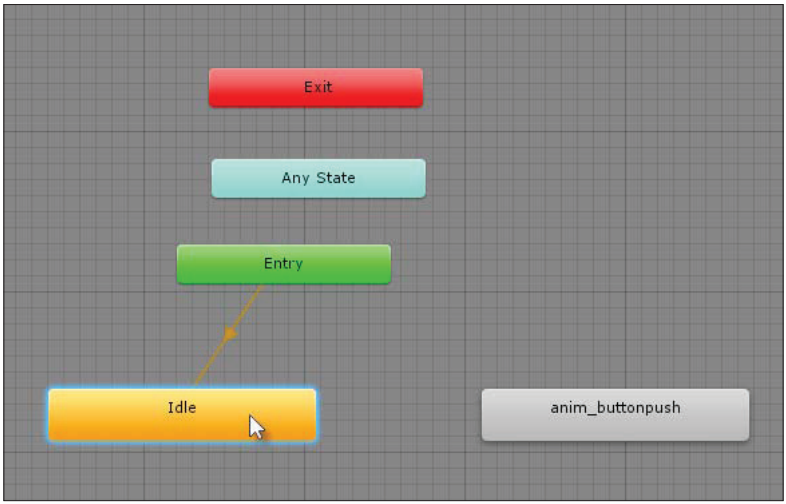

Рис. 4.12. Установка состояния по умолчанию

троль над анимацией нажатия кнопки, запуская ее воспроизведение в нужный момент времени. Для достижения этой цели мы будем использовать **переходы** (transitions) и **параметры** (parameters). Переходы – это просто соединения между двумя состояниями, которые позволяют одному состоянию перейти в другое. Параметры являются переменными или переключателями, которые предназначены для управления переходами между двумя состояниями. Давайте теперь создадим параметр, который позволит нам определить, когда состояние бездействия перейдет в состояние нажатия кнопки. То есть этот параметр будет управлять воспроизведением анимации. Внутри окна аниматора щелкните по вкладке параметров **Parameters** в верхнем левом углу (рис. 4.13).

Затем щелкните по иконке + для добавления нового параметра. Существуют различные типы параметров. Для этого примера выберите тип триггер **Trigger** (рис. 4.14). Триггеры действуют подобно переключателям управления, их активация привет к разрешению перехода из одного состояния в другое.

Задайте триггеру имя, например `PushButton`. После создания триггер появится в списке параметров с флажком рядом с его именем, который может быть установлен или сброшен прямо во время выполнения для целей отладки, это позволяет активировать триггер непосредственно из редактора. Теперь, когда триггер создан, он должен

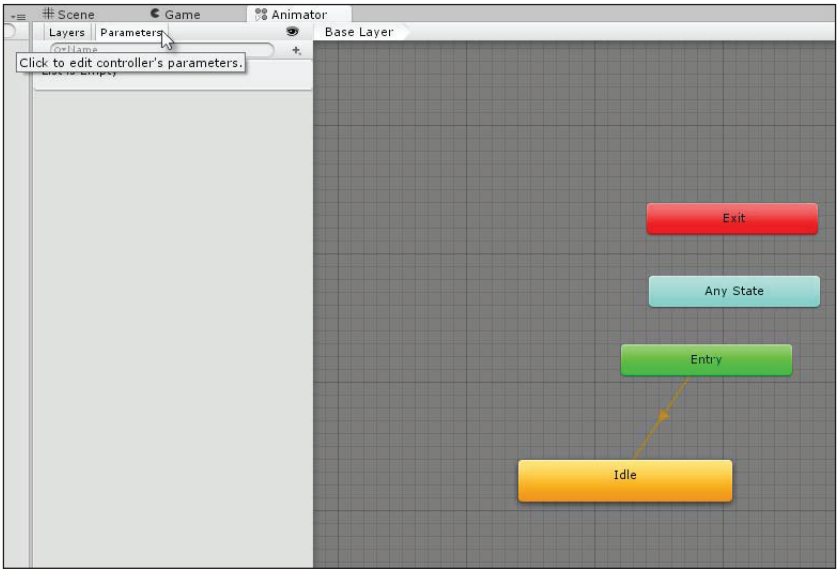

Рис. 4.13. Доступ к закладке параметров **Parameters**

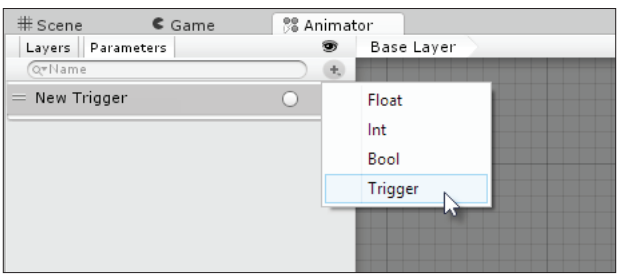

Рис. 4.14. Создание триггера

быть связан с переходом в графе. Переход должен быть создан между состоянием **Idle** и анимацией нажатия кнопки. При этом будет установлена связь между состоянием **Idle** и анимацией. Чтобы создать переход, щелкните правой кнопкой мыши по состоянию **Idle** в графе, а затем выберите опцию **Make Transition** из контекстного меню (рис. 4.15).

Щелкните по стрелке перехода, который начинается от состояния **Idle** и заканчивается в анимации кнопки, создавая их одностороннюю связь. При выборе перехода между состояниями вы увидите в ин-

спекторе объектов несколько свойств. Эти свойства определяют, как и когда происходит переход.

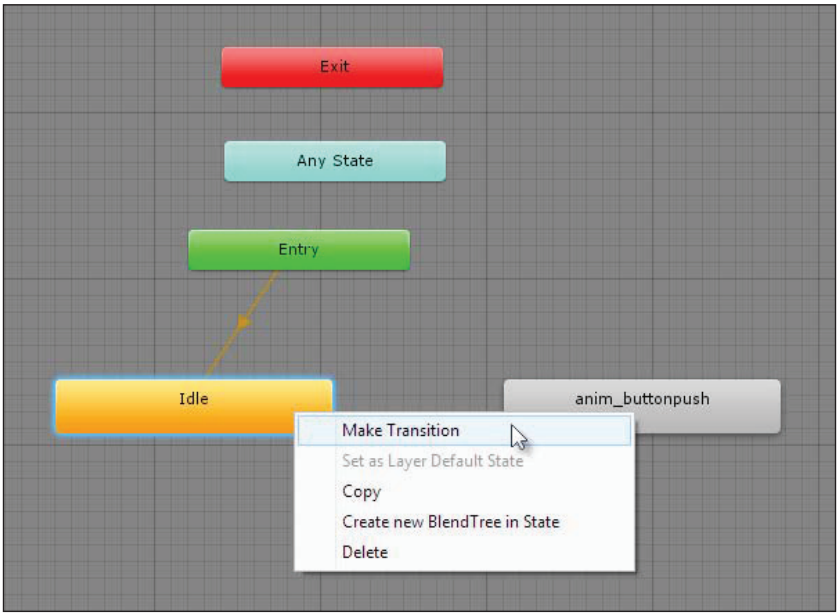

Рис. 4.15. Создание перехода

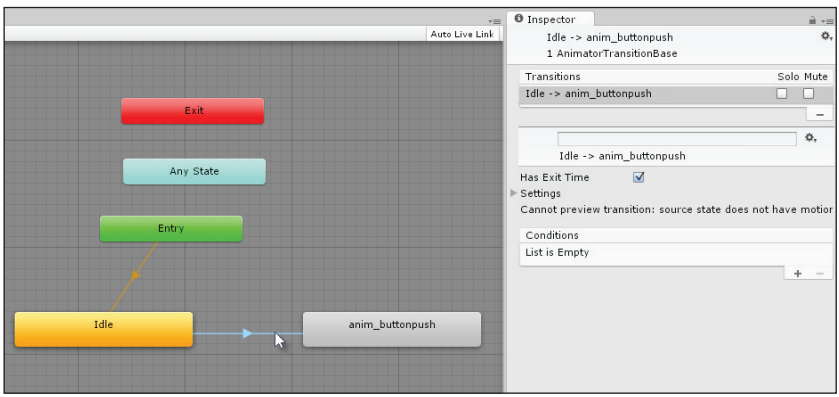

Рис. 4.16. При выборе перехода в панели инспектора появляется несколько свойств

По умолчанию переход происходит в момент завершения анимации, а это значит, что состояние **Idle** воспроизводится один раз или в течение некоторого периода времени, а затем происходит автоматический переход к состоянию анимации нажатия кнопки, даже если игрок не нажал на кнопку. Это можно проверить, запустив сцену и понаблюдав, что произойдет с объектом кнопки на вкладке сцены **Scene**. Эта кнопка будет автоматически утоплена внутрь через несколько секунд после запуска сцены. Переход должен происходить не так, как это определено по умолчанию, а при активации триггера **PushButton**. Чтобы добиться этого, отключите флажок **Has Exit Time** в инспекторе объектов при выбранном переходе. Это предотвратит автоматический переход из состояния **Idle** на анимацию нажатия кнопки.

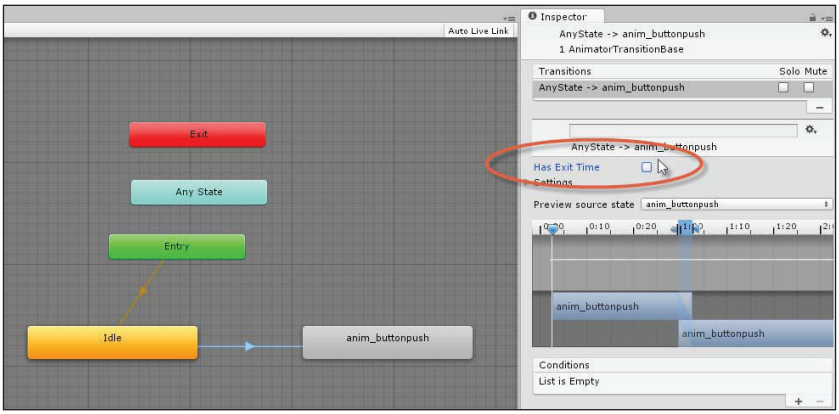

Рис. 4.17. Отключение флажка **Has Exit Time**

Следующая панель условий **Conditions** в инспекторе объектов может быть использована для задания, когда переход должен быть активирован. Нажмите на иконку **+**, чтобы добавить новое условие. Затем из раскрывающегося списка выберите триггер **PushButton**, если он еще не выбран. Это наконец соединит триггер **PushButton** с переходом (рис. 4.18).

Вы можете проверить соединение триггера с переходом во время выполнения. Выберите объект кнопки на сцене, а затем нажмите на кнопку **Play** на панели инструментов. Потом щелкните по флажку **Trigger** в панели параметров окна аниматора. Когда вы сделаете это, то увидите воспроизведение анимации нажатия кнопки на закладке сцены **Scene** или игры **Game**.

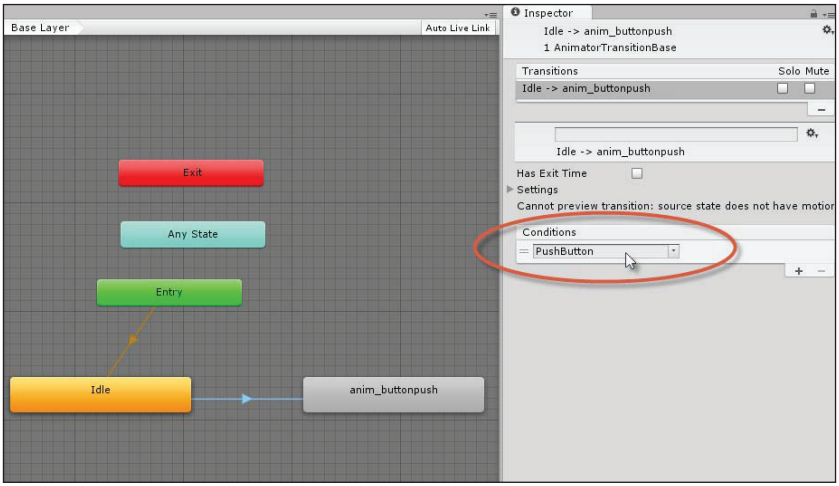

Рис. 4.18. Подключение перехода к триггеру нажатия кнопки **PushButton**

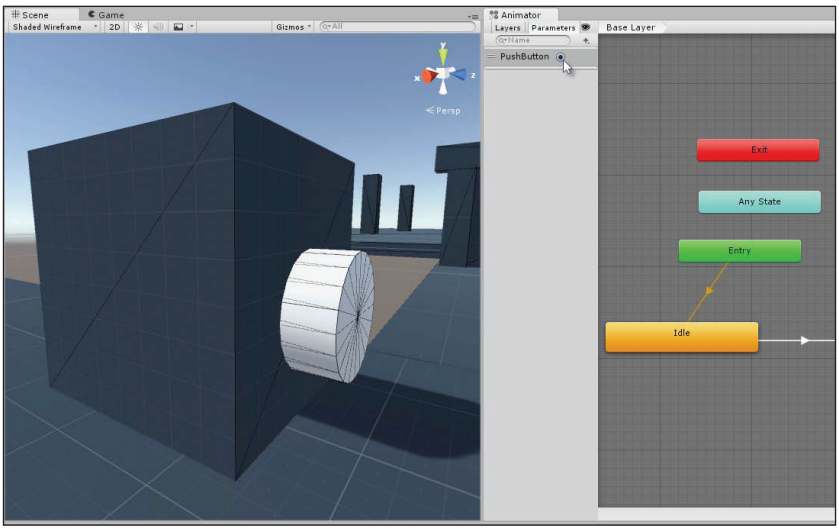

Рис. 4.19. Тестирование триггера нажатия кнопки **PushButton**

Последней проблемой нашего графа автомата состояний системы Mecanim является то, что после того, как нажата кнопка, граф пере-

ходит в состояние анимации нажатия кнопки и останется там и после того, как анимация завершится. Это значит, что игрок не может нажать на кнопку второй, третий или четвертый раз просто потому, что граф «застрял» в состоянии анимации. Мы же хотим, чтобы из состояния анимации нажатия кнопки, после ее завершения, выполнялся переход обратно в состояние **Idle**, что позволит несколько раз нажать на кнопку. Чтобы добиться этого, щелкните правой кнопкой мыши по состоянию анимации кнопки в графе и выберите **Make Transition** из появившегося контекстного меню. Затем перетащите соединение назад на состояние **Idle**. Значения по умолчанию всех настроек для этого перехода можно оставить без изменений.

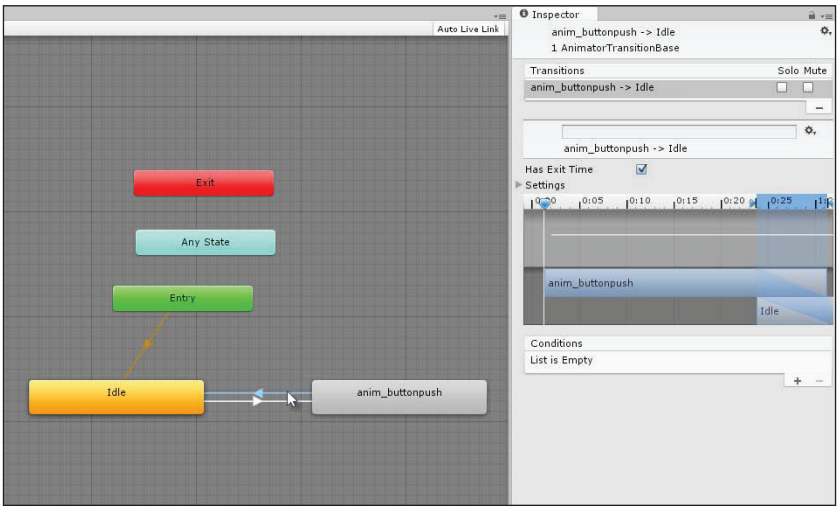

Рис. 4.20. Создание двусторонней связи между состояниями

Теперь граф системы Mecanim для объекта кнопки в сцене завершен. Как мы видели, связь между триггером и переходом можно проверить во время выполнения из окна аниматора. Но даже если это и так, то игрок все еще не может вызвать переход, нажав на объект кнопки в сцене. Это будет исправлено в ближайшее время, после рассмотрения использования графа системы Mecanim применительно к двери.

Создание графа системы Mecanim для открывания двери

Описанное выше создание анимации кнопки можно повторить и для анимации открывания двери, которая также может быть запущена с помощью триггера. Рассмотрим следующий скриншот с законченным графом системы Mecanim для двери. На нем показано, что из состояния **Idle** (дверь закрыта) переход к анимации открытия двери выполняется при активации триггера **OpenDoor**.

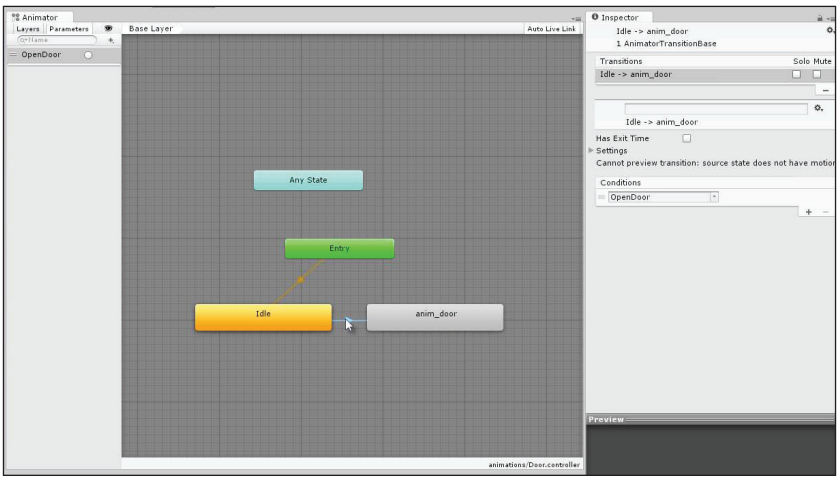

Рис. 4.21. Законченный граф системы Mecanim для двери

Граф открытия двери можно протестировать так же, как и граф кнопки. Выберите объект двери в сцене или в панели иерархии, запустите игру, а затем щелкните по триггеру **Door Open** в панели параметров окна аниматора. Это должно привести к открытию двери (рис. 4.22).

Создание интерактивных сцен

Созданный здесь пример сцены содержит два анимационных клипа (нажатия кнопки и открывания двери) и два анимационных контроллера, которые связывают клипы с состоянием объектов сцены. В предыдущих разделах контроллеры были снабжены триггерами, так что

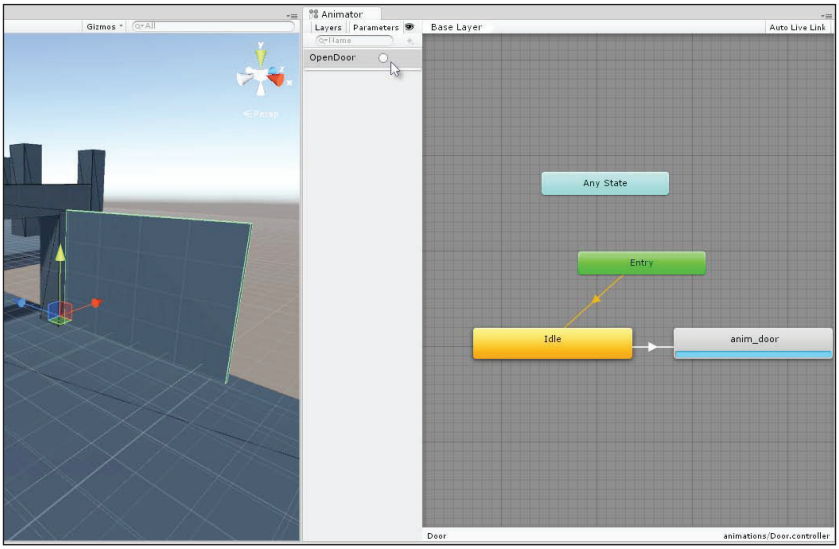

Рис. 4.22. Тестирование анимации открытия двери

и анимация двери, и анимация кнопки могут быть воспроизведены по требованию, при включении соответствующего триггера. Теперь проблема в том, что игрок не может активировать триггеры *из игры*. Только мы как разработчики можем сделать это с помощью редактора Unity в режиме игры. Наша цель сейчас – добиться того, чтобы при нажатии игроком кнопки открывалась дверь. Это значит, что кнопка должна активировать оба триггера контроллеров анимации: один – для воспроизведения анимации кнопки, а другой – для открывания двери. Есть много способов для достижения такой интерактивности. Один из способов – это использование скриптов, но мы не должны заходить так далеко. Мы можем использовать систему событий Unity. В этом разделе мы увидим, как.

Система событий Unity отслеживает перемещения мыши и ввод с клавиатуры и автоматически определяет с помощью специальной камеры, по какому объекту в сцене был произведен щелчок. Кроме того, она позволяет нам задать способ адекватного реагирования на ввод с помощью системы Visual Scripting. Для начала создайте объект системы событий (System Event) в сцене, выполнив **GameObject → UI → Event System**. Хотя эта опция и помещена в категорию пользовательского интерфейса **UI**, на самом деле система событий имеет множество применений и вне интерфейса.

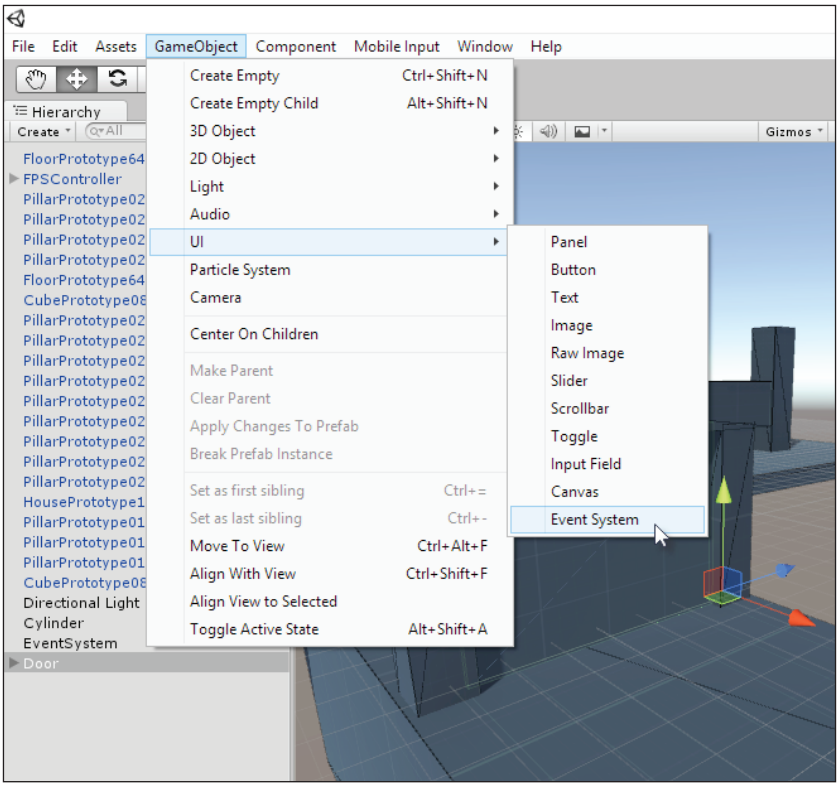

Рис. 4.23. Создание системы событий

Далее найдите основную камеру в сцене и выберите ее. Для примера сцены, которая включена в прилагаемые к этой книге файлы, камера присоединена к FirstPersonCharacter. После того как объект основной камеры выбран, добавьте к нему компонент **Physics Raycaster**, выполнив **Component → Event → Physics Raycaster** из меню приложения. Добавление его к камере позволит системе событий обнаруживать щелчки пользователя и определять, какой из объектов сцены был выбран в момент щелчка, на основании того, на чем была сфокусирована камера. Важно прикрепить этот компонент к камере игры, камере, через которую игрок видит мир, либо нет необходимости обнаруживать щелки по прочим объектам, не находящимся в фокусе камеры.

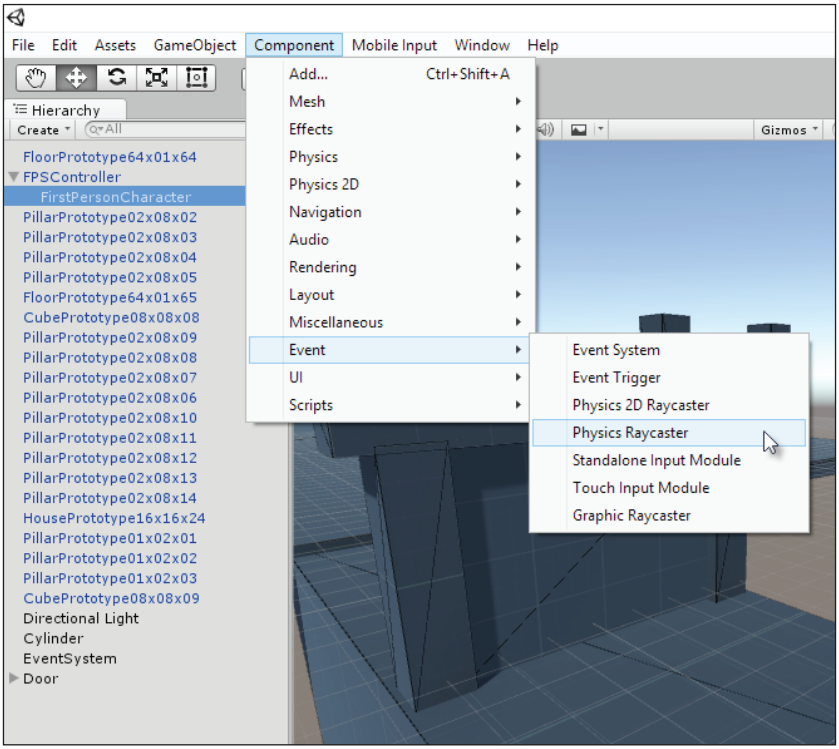

Рис. 4.24. Добавление компонента **Physics Raycaster** к камере

После добавления системы событий в сцену и присоединения компонента **Physics Raycaster** к камере мы готовы к тому, чтобы начать настройку объекта кнопки для воспроизведения необходимых анимаций. Выберите объект кнопки в сцене (объект, который должен быть нажат игроком, чтобы открыть дверь) и добавьте компонент **Event Trigger**, выполнив **Component → Event → Event Trigger** из меню приложения (рис. 4.24).

После добавления компонента **Event Trigger** к объекту кнопки нажмите на кнопку **Add New Event Type** в инспекторе объектов. Делая это, мы определяем, что хотим отслеживать некое событие и получать уведомления, когда оно происходит. Из появившегося контекстного меню выберите событие **PointerClick** (рис. 4.26).

Это приведет к добавлению новой секции или панели, в которой можно выбрать одно или несколько действий, выполняемых при воз-

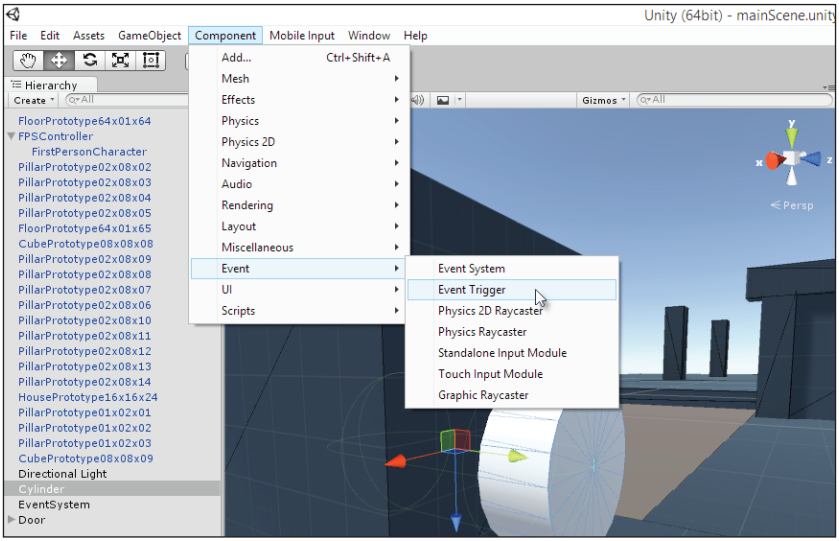

Рис. 4.25. Добавление компонента **Event Trigger** к объекту кнопки в сцене

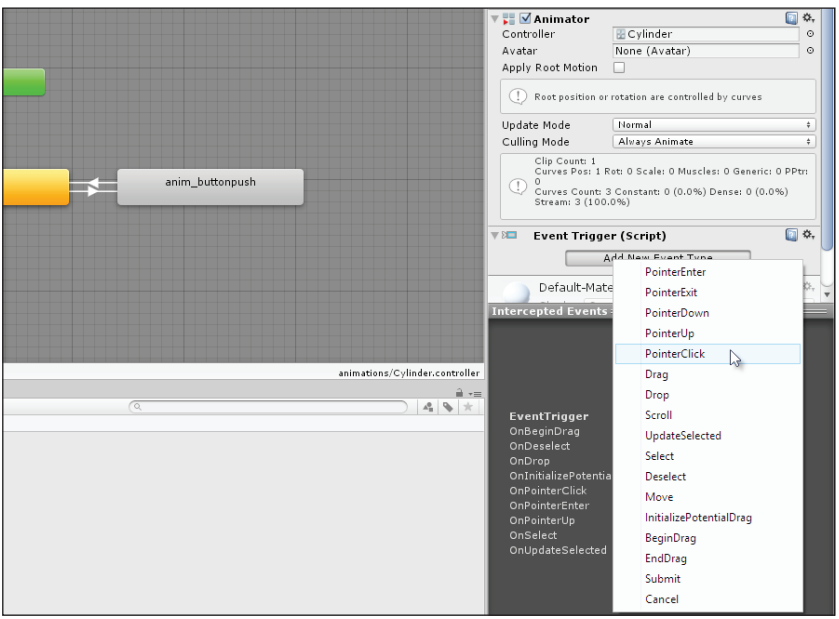

Рис. 4.26. Управление событием **PointerClick**

никновении события **PointerClick** для кнопки. В нашем случае должны быть выполнены два действия. Во-первых, мы должны запустить анимацию кнопки, используя триггер **PushButton**, а во-вторых, мы должны запустить анимацию открывания двери с помощью триггера **DoorOpen**. Чтобы реализовать это, нажмите иконку + в компоненте **Event Trigger**, чтобы добавить новое действие в раздел события **PointerClick**. При этом будет создан пустой слот, готовый к настройке.

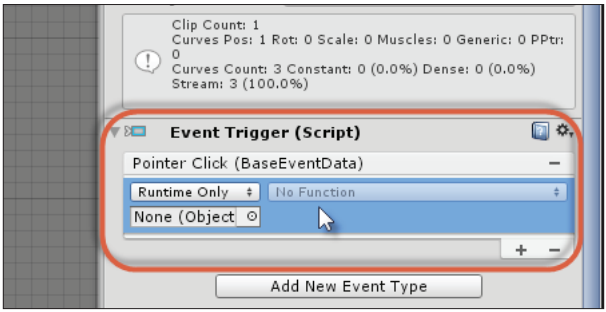

Рис. 4.27. Создание нового действия для компонента **Event Trigger** и события **PointerClick**

Для каждого действия, добавленного в событие, должны быть получены ответы на несколько вопросов. Во-первых, какому объекту сцены будут адресованы наши действия или какой объект сцены они затронут? Во-вторых, какая функция или какое свойство этого объекта должны быть запущены или изменены? Наконец, в-третьих, потребуются ли какие-то аргументы или параметры? Давайте рассмотрим это на нашем примере. При выполнении первого действия должна быть активирована анимация нажатия кнопки. Это действие над объектом кнопки (потому что контроллер анимации **Animation**

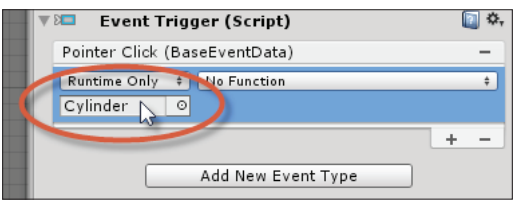

Рис. 4.28. Определение объекта-приемника для действия

Controller присоединен к объекту кнопки). Так что перетащите объект кнопки в слот объекта игры, чтобы определить кнопку как объект-приемник.

Это определит, что для объекта кнопки Unity необходимо активировать триггер, когда происходит щелчок.

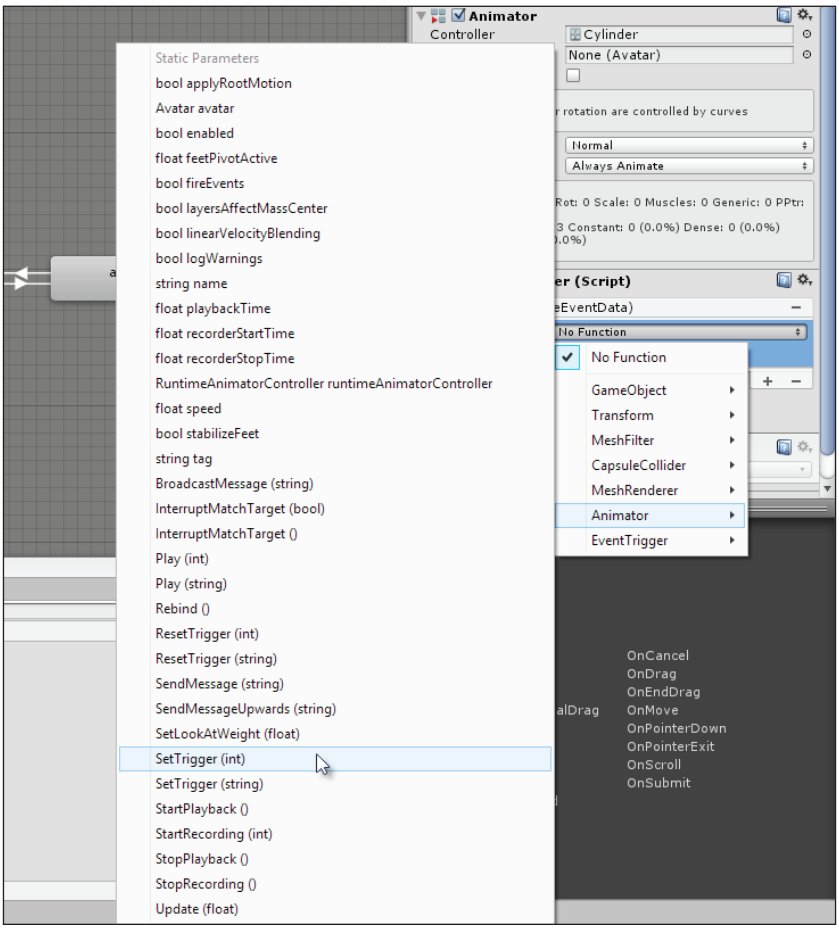

Рис. 4.29. Определение функции для действия объекта кнопки

Наконец, в строковом поле введите имя триггера, который должен быть активирован. Для объекта кнопки это **PushButton**, имя соответствующего триггера, определенного в контроллере анимации системы Mecanim.

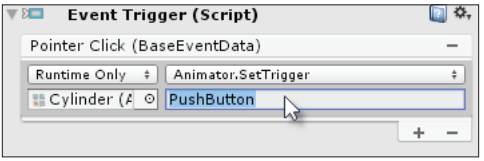

Рис. 4.30. Определение функции для действия объекта кнопки

Щелкните по иконке + еще раз, чтобы добавить второе действие, действие для объекта двери, и, использовав функцию **SetTrigger**, установите триггер **OpenDoor**, открывающий дверь.

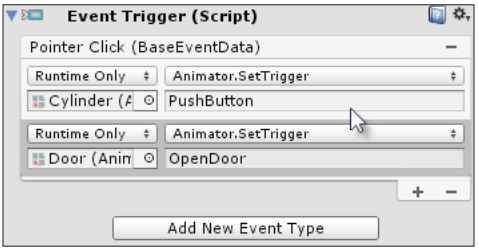

Рис. 4.31. Завершенный компонент **Event Trigger** для объекта кнопки

Теперь мы готовы протестировать проект, чтобы убедиться, что дверь будет открываться при нажатии на кнопку с анимацией. Чтобы проверка прошла успешно, удостоверьтесь, что при выполнении щелчка курсор мыши находится над кнопкой, и тогда и анимация кнопки, и анимация двери будут проигрываться автоматически. Великолепная работа! Вы сейчас создали интерактивную, анимированную сцену с помощью системы Mecanim.

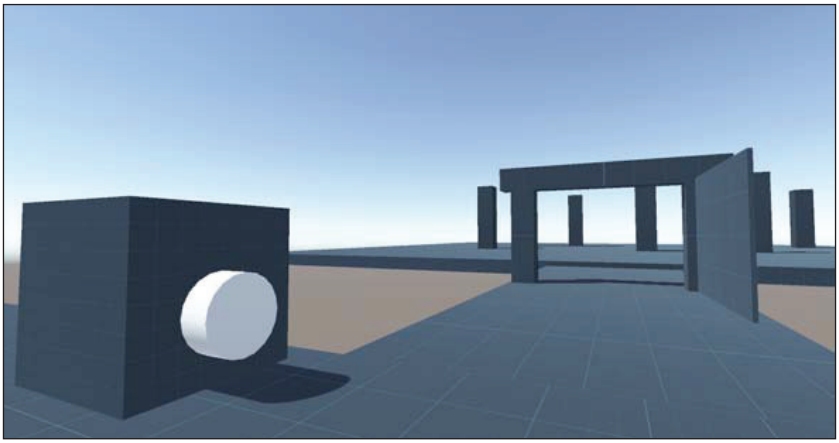

Рис. 4.32. Открытие двери при нажатии кнопки

Итоги

В этой главе показано, как создавать интерактивные сцены, используя анимационные клипы, объединяющие систему Mecanim и систему событий. Используя набор из этих трех инструментов, мы создали анимированную сцену, в которой нажатие кнопки открывает дверь. Несмотря на простоту в дизайне и принципе действия, этот пример охватывает широкий спектр применения анимации и демонстрирует, какую функциональность можно реализовать, используя только анимацию и не привлекая скриптов. В следующей главе мы начнем работу над анимацией персонажей в системе Mecanim.

Глава 5

Основы анимации персонажей

Эта глава начинает тему анимации персонажей, которая будет продолжена в следующей главе. В этой главе мы рассмотрим вопросы импорта и настройки, методы и процессы, необходимые для получения наилучших результатов при анимации персонажей. В частности, мы опишем импорт мешей, аватаров, скелетов, мышц, ретаргейтинг и перемещение корневого объекта. Итак, начнем!

Создание оснащенных персонажей

Для правильной анимации в Unity персонажи должны быть предварительно оснащены в программе 3D-моделирования, такой как 3ds Max, Maya или Blender. Оснащение означает, что художником-дизайнером к модели добавлен внутренний скелет или человекоподобная структура костей и каждая кость (например, кость руки, кость ноги и прочие) специальным образом привешена (примагничена) к вершинам меша. Назначение костей – в задании того, каким образом вершины меша персонажа должны двигаться и деформироваться, продолжая соответствовать скелету. Это упрощает анимацию персонажей, потому что, вместо того чтобы анимировать все вершины модели персонажа с высокой детализацией, аниматор может анимировать скелет (кости) и получать деформации персонажа автоматически в соответствии со скелетом.

Подробное рассмотрение самой оснастки персонажей и процесса создания такой оснастки специфично для выбранного для этого программного обеспечения и выходит за рамки этой книги. Автор данной книги предполагает, что у вас уже есть готовый для импорта оснащенный персонаж. Кроме того, вы можете создать оснащенные персонажи быстро и бесплатно с помощью программы MakeHuman. Эта программа может быть загружена с сайта http://www.makehuman.org/.

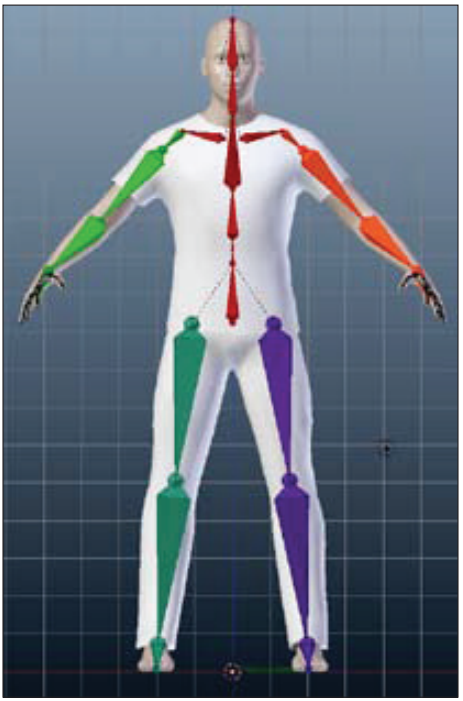

Рис. 5.1. Оснащенный персонаж

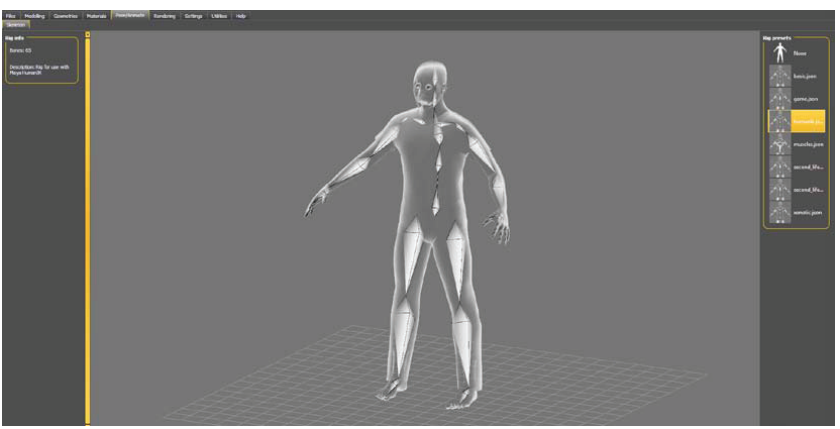

Рис. 5.2. Оснащенный персонаж,
созданный с помощью программы MakeHuman

Более подробную информацию о создании оснащенных персонажей для Unity можно найти в Интернете по адресу https://www.youtube.com/watch?v=IfIzMxdvEtQ.

Законченный проект Unity для этого раздела можно найти в файлах, прилагаемых к этой книге, он включает в себя оснащенный в MakeHuman персонаж и находится в папке Chapter05/char_anim_walk_end. Вы также можете использовать предварительно подготовленные персонажи из пакета активов персонажей Unity 5. Для импорта этого пакета выполните **Assets → Import Package → Characters**.

Импорт оснащенных персонажей

Импорт оснащенных персонажей в Unity представляет собой простое перетаскивание. Перетащите файл персонажа формата FBX из проводника Windows (или Mac Finder) в панель проекта, и Unity загрузит меш и его оснащение автоматически. При этом меш должен отобразиться в панели предварительного просмотра. Однако, как мы узнаем позднее, можно импортировать только оснастку и данные анимации без меша. Для приводимого здесь примера мы будем использовать созданный в MakeHuman оснащенный персонаж, включенный в файлы, прилагаемые к этой книге, находящийся в папке Chapter05/Char_Anim. Этот меш показан на рис. 5.3.

Первый делом после импорта оснащенного персонажа надо установить его масштаб, так как впоследствии изменение коэффициента масштабирования персонажа может привести к повреждению или несоответствию другим параметрам меша. Чтобы сделать это, перетащите меш персонажа в сцену и проверьте пропорциональность его размеров по отношению к остальной части вашей сцены. В идеале желательно придерживаться *реальных единиц измерения*; то есть использовать метр в качестве единицы измерения, так как основная единица измерения Unity соответствует 1 м. Вы должны изменить масштабный коэффициент так, если это необходимо, чтобы ваш меш имел реальные для сцены размеры (рис. 5.4). Это будет важно и для физических расчетов, и для расчетов освещения.

Затем переключитесь на вкладку оснащения **Rig** в инспекторе объектов и в качестве типа анимации в раскрывающемся списке выберите тип анимации **Humanoid**, если это еще не сделано. По умолчанию этим значением, вероятно, будет тип анимации **Generic**, если меш содержит оснащение, и тип анимации **None**, если меш не содержит оснащения. Тип анимации **Generic** используется для оснащения мешей, не относящихся к гуманоидным персонажам, например для кранов,

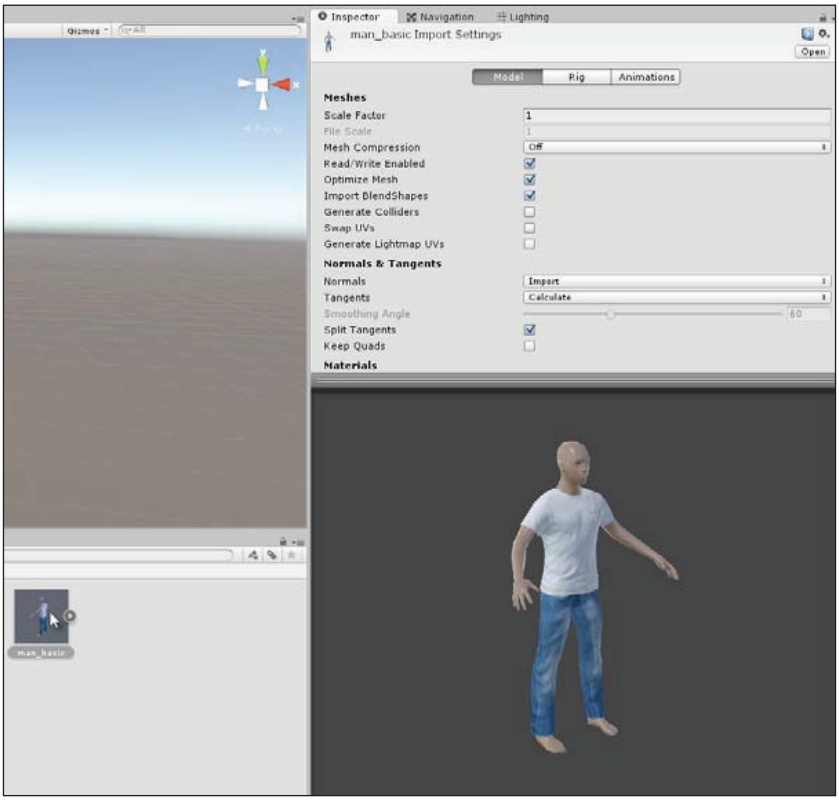

Рис. 5.3. Импорт оснащенного персонажа в Unity

трехглавых чудовищ, змей, деревьев и др. Тип анимации **Humanoid** является рекомендуемым выбором для человекоподобных персонажей, потому что дает нам доступ ко всем функциям анимации в системе Mecanim, как мы это увидим позже. После выбора типа анимации **Humanoid** щелкните по кнопке **Apply** (рис. 5.5).

При нажатии на кнопку **Apply** для подтверждения выбора оснастки с типом анимации **Humanoid** галочка или крест появится рядом с кнопкой **Configure** в инспекторе объектов. Это индикатор того, успешно ли система Mecanim идентифицировала оснащение гуманоидного персонажа внутри меша. Для меша, предоставляемого с книгой, и для мешей активов из примеров Unity успешно обнаружит данные оснащения персонажа. Но если ваши собственные меши по-

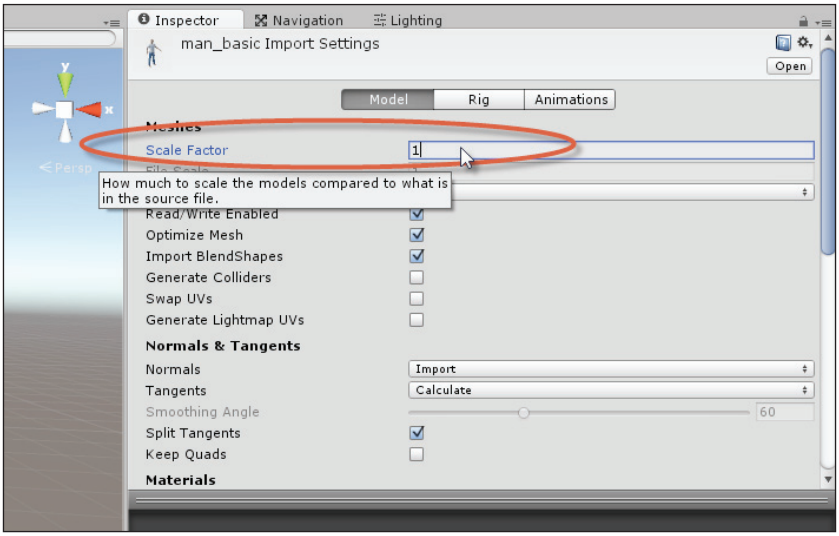

Рис. 5.4. Установка масштаба для персонажа

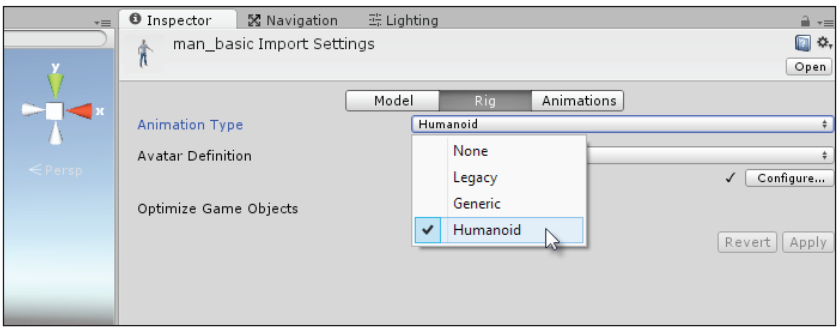

Рис. 5.5. Выберите опцию **Humanoid** из меню **Animation Type**

терпят неудачу, их можно будет настроить вручную. Далее мы увидим, как это сделать.

Аватары и ретаргейтинг

Выбор типа анимации оснастки **Humanoid** для импортированного меша персонажа означает, что Unity будет автоматически пытать-

ся настроить меш специальным образом под требования системы Mecanim. Приложение Unity делает это, используя структуру данных, называемую **аватар** (Avatar). Процесс автоконфигурирования может завершиться либо успешно, либо неудачей. Если процесс автоконфигурирования завершился неудачей, то вам нужно будет настроить аватар меша вручную. Для этого щелкните кнопку настройки **Configure**, чтобы отрыть редактор аватара. Приложение Unity при этом переключится в другой режим и, возможно, предложит сохранить текущую сцену. Заметим, что даже если процесс автоматической генерации аватара завершился успешно, вы, при необходимости, можете вручную подстроить или изменить аватар персонажа. Поэтому не пропускайте эту секцию, даже если с мешем вашего персонажа не возникло никаких вопросов.

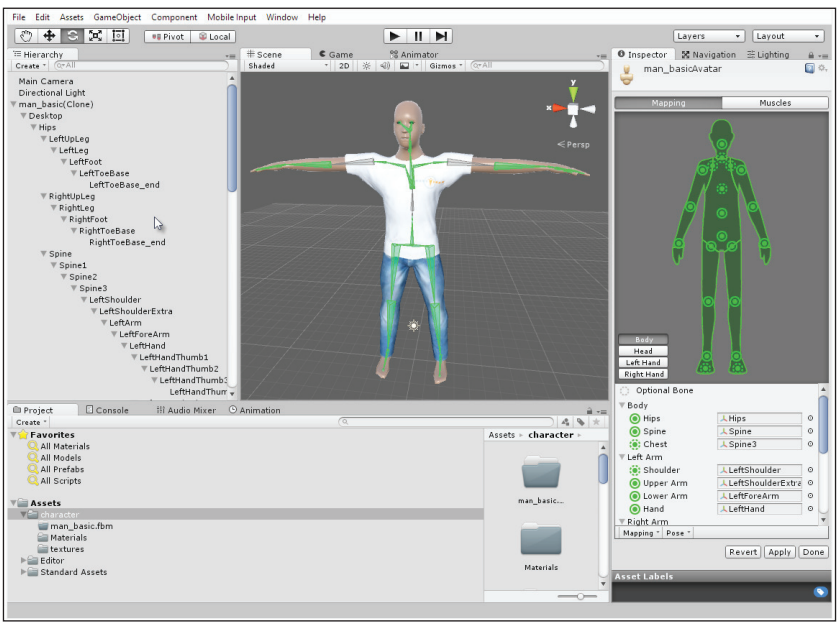

Рис. 5.6. Использование редактора аватара для настройки персонажа

Редактор аватара охватывает сразу три интерфейсные панели: панель иерархии, панель сцены и панель инспектора объектов (как это показано в предыдущем скриншоте). Вместе они содержат все данные

редактора аватара. В панели иерархии перечислены все кости меша и отражена их структура. Панель сцены содержит изображение меша персонажа вместе с изображениями костей. Панель инспектора объектов по умолчанию отображает схему человеческого тела. Задача редактора аватара состоит в том, чтобы обеспечить соответствие костей меша и их иерархии карте человеческого тела в панели инспектора объектов. Выделенные зеленым цветом кости в панели были автоматически правильно сопоставлены Unity с аватаром. Не выделенные цветом кости были проигнорированы. Кости, выделенные красным цветом, вызвали проблему: либо они неправильно связаны с картой тела, либо они не сориентированы и не выровнены должным образом. Чтобы проверить соответствие костей меша карте, щелкните по любому из круглых значков на карте тела. При этом произойдёт выбор объекта соответствующей кости в панелях сцены и иерархии. С помощью щелчка по любому кружку вы сможете проверить правильность сопоставления кости выбранному положению на карте тела.

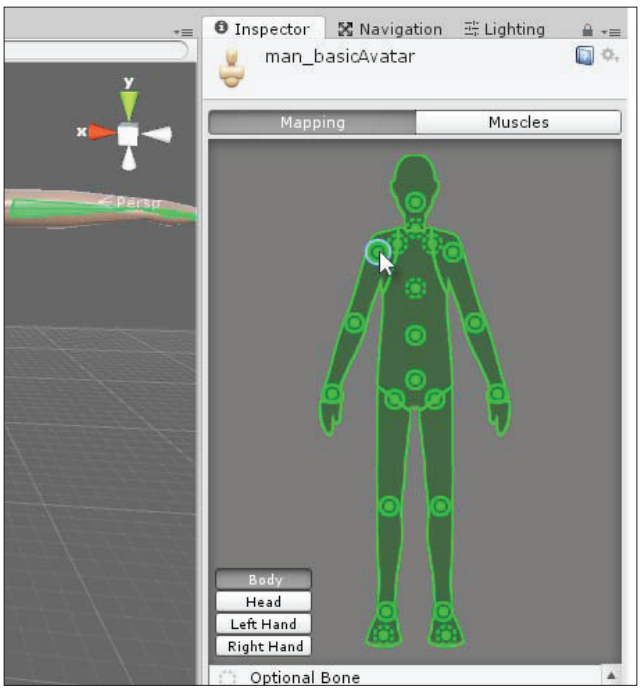

Рис. 5.7. Карта костей для тела

Вы можете переназначить любую кость меша ее правильному положению на карте, перетащив объект кости из панели иерархии в область, соответствующую кости, в инспекторе объектов.

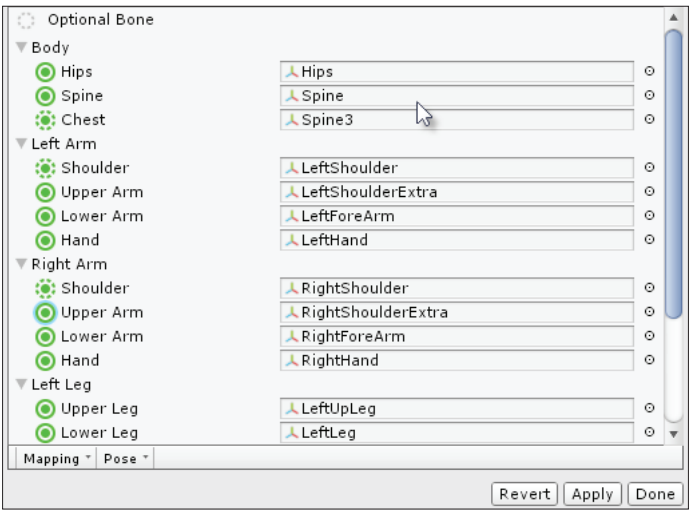

Рис. 5.8. Переназначение костей там, где это необходимо

Отсутствие окрашенных красным цветом костей в редакторе аватара еще не означает, что кости правильно связаны с сеткой и их деформации и анимация оправдают ваши ожидания. Следовательно, имеет смысл выполнить стресс-тест меша с помощью вкладки мышц **Muscles** инспектора объектов. Для доступа к ней выберите закладку **Muscles** (рис. 5.9).

При открытии вкладки мышц **Muscles** персонаж поменяет позу в окне просмотра. Кроме того, в инспекторе объектов отобразится новый набор опций. Используя ползунки секции **Muscle Group Preview**, вы можете перевести персонаж из нейтральной позы в экстремальную. Это важно, потому что вы сможете увидеть, как скелет персонажа деформируется во всем спектре поз, что позволит проверить деформации в точках экстремума (рис. 5.10). А это означает, что вы сможете быстро определить потенциальные проблемы при деформации.

Если ваш персонаж не деформируется в экстремальных позах должным образом, вы можете раскрыть группу регуляторов **Per-**

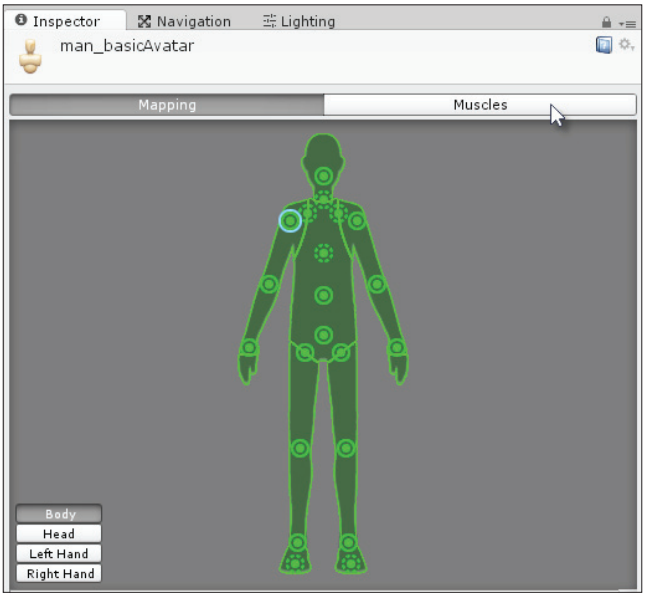

Рис. 5.9. Доступ к закладке мышц **Muscles** для стресс-теста меша

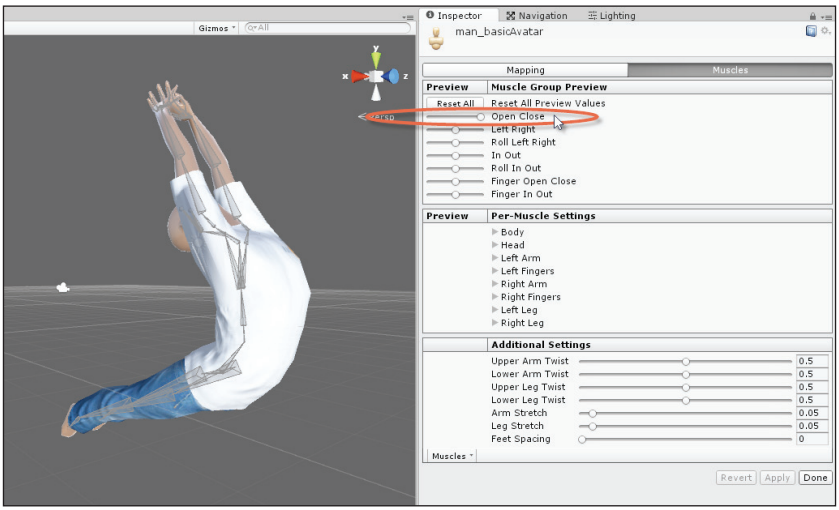

Рис. 5.10. Придание персонажам экстремальных поз для тестирования их потенциальных деформаций

Muscle Settings и задать максимальные и минимальные пределы деформаций. Другими словами, регуляторы контролируют диапазон, в которых скелет влияет на меш, что позволяет уменьшить или увеличить пределы деформаций.

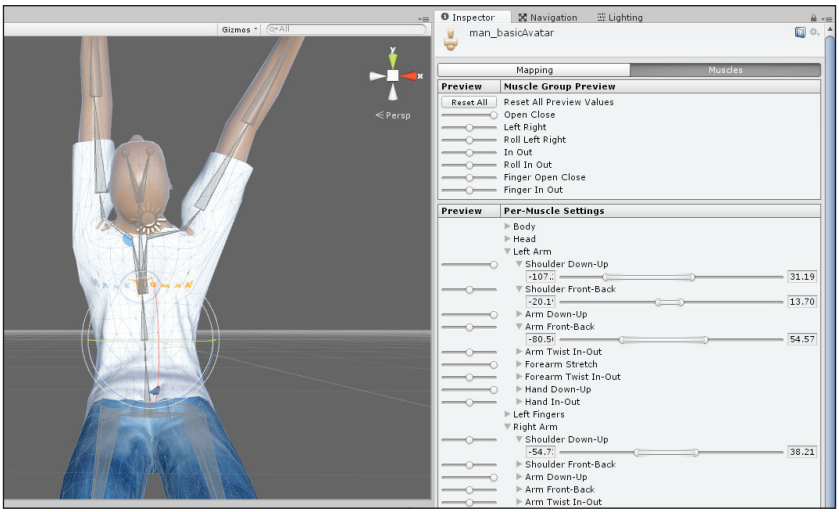

Рис. 5.11. Корректировка или настройка экстремальных поз с помощью закладки **Muscles**

Когда ваш аватар работает и деформации вашего персонажа соответствуют норме, вы можете щелкнуть кнопку **Apply**. При этом конфигурация аватара будет применена к файлу меша. Правильная настройка аватара важна для системы Mecanim, потому что это позволит выполнить ретаргейтинг. Это значит, что *любой* персонаж с правильно настроенным аватаром может быть анимирован с помощью оснастки и анимации любого другого персонажа с правильно настроенным аватаром. Все настройки и анимации полностью взаимозаменяемы между всеми мешами с настройками аватара гуманоида, отсюда и название **ретаргейтинг** (Retargeting) – перенацеливание, потому анимация от одного меша может быть перенацелена для работы с другим мешем. Это позволит вам повторно использовать все анимации персонажей. Например, вы можете сделать только один цикл ходьбы, а затем поделиться им со всеми ходящими персонажами.

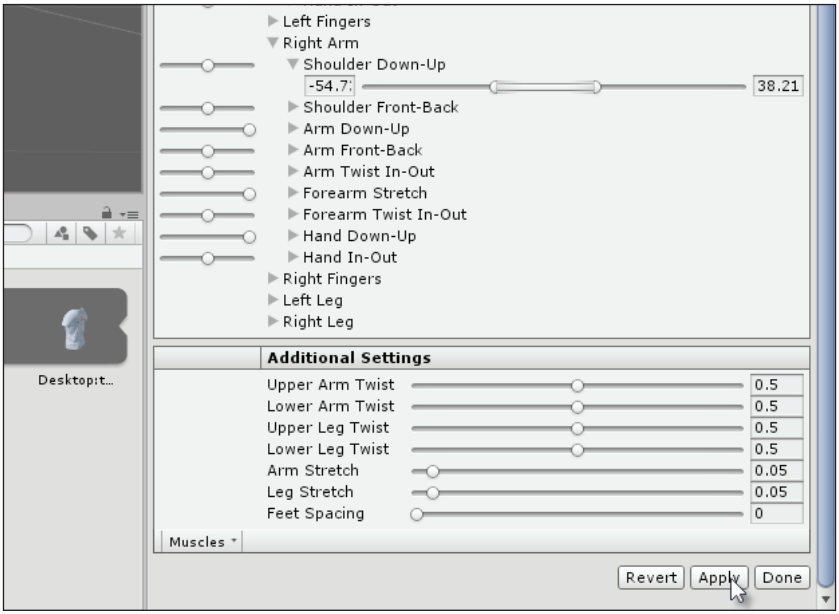

Рис. 5.12. Щелкните кнопку **Apply** в редакторе аватара для создания аватара меша персонажа

Ретаргейтинг анимации

У персонажей, включенных в файлы для этой книги, отсутствуют данные анимации, хотя и есть оснащение. Это не проблема благодаря аватарам и ретаргейтингу. Как уже упоминалось, можно взять анимации у одного персонажа или в файле и применить его к любому другому персонажу с правильно настроенным аватаром. В этом разделе будет показано, как такие анимации работают совместно. Для начала импортируем пакет активов персонажей. Он включает в себя контроллеры от первого и третьего лиц вместе с анимациями персонажей для циклов ходьбы и бега. Для доступа к этому пакету выполните **Assets → Import Package → Characters** из меню приложения, как показано на рис. 5.13.

Анимации, включенные в пакет, можно найти в панели проекта, выполнив **Standard Assets → Characters → ThirdPersonCharacter → Animation**. Анимации, включенные в файлы формата FBX, содержат оснащение и данные анимации, но в них отсутствуют меши. Для до-

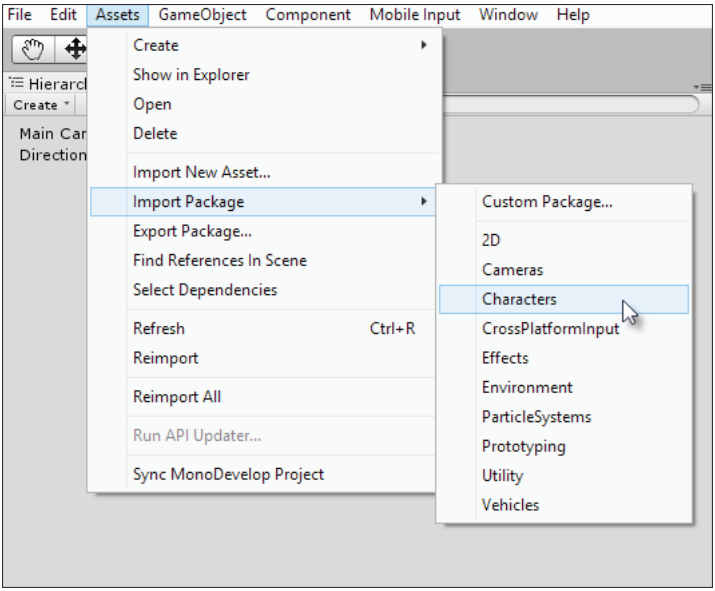

Рис. 5.13. Импорт пакета активов персонажей с анимациями

ступа к данным анимации раскроем файл меша и выберем анимационный клип, как это показано на рис. 5.14.

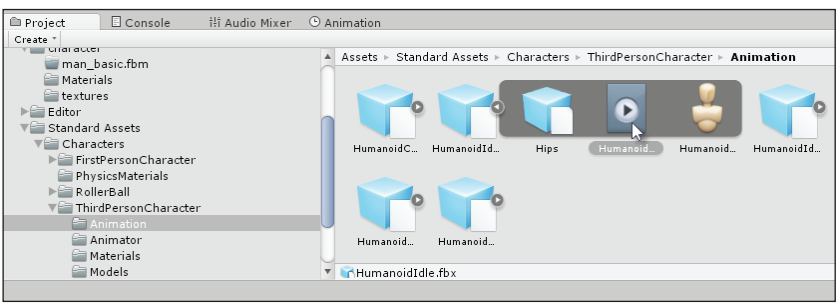

Рис. 5.14. Выбор анимационного клипа для ретаргейтинга

Большинство анимаций могут быть просмотрены в инспекторе объектов с помощью модели Unity по умолчанию. При нажатии кнопки **Play** в панели инструментов можно просмотреть анимацию персонажа (рис. 5.15).

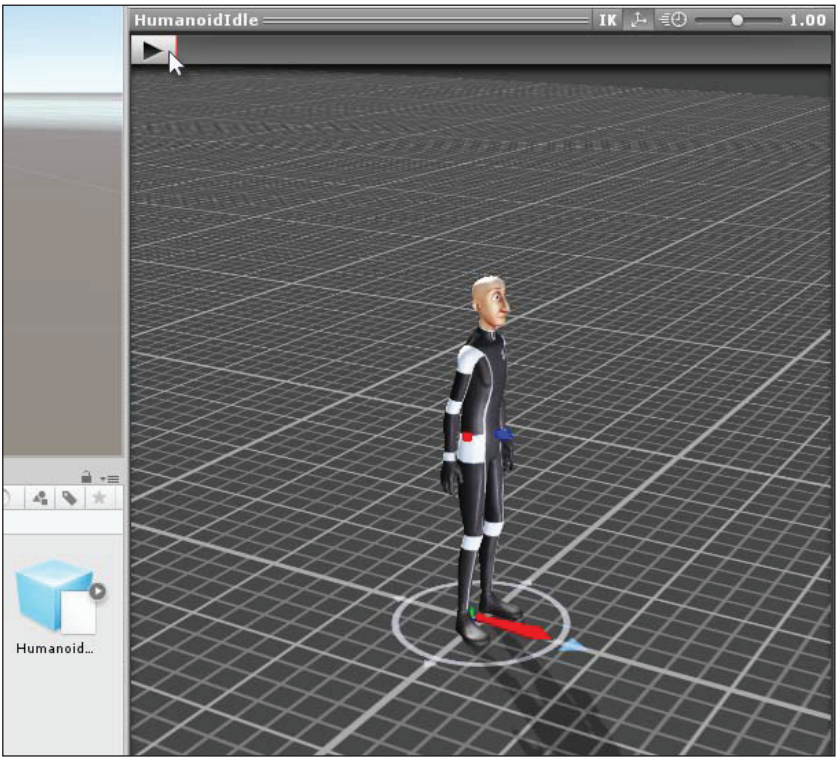

Рис. 5.15. Предварительный просмотр анимации персонажа

Вы также можете посмотреть на ретаргейтинг в действии на примере пробы анимации для вашего импортированного персонажа. Измените модель персонажа для предварительного просмотра, нажав на иконку аватара в нижнем правом углу, и выберите модель для предварительного просмотра (рис. 5.16).

После нажатия на иконку аватара выберите опцию **Other** и затем выберите модель персонажа из панели проекта. Панель предварительного просмотра в инспекторе объектов обновится, чтобы показать анимацию, примененную к персонажу. Если она выглядит хорошо, то аватар успешно настроен для ретаргейтинга (рис. 5.17).

Давайте опробуем модель в комплекте с анимацией в реальной сцене. Для начала перетащите меш персонажа из панели проекта в сцену и расположите его в центе мира (0, 0, 0). Unity определит, что вновь добавленный персонаж, в принципе, может быть анимирован, поэто-

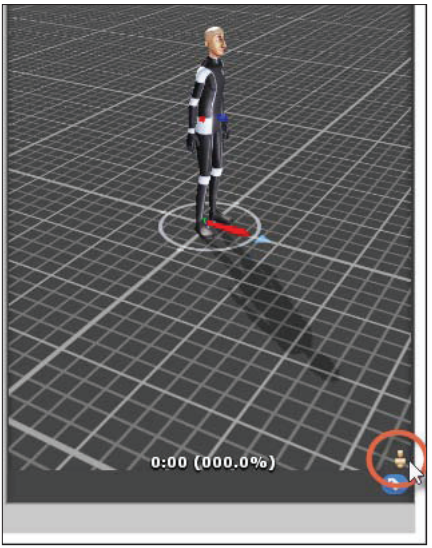

Рис. 5.16. Выбор аватара

Рис. 5.17. Ретаргейтинг в действии

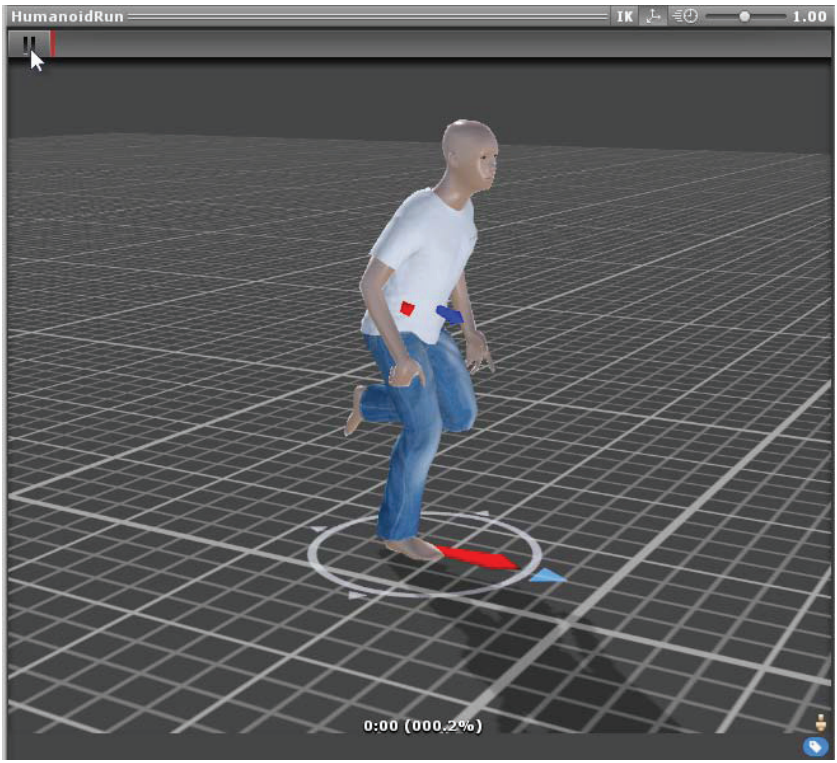

Рис. 5.18. Позиционирование персонажа в сцене

му к нему автоматически будет добавлен компонент **Animator**, хотя пока объект и не имеет ни контроллера, ни анимации.

Теперь мы назначим персонажу анимацию ходьбы. Чтобы сделать это, найдите анимацию для ходьбы в пакете активов персонажей, а затем перетащите ее на меш персонажа в сцене. Анимацию ходьбы можно найти, перейдя **Standard Assets → ThirdPersonCharacter → Animation**. Имя файла HumanoidWalk.fbx. Когда вы сделаете это, Unity автоматически настроит контроллер анимации и назначит его компоненту аниматора (рис. 5.19).

Настройте камеру сцены, чтобы получить хороший обзор персонажа, и создайте плоскость пола. Затем нажмите кнопку **Play** на панели инструментов. Персонаж сразу же пойдет. Поздравляю! Вы настроили свой первый персонаж для анимации.

Рис. 5.19. Присоединение анимации ходьбы к персонажу

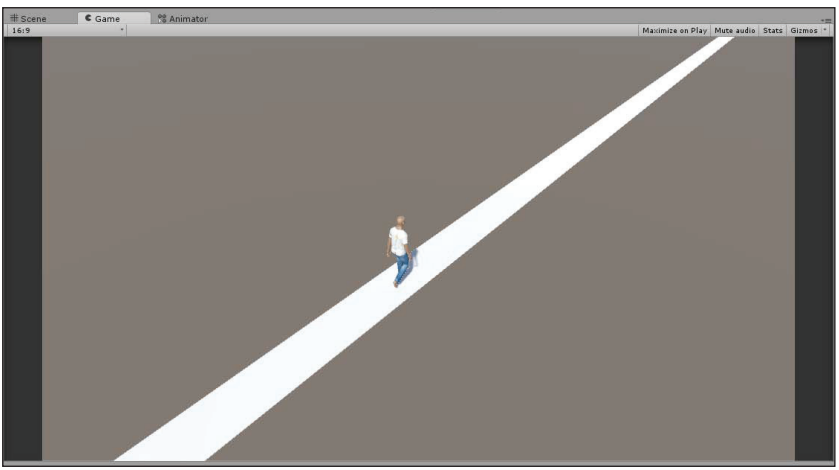

Рис. 5.20. Создание ходящего персонажа

Готовый проект на этой стадии вы можете найти в прилагаемых к этой книге файлах в папке Chapter05/char_anim_walk_start.

Перемещение корневого объекта

Обратимся к предыдущему примеру, где мы создали сцену, в которой к импортированному персонажу применена анимация ходьбы из пакета персонажей Unity. При нажатии клавиши **Play** персонаж не только воспроизводит анимацию ходьбы, но и на самом деле идет, перемещаясь вперед в сцене, а не выполняет ходьбу на месте. Другими словами, *воспроизводится* анимация ходьбы, и персонаж действительно *движется* относительно его положения в сцене. Это происходит из-за **перемещения корневого объекта** (root motion). Перемещение корневого объекта применимо к любой анимации и изменяет положение или поворачивает объект самого верхнего уровня (корневой) в иерархии объектов меша. Импортируемые меши персонажей содержат много дочерних объектов для своих костей и частей. Но анимация применяется к объекту верхнего уровня, рассматриваемого как корневой объект для движения. По умолчанию перемещение корневого объекта для персонажа включено, а это значит, что он будет перемещаться при воспроизведении анимации. Однако можно отключить перемещение корневого объекта, в результате чего персонаж будет ходить на месте, то есть ходить, не сходя с места. Чтобы сделать это, выберите персонаж в сцене и отключите флажок **Apply Root Motion** компонента **Animator** в инспекторе объектов (рис. 5.21).

Возникает вопрос: когда перемещение корневого объекта должно быть включено или когда отключено. Перемещение корневого объекта, как правило, делает движение персонажа внешне более реалистичным, потому что с помощью доступа к данным кривой анимации движение персонажа можно интерполировать более эффективно. Однако отключение перемещения корневого объекта делает движения персонажа более быстрыми и лучше поддающимися управлению, это подходит для аркадных и других активных игр. В конечном счете решение о том, когда использовать перемещение корневого объекта, а когда нет, зависит от того, что лучше подойдет вашей игре.

Исправление смещения при движении

Общей проблемой при импорте анимации движения с перемещением корневого объекта в Unity является смещение. Чтобы увидеть своими глазами эту проблему, запустите сцену анимации ходьбы, которую

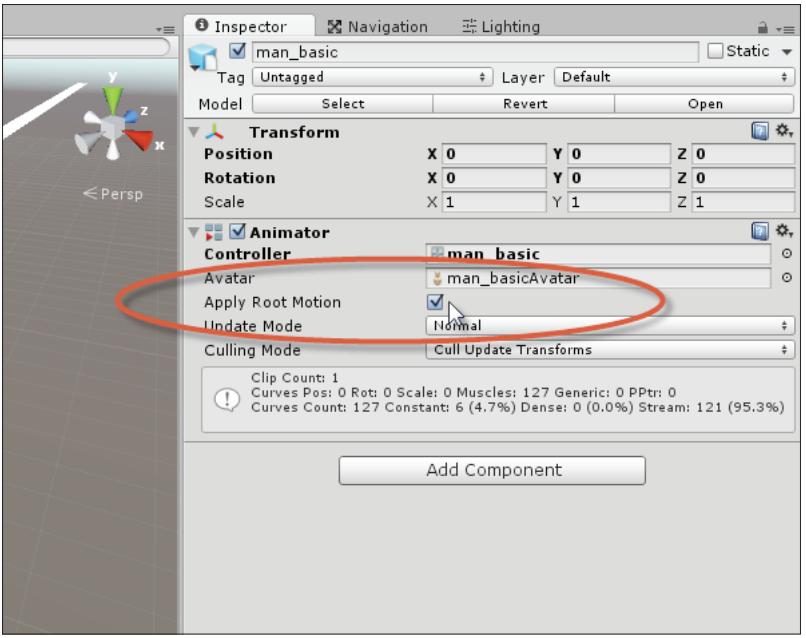

Рис. 5.21. Отключение перемещения корневого объекта персонажа для ходьбы на месте

вы только что создали, или загрузите ее из сопроводительных файлов этой книги из папки Chapter05/char_anim_walk_start. Когда вы запустите этот проект и понаблюдаете за ходьбой персонажа с включенным перемещением корневого объекта, вы увидите, что он не просто идет вперед, а постепенно сворачивает или сбивается с курса. Это потому, что он идет не идеально по прямой. Существует определенная степень отклонения или смещения, которая заставляет его траекторию медленно удаляться от прямой. Вначале это не особенно заметно, но по истечении большого промежутка времени персонаж резко отклонится от своей первоначальной траектории (рис. 5.22).

Эта проблема не связана со сценой или персонажем, она заключена в анимации перемещения корневого объекта. Чтобы исправить это, вы должны выбрать файл анимации в панели проекта для отображения его свойств. Для примера анимации ходьбы, используемого здесь, файл анимации можно найти, перейдя **Standard Assets → ThirdPersonCharacter → Animation**. Выберите файл HumanoidWalk.fbx. В панели проекта выберите вкладку анимации **Animation** для

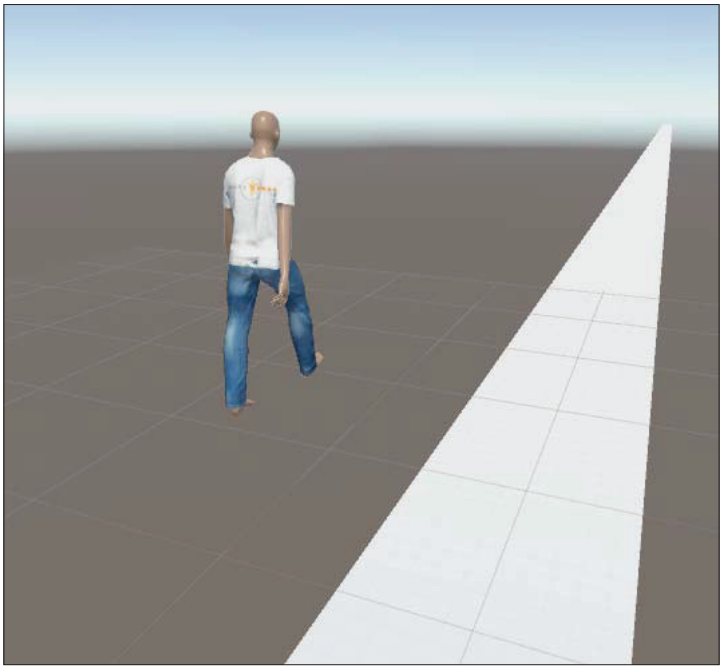

Рис. 5.22. Девиация перемещения корневого объекта

просмотра данных анимации в файле (рис. 5.23). Если вы воспроизведете файл в панели предварительного просмотра, то увидите проблему смещения и там.

Проблема связана с угловой скоростью. Ее значение отображено в инспекторе объектов. Вектор угловой скорости **Average Velocity** указывает направление и ориентацию персонажа на протяжении всего срока анимации. По умолчанию для анимации ходьбы это значение включает в себя незначительное отрицательное значение *x* составляющей скорости. Следовательно, модель отклоняется от курса с течением времени (рис. 5.24).

Чтобы исправить эту проблему, установите флажок **Bake Into Pose** для секции **Root Transform Rotation** и изменяйте поля смещения, пока среднее значение скорости параметра *x* не станет равным 0. Значение *z* должно оставаться ненулевым, так как персонаж должен идти вперед. Для изменения этих параметров вам, возможно, придется нажать кнопку **Edit** в верхнем правом углу инспектора объектов при выборе меша (рис. 5.25).

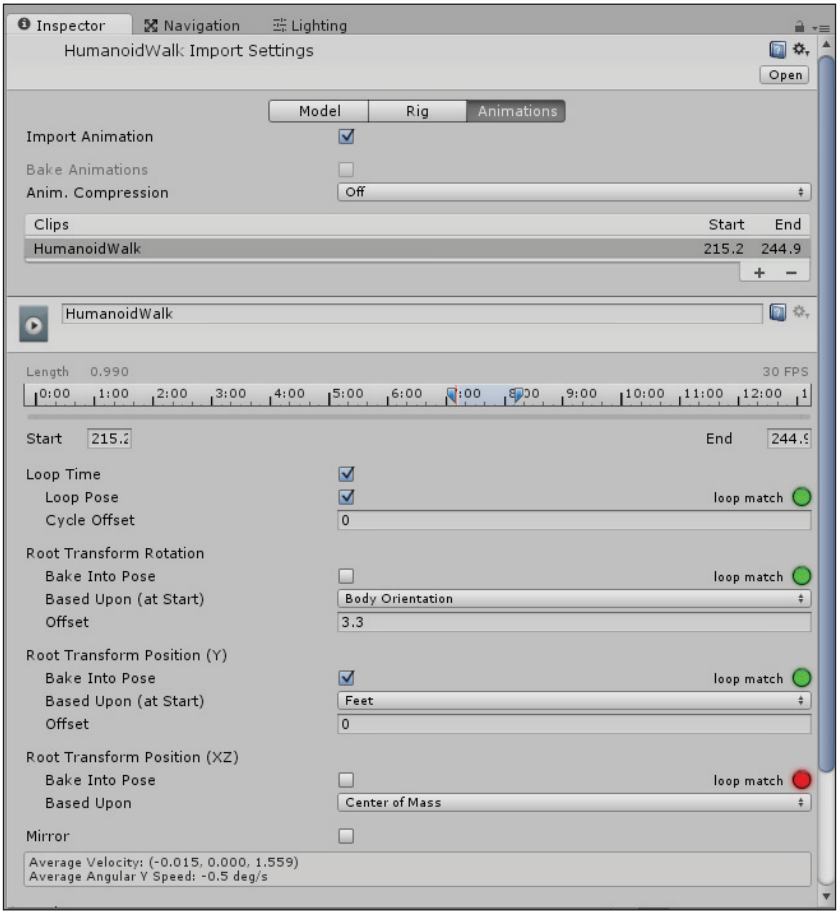

Рис. 5.23. Просмотр данных анимации для выбранного файла

Наконец, если окажется, что ваши персонажи ходят под полом или над полом, то есть либо выше, либо ниже, чем это должны быть, вы должны изменить значения **Root Transform Position (Y)**, установив значение **Feet** для поля **Based Upon (at Start)**, рис. 5.26. Это гарантирует, что положение корневого объекта вашего персонажа будет привязано к костям ног.

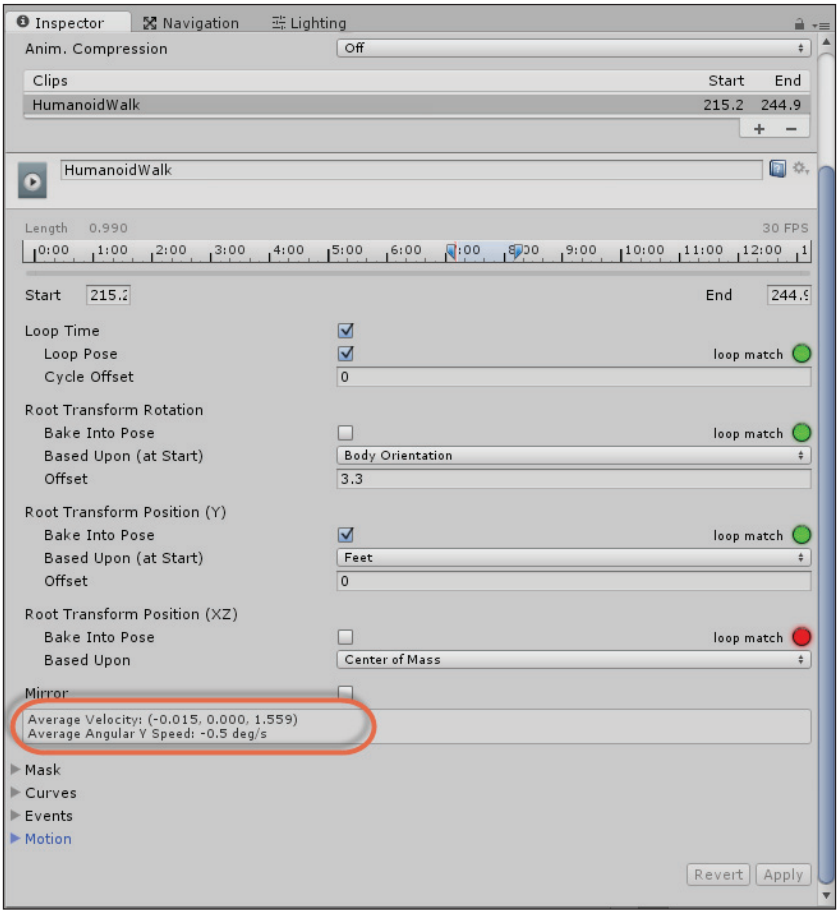

Рис. 5.24. Проверка средней скорости движения при перемещении корневого объекта

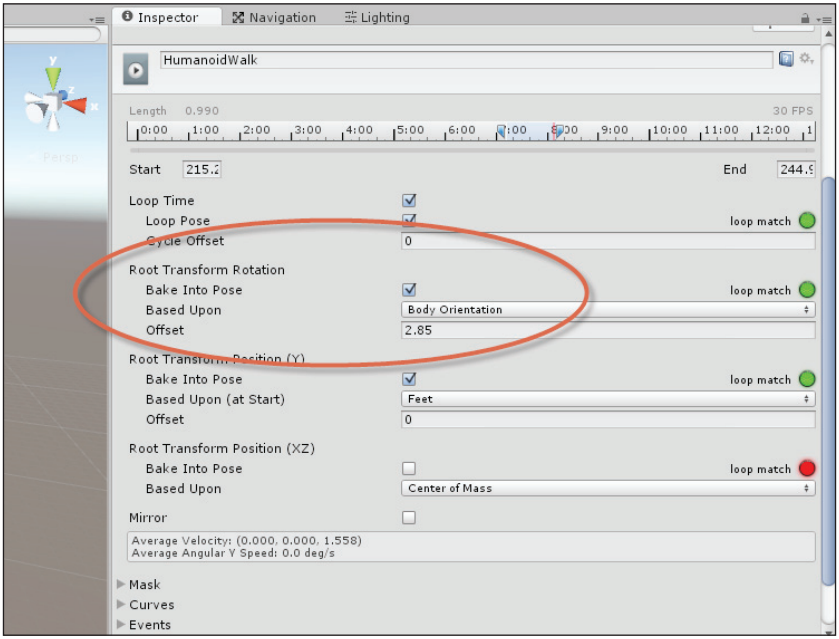

Рис. 5.25. Настройка смещения перемещения ключевого объекта с помощью **Bake Into Pose**

Итоги

Поздравляю! Теперь вы освоили надежный и универсальный процесс импорта для добавления оснащенного меша персонажа в Unity. Они уже сотрудничают с системой Аватар и могут использовать готовые анимации из множества других похожих мешей. Но это лишь половина истории. Система Mecanim не только позволяет импортировать и настраивать многократно используемые персонажи для анимации, как мы это делали до сих пор, но и дает возможность определить более сложные контроллеры анимации, что позволит смешивать анимации и реагировать на действия пользователя. Эта тема будет рассмотрена в следующей главе.

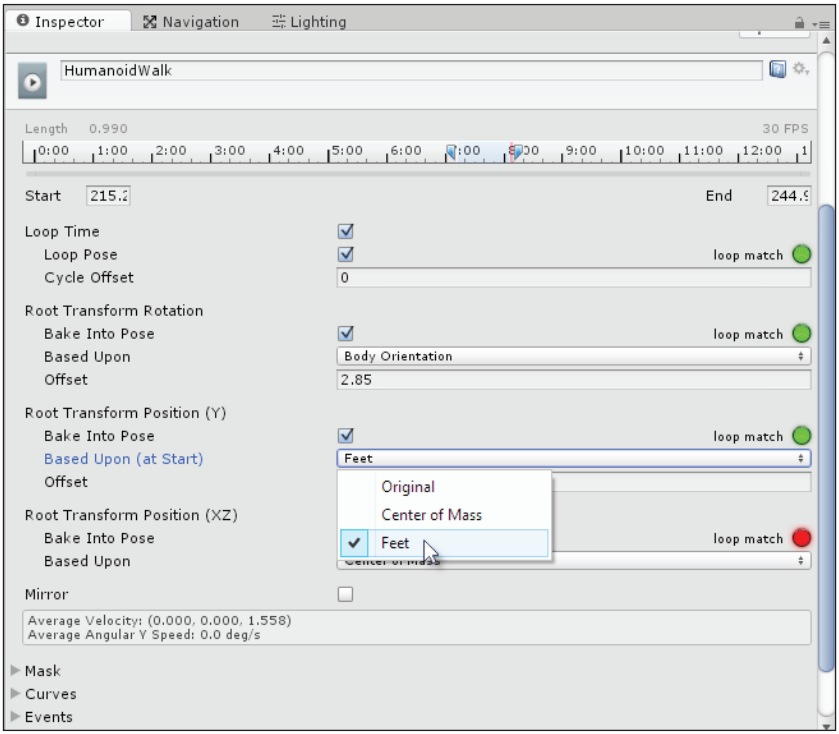

Рис. 5.26. Настройка положения ноги персонажа для анимации

Глава 6

Продвинутая анимация персонажей

В предыдущей главе мы полностью рассмотрели важный процесс импорта оснащенных анимированных моделей персонажей в Unity и их настройки для эффективной работы с системой Mecanim. Создав аватар для импортированного персонажа, мы сможем легко применить к нему множество различных гуманоидных анимаций из разных источников. Но это только часть возможностей системы Mecanim, с ее помощью можно достичь гораздо большего. После того как мы импортировали и настроили персонаж, становится доступным множество возможностей управления анимацией и придания ей интерактивности, их применение позволит создавать весьма реалистичные персонажи. В этой главе мы воспользуемся ими для создания управляемого игроком персонажа.

Создание управляемого персонажа

В этой главе будет описано, как создать управляемого игроком персонажа с помощью системы Mecanim с нуля. Персонаж при запуске игры будет находиться в позе ожидания, а затем при помощи клавиш **WASD** мы заставим его ходить, бегать и поворачиваться там, где это необходимо. Этот проект начнется со специально подготовленного проекта Start, включенного в прилагаемые к этой книге файлы. Данный проект содержит только пустую сцену, импортированную модель персонажа MakeHuman и встроенные активы для анимации персонажей, включенные в пакет активов персонажей. Эти анимации включают в себя ходьбу, бег, ожидание и анимации поворотов (как было описано в предыдущей главе). Они изначально были предназначены для другой модели персонажа, но, как мы уже знаем, система Аватар Unity позволяет перенацеливать анимации на множество разных моделей.

Рис. 6.1. Создание управляемого игроком персонажа с помощью системы Mecanim

⚠ Проект Start для этой главы вы можете найти в прилагаемых к этой книге файлах в папке Chapter06/Start. Окончательный проект можно найти в папке Chapter06/End.

Для того чтобы начать работу по созданию анимированного персонажа, убедитесь, что вы импортировали гуманоидные анимации для ходьбы, бега, ожидания и поворотов (или используйте анимации, включенные в сопроводительный проект). Кроме того, убедитесь, что модель персонажа была импортирована и настроена как аватарсовместимый персонаж, как это было описано в предыдущей главе (или используйте модель персонажа MakeHuman, включенную в сопроводительный проект). Затем создайте новый актив **Animation Controller** из панели проекта, щелкнув правой кнопкой мыши и выполнив **Create → Animator Controller**. Актив **Animator Controller** будет нужен для управления игроком анимациями при помощи клавиатуры.

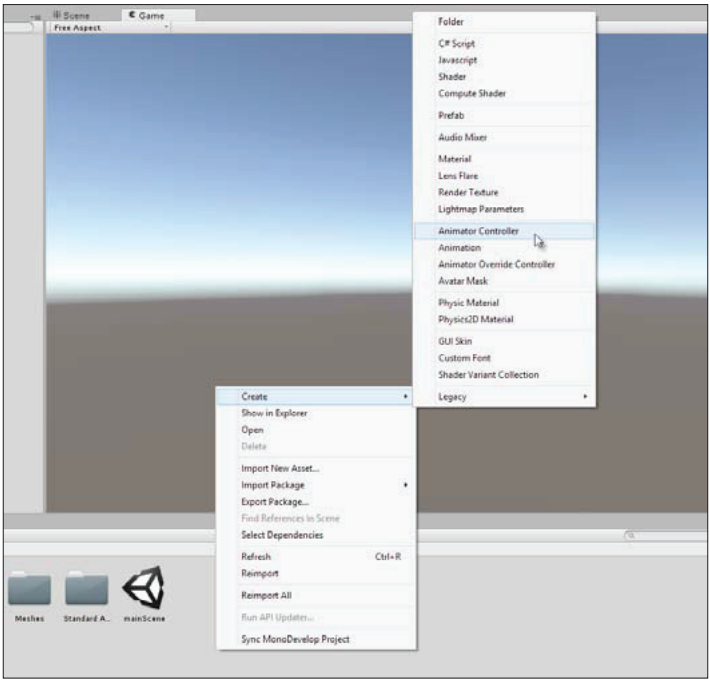

Рис. 6.2. Создание актива **Animator Controller**

Смешивание деревьев

Рис. 6.3. Персонаж для анимации MakeHuman

Тщательно продумав действия персонажа, показанного на следующем скриншоте, мы выделим несколько состояний, в которых он может находиться перед вводом команды игроком. В частности, когда игрок ничего не нажал, персонаж должен находиться в нейтральной позе ожидания, стоя на месте неподвижно.

Если игрок нажимает одну только клавишу влево или вправо, то персонаж должен оставаться там, где стоит, повернуться в соответствующем направлении. Если игрок нажимает клавишу вперед или вверх, то персонаж должен идти или

бежать вперед. Здесь мы определили набор состояний персонажа, таких как ожидание, ходьба, бег и поворот. Но все не так просто, потому что персонаж может повернуть *при* ходьбе или *при* беге. В этих случаях игрок задает два направления движения одновременно, и мы должны предусмотреть адекватную реакцию персонажа и в таких ситуациях. Должны воспроизводиться анимации ходьбы и поворота, но они должны непринужденно смешиваться с анимациями всех других состояний, чтобы внешне персонаж продолжал выглядеть органично. Чтобы добиться этого, мы используем смешивание деревьев, которые позволят нам плавно смешивать несколько гуманоидных анимаций. Чтобы создать смешивание деревьев, сначала откройте окно аниматора для нового актива **Animator Controller**. Чтобы сделать это, выполните **Window → Animator** из меню приложения, как показано на рис. 6.4.

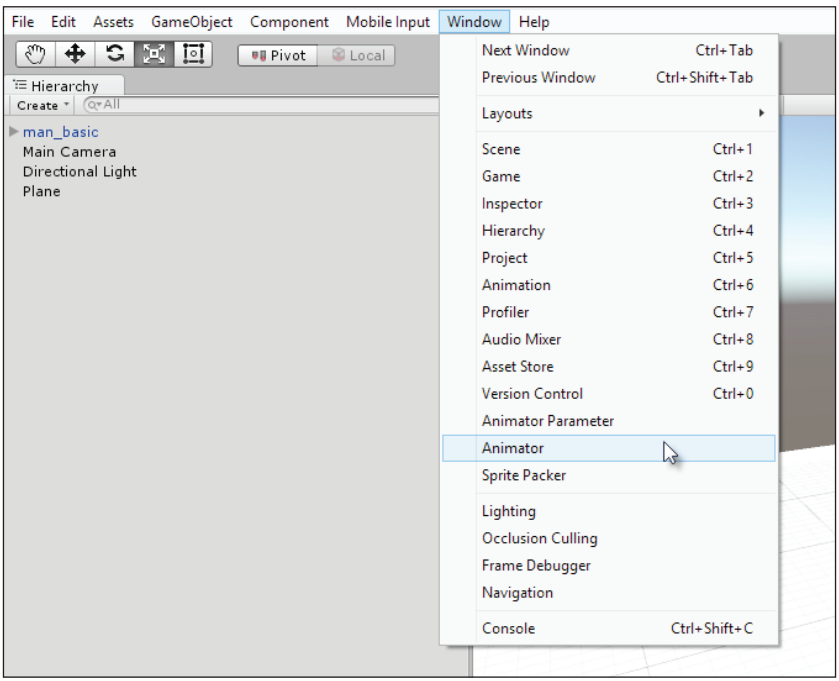

Рис. 6.4. Открытие окна аниматора

Для создания смешивания деревьев щелкните правой кнопкой мыши внутри графа аниматора и выполните **Create State → From New Blend Tree**, как показано на рис. 6.5.

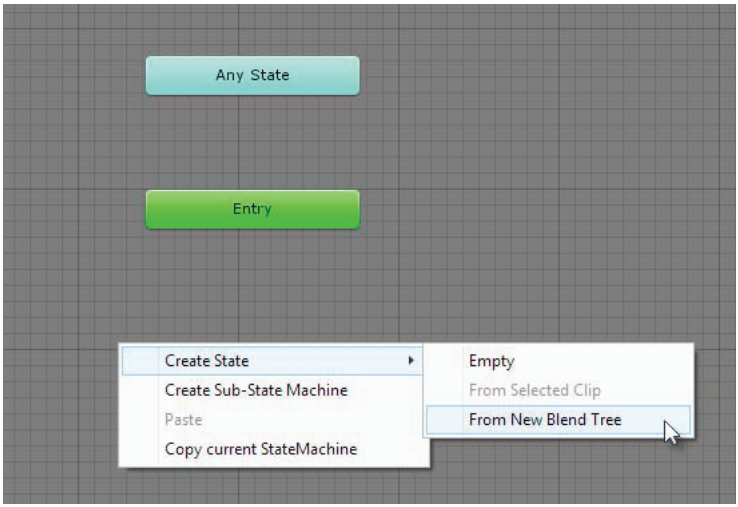

Рис. 6.5. Создание нового смешивания деревьев анимаций

После своего создания смешивание деревьев отображается как обычный узел анимации. Если это первый созданный узел, то он выделяется оранжевым цветом и будет подключен к узлу **Entry**, так как он автоматически становится узлом по умолчанию. Если узел смешивания деревьев не подключился автоматически, то щелкните правой кнопкой мыши на узле и выберите **Set as Layer Default State**, чтобы сделать его узлом по умолчанию. Это значит, что узел смешивания деревьев будет автоматически воспроизводиться при запуске игры и активации аниматора. Однако узел смешивания деревьев представляет собой более сложный узел, чем прочие узлы анимации. Чтобы выявить его возможности, дважды щелкните по нему левой кнопкой мыши. При двойном щелчке по узлу смешивания деревьев в окне аниматора отобразится новый режим, содержащий больше опций. С помощью интерфейса смешивания деревьев мы сможем собрать воедино сложную анимацию персонажей. Обратите внимание, мы можем использовать **полосу следов** в виде направленных кнопок с заостренным правым концом (Bread Crumb) в верхней части окна аниматора для возврата к предыдущему виду, нажав на кнопку базового слоя **Base Layer**.

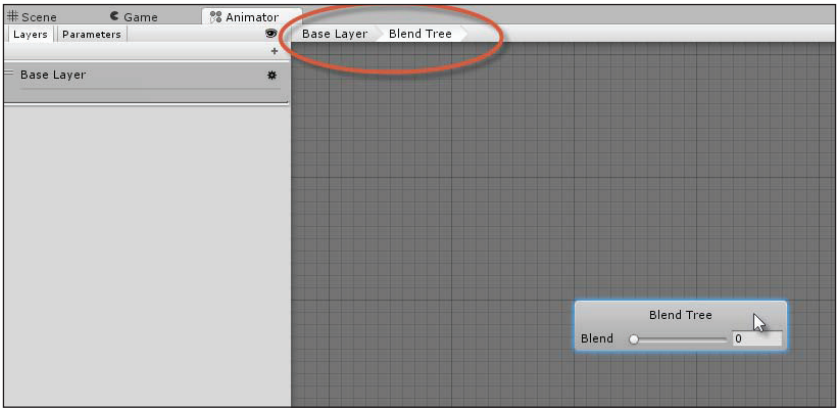

Рис. 6.6. Использование режима Blend Tree

Размерности

По умолчанию каждое смешивание деревьев будет одномерным. Другими словами, по умолчанию каждое смешивание деревьев смешивает линейную последовательность различных анимаций. Мы можем увидеть это в инспекторе объектов, приведенном на рис. 6.7.

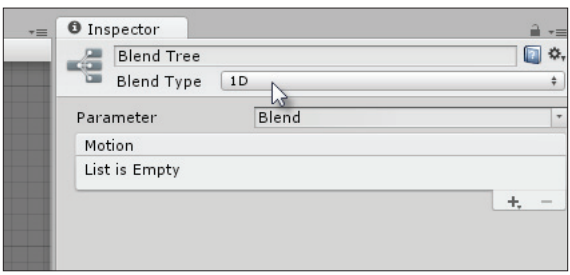

Рис. 6.7. Настройка размерности смешивания деревьев

Одномерные смешивания деревьев хорошо подходят для сценариев, где персонаж ходит только вперед и назад вдоль линии, сменяя анимации состояний ожидания, ходьбы и бега в соответствии с командами игрока. Но в нашем случае персонаж должен уметь еще и поворачивать, устремляясь в новом направлении. По сути, наш персонаж должен перемещаться по 2D-плоскости пола, и он реагирует на

вводы по двум осям (горизонтальной и вертикальной) вместо одной. Мы должны иметь возможность использовать клавиши вверх, вниз, влево и вправо для управления его движением в любом месте уровня. Поэтому сменим тип смешивания деревьев с **1D** на **2D Freeform Cartesian**.

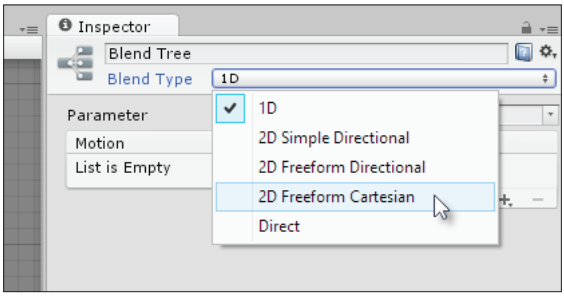

Рис. 6.8. Использование смешивания деревьев 2D

При создании смешивания деревьев 2D инспектор объектов позволяет определить поля движения, добавляемые к графу. **Поле движения** (motion field) просто присоединяется к анимации клипа, который должен быть частью смешивания деревьев. У персонажа в этом сценарии имеются анимации ожидания, ходьбы, бега и поворота, и поэтому должны быть поля движения для всех этих состояний. Давайте создадим несколько полей движения в графе. Для этого нажмите кнопку **+** в панели движения в инспекторе объектов, а затем выберите **Add Motion Field** из контекстного меню.

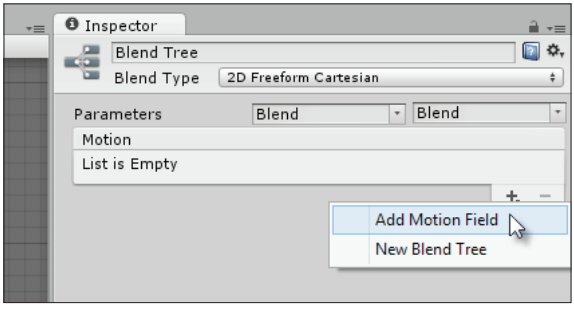

Рис. 6.9. Добавление полей движения

Нажмите на кнопку + девять раз, чтобы добавить девять полей движения. Они будут соответствовать следующим возможным состояниям персонажа: ожидание, ожидание – поворот налево, ожидание – поворот направо, ходьба, ходьба – поворот налево, ходьба – поворот направо, бег, бег – поворот налево и бег –поворот направо. Для каждого состояния требуется отдельный анимационный клип, и нам понадобится смешивание деревьев для смешивания их при получении команды игрока. При создании двух или более полей движения графическое представление их появляется в инспекторе объектов.

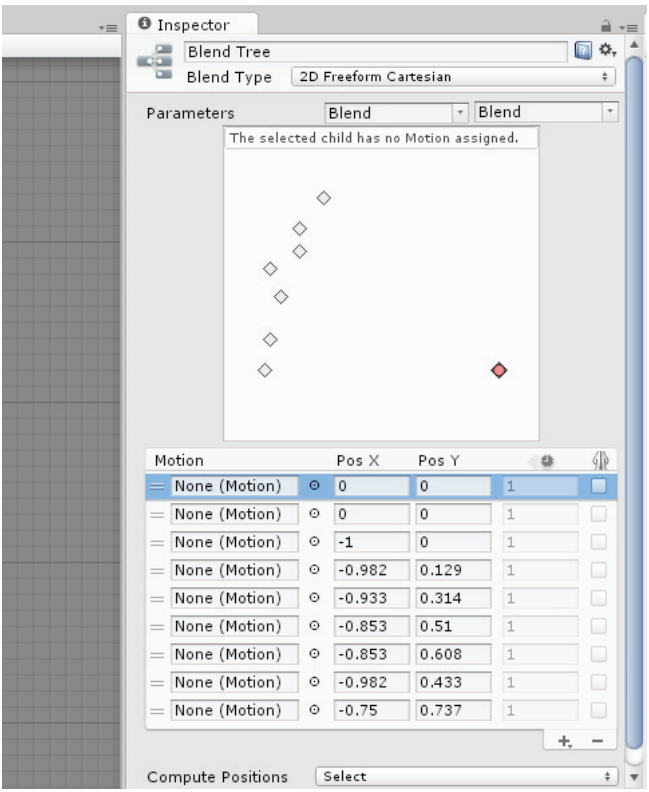

Рис. 6.10. Визуализация полей движения

Добавленные поля движения беспорядочно разбросаны внутри графа, и это недопустимо. Сейчас мы это исправим. Во-первых, давайте назначим полям анимационные ролики. Чтобы сделать это,

перейдите в папку **Assets** → **Characters** → **ThirdPersonController** → **Animation** и назначьте клипы каждому соответствующему слоту. Я сделал назначения, показанные на рис. 6.11.

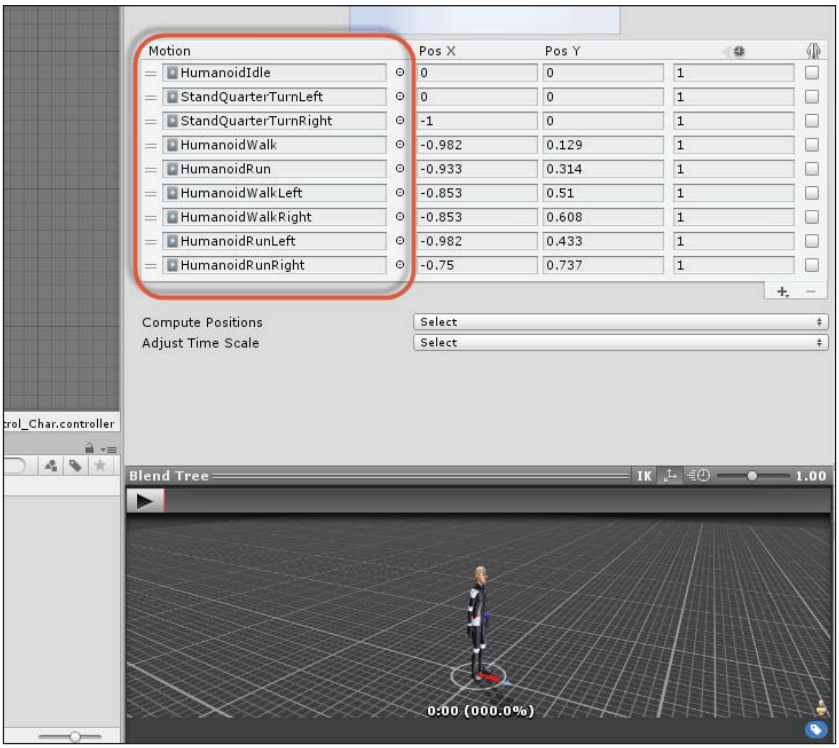

Рис. 6.11. Связывание анимационных клипов с полями движения

Анимационные клипы теперь назначены соответствующим полям движения, но сами поля в графе (он показан в инспекторе объектов) по-прежнему остаются разбросанными, а не расположенными систематически по строкам и столбцам. Давайте исправим это, изменив значения полей **Pos X** и **Pos Y** для каждого поля движения. Сделав это, мы подготовим поля движения для правильного приема 2D-входных данных. Рассмотрим этот скриншот (рис. 6.12).

Давайте более внимательно рассмотрим расположение полей движения. Состояние ожидания – нейтральная и неподвижная поза занимает в графе позицию (0, 0). Бег по прямой закодирован как (0, 1),

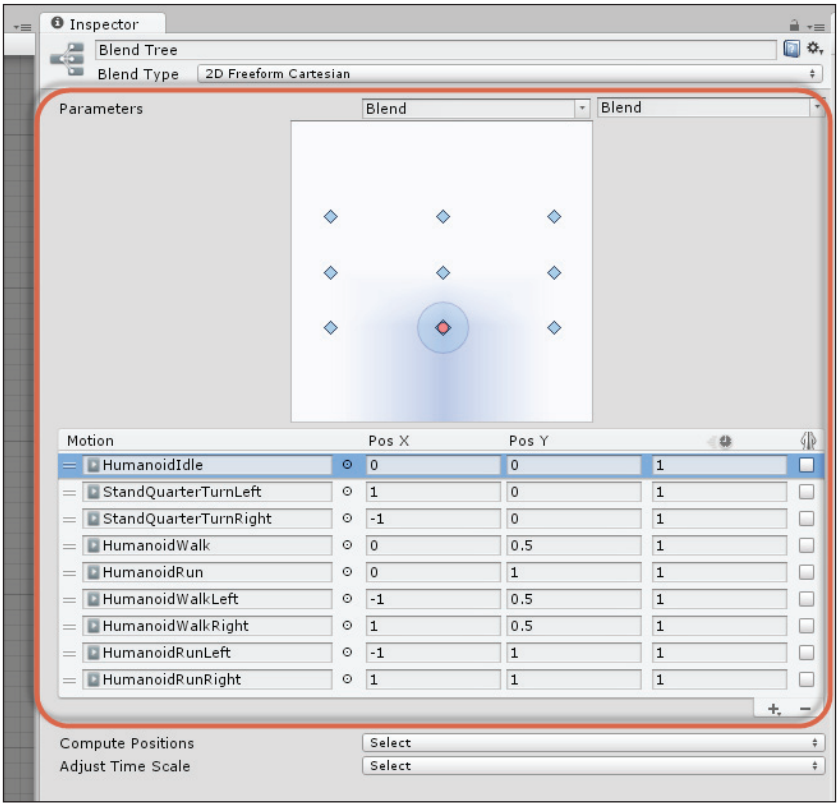

Рис. 6.12. Размещение полей движения внутри графа

а ходьба как (0, 0.5), что соответствует состоянию, находящемуся в середине между неподвижностью и бегом. У состояний поворота, напротив, значения по оси *x* достигают экстремумов. Таким образом, повороты представлены парами значений (–1, 1) (поворот налево) и (1, 1) (поворот направо).

Отображение десятичных чисел с дробной частью

Теперь для персонажа определены 2D-оси движения при смешивании деревьев. Далее мы должны создать два параметра – два десятичных числа с дробной частью, дающих нам контроль над смешиванием

анимаций из скрипта. Во-первых, создайте два десятичных значения с дробной частью, переключившись на вкладку параметров **Parameters** в окне аниматора **Animator**, как это показано ниже (рис. 6.13).

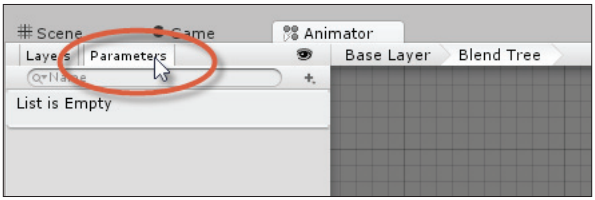

Рис. 6.13. Доступ к вкладке параметров **Parameters**

Нажмите кнопку **+**, чтобы добавить два новых параметра **Float** (десятичное число). Назовите их **Horz** (горизонтальный) и **Vert** (вертикальный) соответственно. Вместе значения этих параметров будут определять или задавать, какую анимацию воспроизвести для оснащенного персонажа при поступлении ввода.

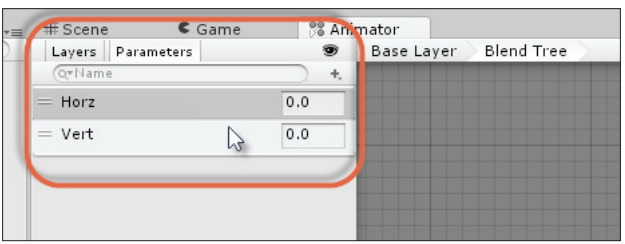

Рис. 6.14. Создание числового параметра с десятичной точкой

Теперь, когда мы создали два параметра, по одному для каждой из осей ввода игрока, давайте привяжем их к смешиванию деревьев в 2D-пространстве. Чтобы сделать это, заполните область параметров **Parameters** в инспекторе объектов. Щелкните по стрелке раскрывающегося списка первого из них и выберите ему значение **Horz** для отображения параметра **Horz** на ось *x*. Затем повторите эту процедуру для **Vert**, отобразив его на ось *y* (рис. 6.15).

После того как такое отображение установлено, можно, наконец, просмотреть анимации и смешивание деревьев как одно целое в панели предварительного просмотра в инспекторе объектов. Узел смешивания деревьев в графе аниматора теперь имеет два ползунка для установки параметров. Это даст вам возможность, перемещая их между экстремумами, протестировать комбинации смешивания деревьев (рис. 6.16).

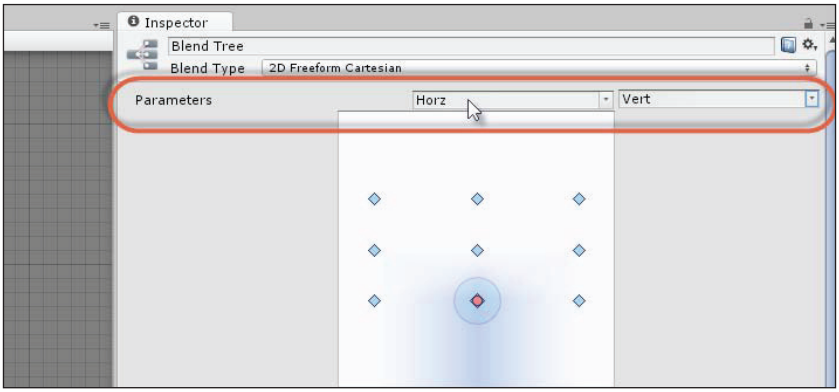

Рис. 6.15. Отображение десятичных параметров на смешивание деревьев

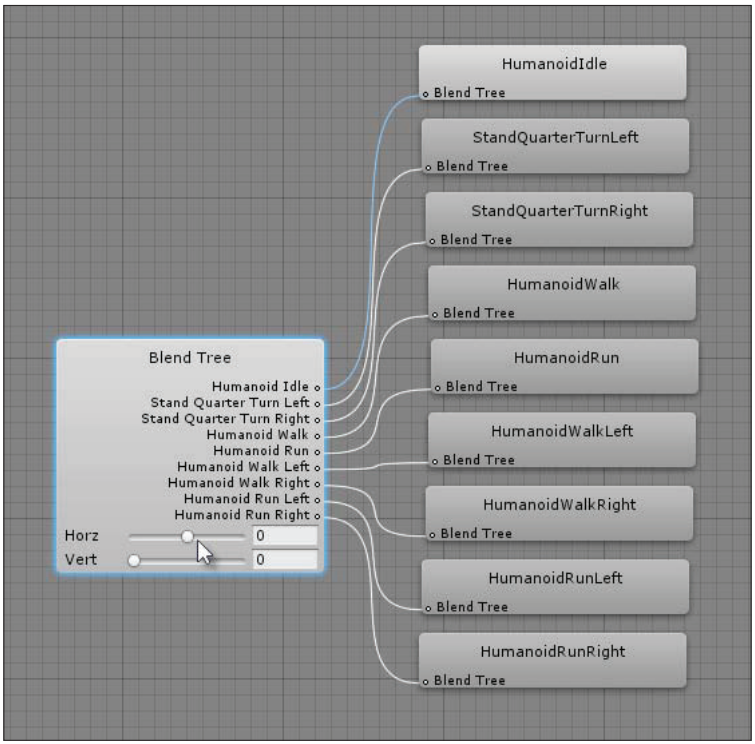

Рис. 6.16. Тестирование смешивания деревьев

Подготовка к написанию скрипта для смешивания деревьев анимаций

На протяжении предыдущих разделов контроллер аниматора системы Mecanim и связанное смешивание деревьев было полностью настроено для воспроизведения анимаций персонажа. Однако состояния (ходьба, бег, ожидание и прочие) не должны включаться в произвольные моменты времени. Напротив, они должны управляться командами игрока, посылаемыми клавишами **WASD**. Чтобы добиться этого, мы должны будем использовать скрипт, связав ввод с клавиатуры с параметрами системы Mecanim. Для начала убедитесь, что вы создали сцену, у вас есть персонаж и вы назначили актив **Animator Controller** в поле **Controller** компонента **Animator** объекта. Это гарантирует, что модель персонажа в сцене использует соответствующий контроллер системы Mecanim.

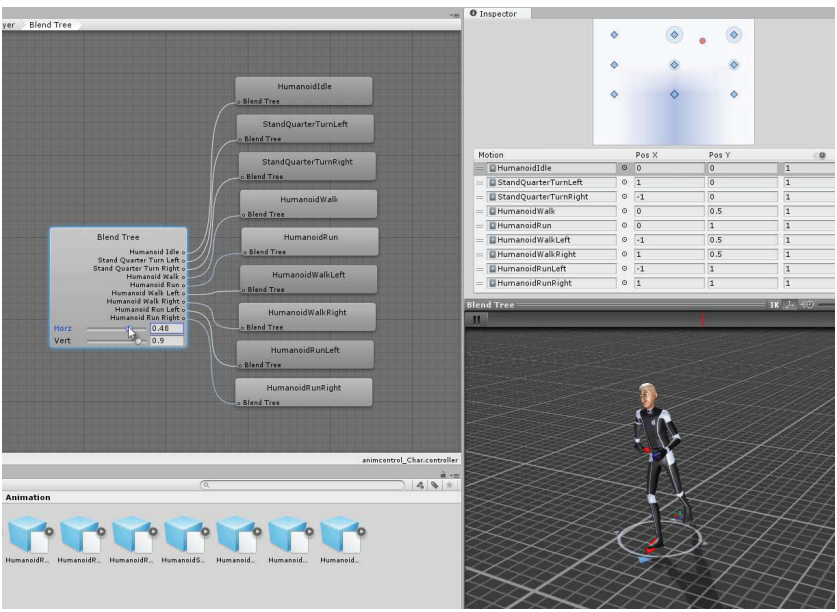

Рис. 6.17. Присоединение актива **Animator Controller** к персонажу

Далее мы должны создать файл скрипта C#, который считывает ввод игрока и отображает его на параметры контроллера, воспроизводя анимацию. Чтобы создать скрипт, щелкните правой кнопкой мыши в панели проекта и выполните **Create → C# Script** в контекстном меню, как это показано на рис. 6.18.

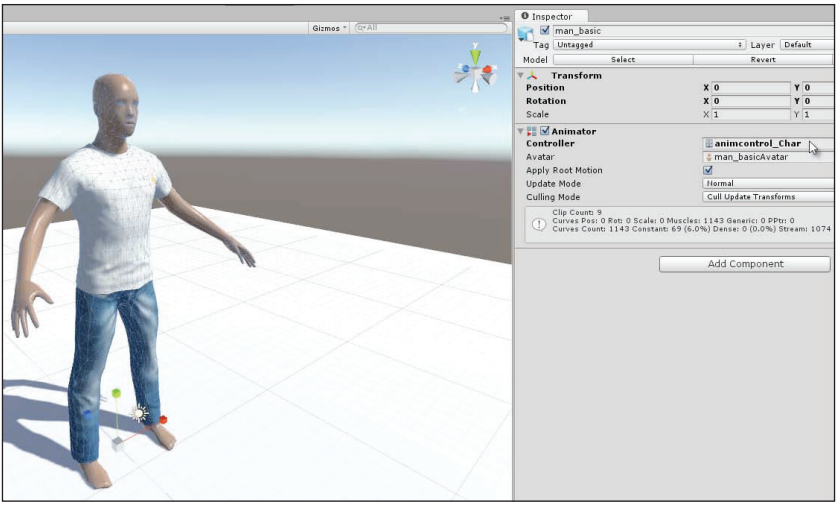

Рис. 6.18. Создание файла скрипта на C#

Назовите файл скрипта CharControl.cs. Затем перетащите файл из панели проекта на объект персонажа в сцене, создав экземпляр скрипта для персонажа.

Написание скрипта для управления смешиванием деревьев системы Mecanim

Теперь мы почти готовы, чтобы оживить персонажа игрока с помощью скрипта. Нам просто нужно только написать вновь созданный скрипт. Рассмотрим следующий пример кода, за ним последует подробное пояснение:

```
using UnityEngine;
using System.Collections;

public class CharControl : MonoBehaviour
{
```

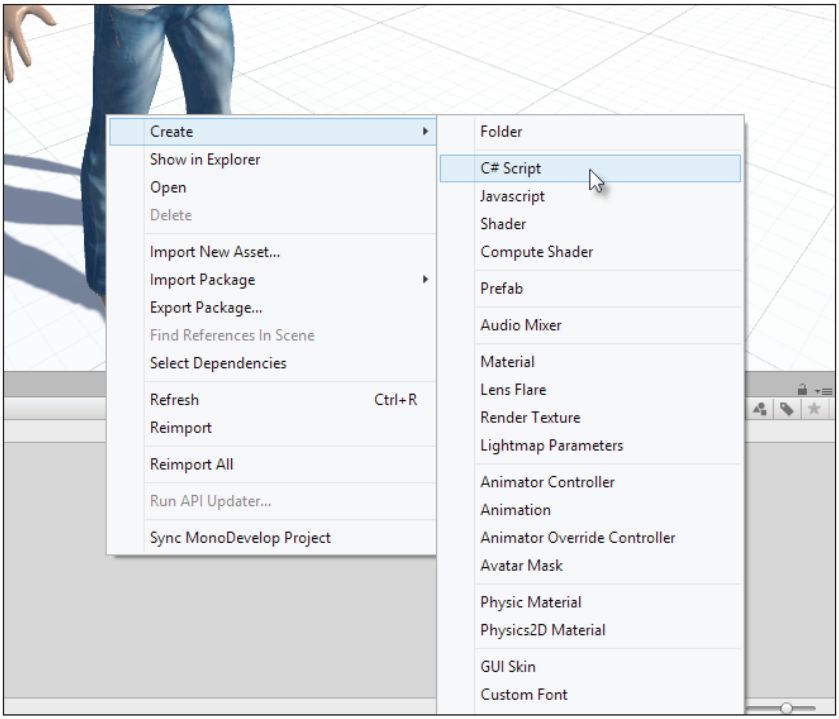

Рис. 6.19. Создание скрипта **CharControl**

```
//Animator Controller
private Animator ThisAnimator = null;

//Float names
private int HorzFloat = Animator.StringToHash("Horz");
private int VertFloat = Animator.StringToHash("Vert");

void Awake()
{
  //Get animator component on this object
  ThisAnimator = GetComponent<Animator>();
}

// Update is called once per frame
void Update ()
{
  //Read player input
  float Vert = Input.GetAxis("Vertical");
  float Horz = Input.GetAxis("Horizontal");
```

```
    //Set animator floating point values
    ThisAnimator.SetFloat(HorzFloat, Horz, 0.2f, Time.deltaTime);
    ThisAnimator.SetFloat(VertFloat, Vert, 0.2f, Time.deltaTime);
  }
}
```

Вот некоторые комментарии к этому коду:

- функция `Animator.StringToHash` используется для преобразования значений строковых имен параметров `Horz` и `Vert` в более эффективные целые переменные, которые будут использованы позже как аргументы для функции `SetFloat`. Путем преобразования строковых имен в целые числа и последующего их использования мы ускорим выполнение кода;
- далее функция `Animator.SetFloat` используется для установки числовых параметров с десятичной точкой графа, управляя анимацией.

Тестирование смешивания деревьев системы Mecanim

Давайте протестируем код, который мы создали. Просто щелкните кнопку **Play** на панели инструментов и нажмите клавиши ввода для управления персонажем (рис. 6.20). Если вы выберете персонажа при запущенной игре и просмотрите окно аниматора, то увидите работу смешивания деревьев при нажатии одной из клавиш **WASD** для перемещения персонажа. Вы увидите, что разные анимации смешиваются плавно и естественно, отображаясь как единая полная и самодостаточная последовательность анимаций. Мои поздравления!

Законченный проект вы можете найти в папке Chapter06/End.

Итоги

К этому моменту у вас уже имеются прочные знания и понимание системы Mecanim как для импорта и настройки готовых персонажей, так и для построения графов и смешивания деревьев для обеспечения общих потребностей анимации, которые отвечают за восприятие ввода игрока и управление. В следующей главе мы завершим нашу экскурсию по анимации в Unity, рассмотрев смешивание форм, кривые анимации и текстуры с видео.

Рис. 6.20. Тестирование анимаций персонажа

Глава 7

Смешивание форм, инверсная кинематика и анимированные текстуры

В этой заключительной главе мы рассмотрим три не связанных между собой метода анимации, доступных в Unity. Это смешивание форм для создания анимаций превращения, таких как синхронное с голосом движение губ и мимика лица, инверсная кинематика (ИК) для позиционирования рук и ног персонажа и анимированные текстуры для воспроизведения предрендеренных видеофайлов, таких как файлы формата MP4, на 3D-поверхностях в виде текстуры. Эти методы анимации не очень часто используются в Unity, но тем не менее могут сыграть важную роль в создании эмоциональных и реалистичных эпизодов. Давайте рассмотрим каждый из них по очереди.

Смешивание форм

Если вы создаете анимацию лица, такую как смена выражения лица или синхронное с голосом движение губ, вам наверняка нужно будет использовать смешивание форм или анимацию превращения. Этот вид анимации, как правило, создается художником-аниматором в программе 3D-моделирования, а затем импортируется в Unity. Ее создание – это двухступенчатый процесс. Во-первых, аниматор определяет все возможные и предельные гримасы меша лица, то есть положение и порядок всех вершин меша лица в экстремумах, и создает ключ формы (Shape Key) или смешивание форм (Blend Shape) для записи состояния меша в этой позе. С помощью записи серии разных

гримас аниматор создает лицевые анимации, в которых генерируются промежуточные положения как взвешенная смесь между разбежностями в разных гримасах.

Рассмотрим следующие три скриншота, в которых меш головы обезьяны в Blender (http://www.blender.org) был установлен в различные экстремальные гримасы. Каждая гримаса записана в виде ключа формы или смешивания форм. В следующем скриншоте мы отобразили начальную, или основную, гримасу. Иногда она называется гримасой покоя.

Законченный проект для этого раздела вы можете найти в прилагаемых к этой книге файлах в папке Chapter07/BlendShapes.

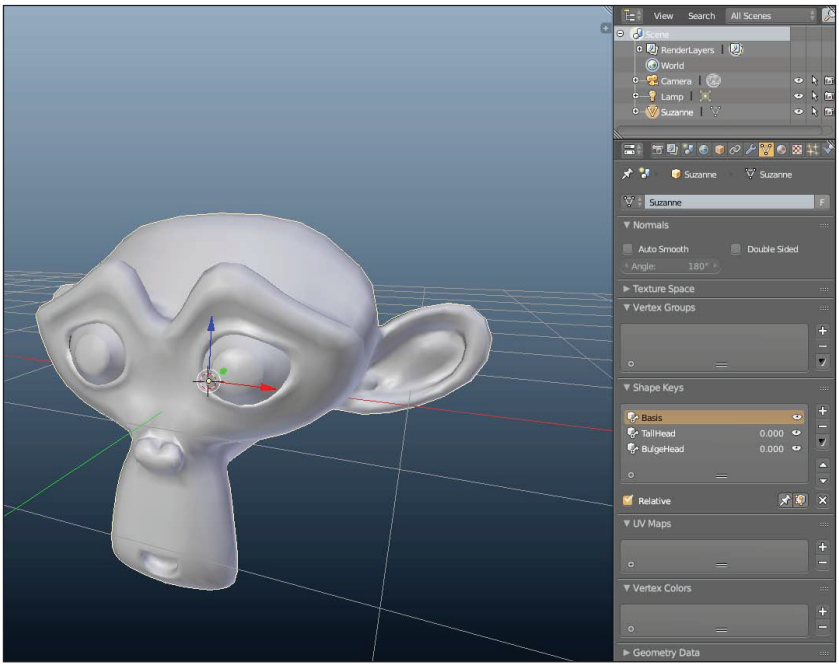

Рис. 7.1. Основная гримаса для головы обезьяны

На следующем скриншоте представлена промежуточная гримаса (рис. 7.2).

На следующем скриншоте представлена финальная, или зацикливающая, гримаса (рис. 7.3).

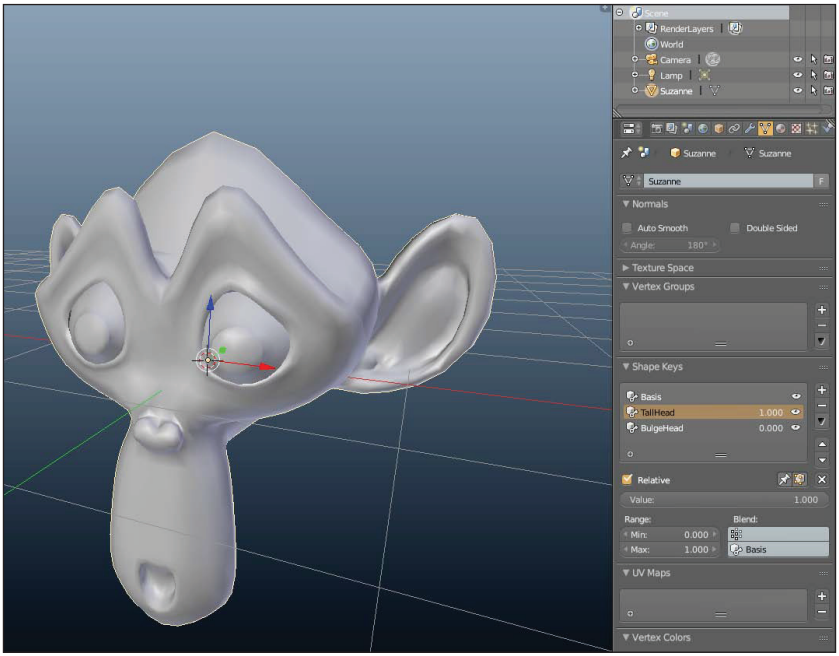

Рис. 7.2. Экстремальная гримаса 1 для головы обезьяны

После создания набора механизмов вершин и гримас вы можете экспортировать меш из программы 3D-моделирования в файл формата FBX, как и обычные меши. Шаги экспорта отличаются для разных пакетов. При импорте меша в Unity не забудьте выбрать меш в панели проекта и включить опцию **Import BlendShapes** в инспекторе объектов, как это показано на рис. 7.4.

Вы можете легко проверить, правильно ли были импортированы смешивания форм вашего меша, перетащив меш в сцену. Затем раскройте поле **BlendShapes** в инспекторе объектов (рис. 7.5). В этом поле будут перечислены все импортированные смешивания форм.

Значения в полях **BlendShapes** для ваших моделей будут колебаться в диапазоне 0–100. При этом поле будет в полной мере поддерживать ваш меш, когда его значение равно 100 (рис. 7.6).

Если вам нужно получить доступ на чтение или изменить веса и значения **BlendShape** из скрипта, вы можете использовать переменную `Mesh.BlendShapeCount` и функцию `SkinnedMeshRenderer.SetBlendShapeWeight` (http://docs.unity3d.com/ScriptReference/Skinned-

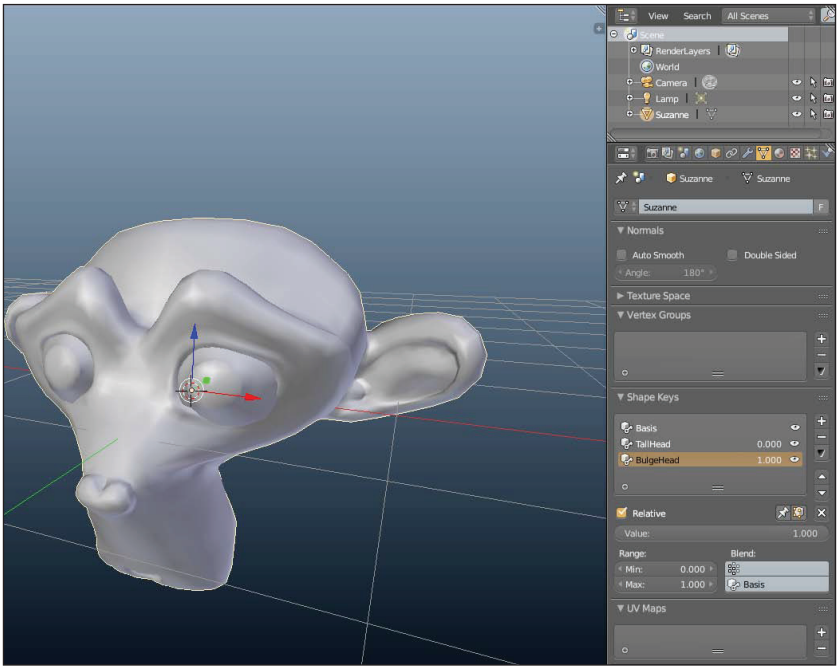

Рис. 7.3. Финальная гримаса для головы обезьяны

MeshRenderer.SetBlendShapeWeight.html). Однако вы, скорее всего, хотите воспроизвести анимацию **BlendShapes**. Чтобы сделать это, вы можете использовать окно **Animation** Unity, выполнив **Window → Animation** из меню приложения. Подробно это окно было описано в предыдущих главах. Как и для других мешей, вы можете создать ключевые кадры для смешивания форм. Просто переместитесь с помощью мыши на нужный момент на временной шкале в окне анимации, а затем установите значения **BlendShape** в инспекторе объектов на выбранный момент. При этом Unity автоматически сгенерирует ключевые кадры для смешивания форм (рис. 7.7).

Инверсная кинематика

Инверсная кинематика, часто сокращается как **ИК**, очень важна для достижения непринужденной и реалистичной анимации персонажа. Практически инверсная кинематика позволяет спозиционировать

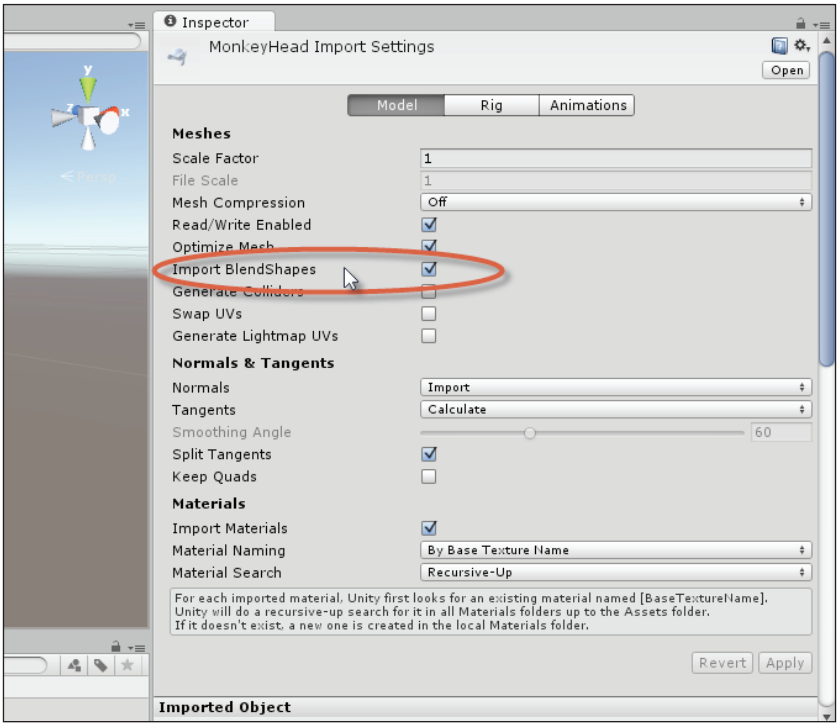

Рис. 7.4. Импорт анимации смешивания форм с мешами

кость руки или ноги в оснащении, а остальные суставы и кости будут размещены автоматически для достижения реалистичного вида. Другими словами, при создании анимации ходьбы или жестов, где ноги или руки вступают в контакт с поверхностью, инверсная кинематика сильно облегчает работу аниматоров, вместо анимации всех костей им достаточно определить положение стопы и бедра или руки и плеча (рис. 7.8).

Большинство приложений 3D-моделирования, таких как Maya, 3ds Max и Blender, предоставляют возможности и функции для создания инверсной кинематики. Цель этих особенностей – в том, чтобы помочь художникам создавать анимации легче и быстрее. Однако разные программы сохраняют данные инверсной кинематики по-разному, и эти данные обычно не могут быть импортированы в Unity даже через файлы формата FBX. Это значит, что ваши модели персонажей почти всегда будут импортированы без данных инверсной ки-

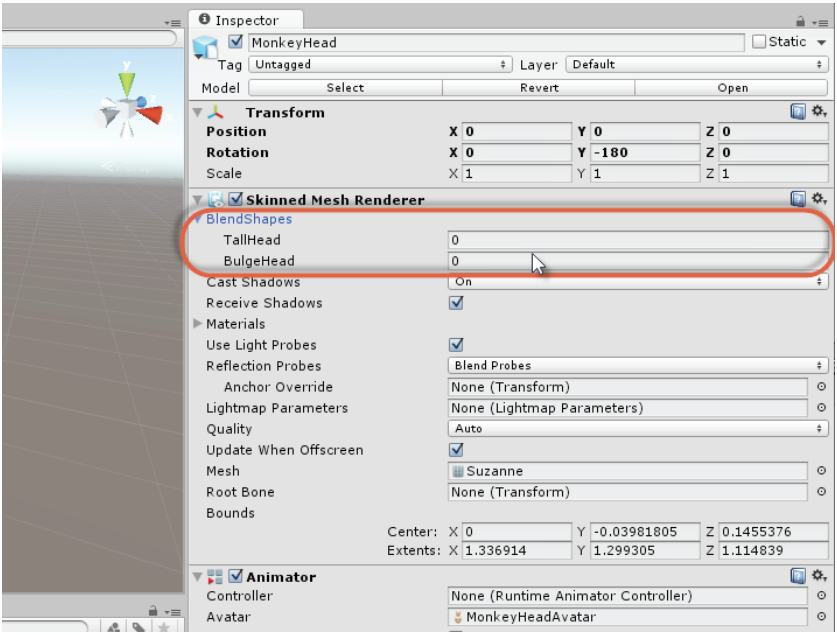

Рис. 7.5. Доступ к смешиванию форм для выбранного меша

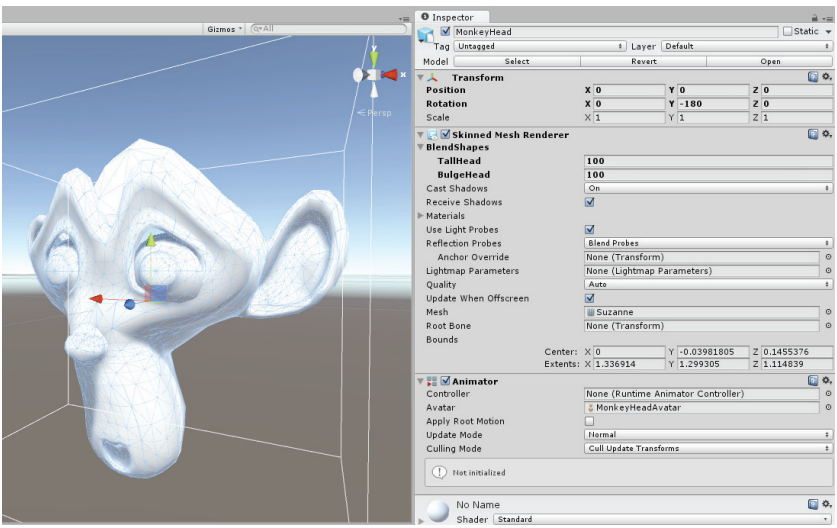

Рис. 7.6. Тестирование смешивания форм для выбранного меша

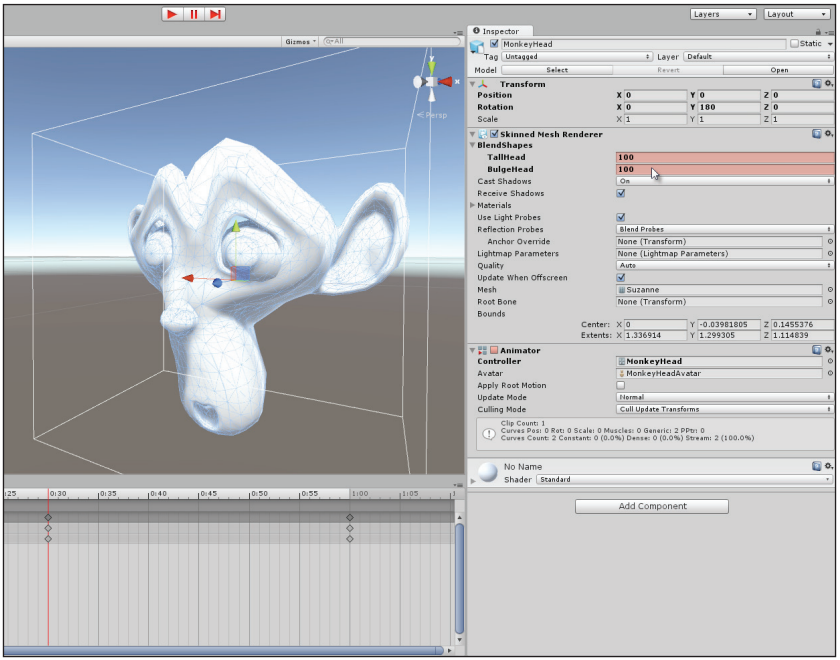

Рис. 7.7. Анимация смешивания форм

нематики. Но это не означает анимации инверсной кинематики, созданное в программах 3D-моделирования будет выглядеть по-другому после импорта в Unity. Это лишь означает, что все анимации инверсной кинематики будут импортированы в Unity как анимации **прямой кинематики (ПК)**. Они будут выглядеть так же, но не дадут вам доступа к функциям инверсной кинематики, не позволяя вам легко позиционировать и перемещать скелет динамически с помощью инверсной кинематики. Чтобы исправить это, вы должны будете вручную настроить инверсную кинематику для костей в Unity, если захотите получить доступ к инверсной кинематике для динамичного перемещения рук и ног ваших персонажей. Чтобы получить правильно работающую инверсную кинематику, мы сначала импортируем оснащенную модель персонажа в Unity. Затем мы выбираем модель в панели проекта и включаем оснащение **Humanoid**, перейдя на вкладку **Rig** в инспекторе объектов и выбрав тип анимации **Humanoid**. Это не подключит инверсную кинематику как таковую, но настроит модель для использования системы Аватар, которая необходима для инверс-

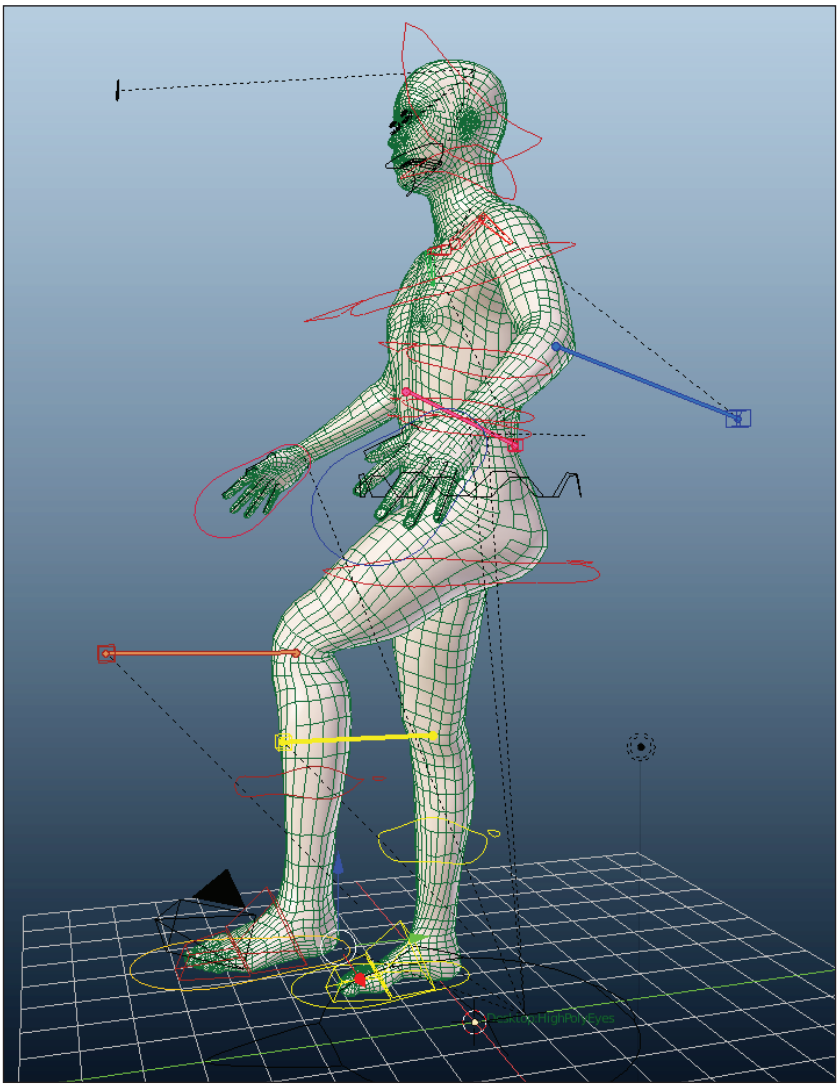

Рис. 7.8. Настройка инверсной кинематики на Blender

ной кинематики. Подходящая модель персонажа входит в сопроводительные файлы и находится в папке Chapter07/IK (рис. 7.9).

Давайте применим инверсную кинематику к рукам персонажа, она обеспечит, что при размещении рук все другие кости руки автомати-

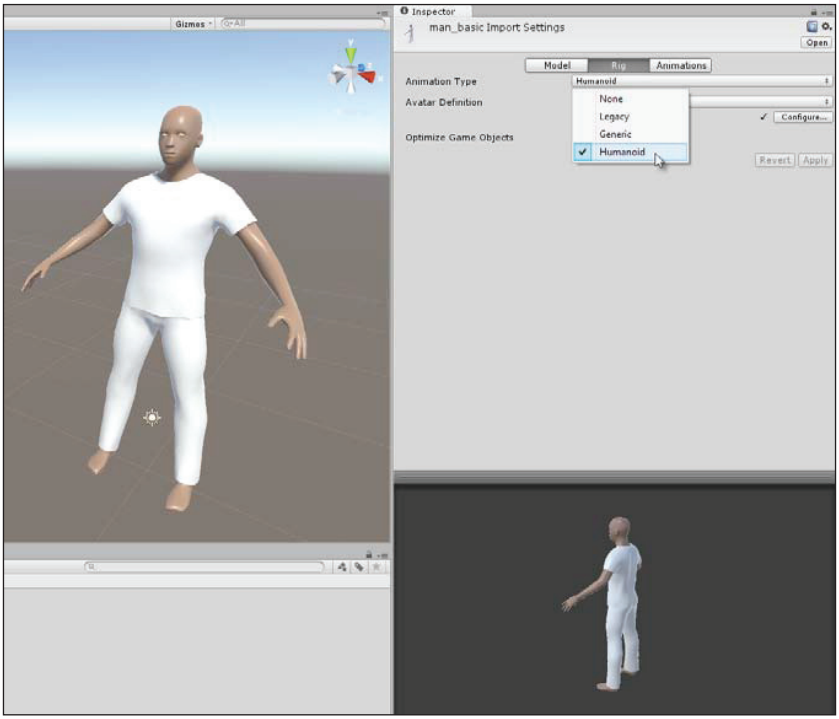

Рис. 7.9. Настройка персонажа как оснастки **Humanoid**

чески разместятся так, чтобы соответствовать положению рук. Чтобы сделать это, создайте два новых пустых игровых объекта в сцене. Они будут представлять собой кости игрового персонажа. Эти кости будут использоваться для позиционирования руки персонажа. Назовите их `IK_LeftHand` и `IK_RightHand` (рис. 7.10).

По умолчанию кости не видны в окне, если они не выбраны в панели иерархии. Для того чтобы кости стали видимыми и интерактивными объектами, выберите иконку куба в инспекторе объектов и назначьте представление **2D Gizmo** к объекту. После такого выбора кости станут видимыми и могут быть выбраны в окне просмотра (рис. 7.11).

Расположите кости в сцене перед руками персонажа, левую кость – перед левой рукой, а правую кость – перед правой рукой (рис. 7.12).

Создайте новый актив **Animator Controller** (`animCharControl`) и свяжите узел **Entry** с пустым состоянием. Для создания пустого состояния щелкните правой кнопкой мыши внутри графа и выполните

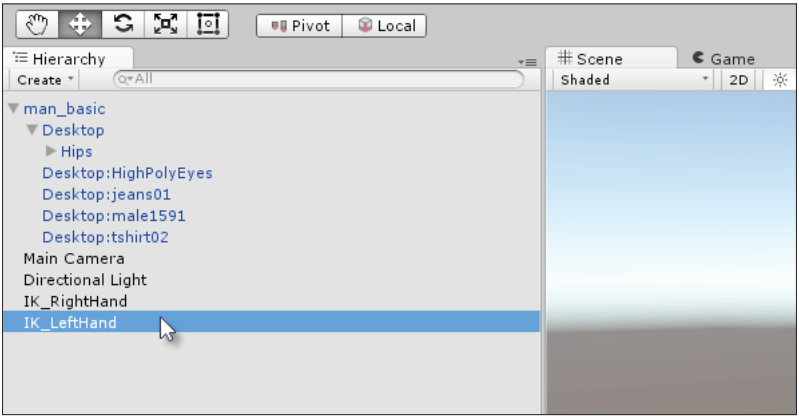

Рис. 7.10. Создание инверсной кинематики костей для пустых объектов

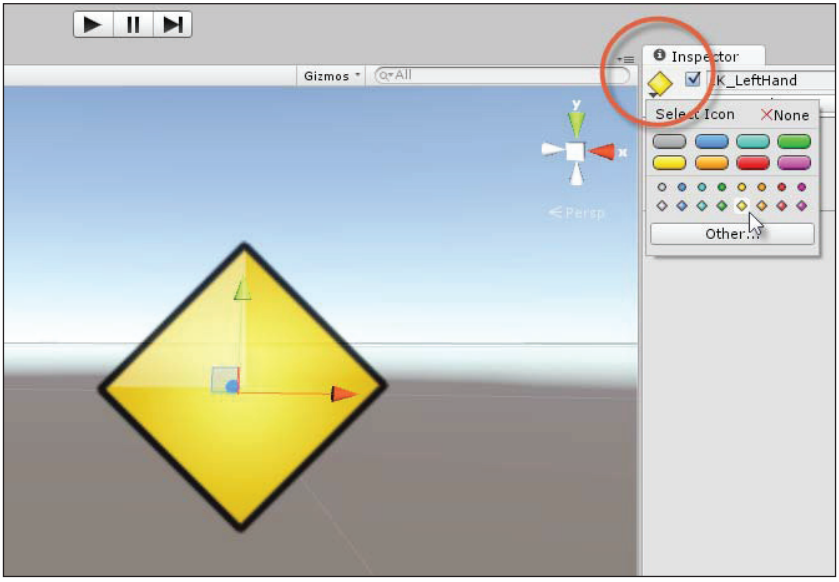

Рис. 7.11. Включение видимости для пустых объектов

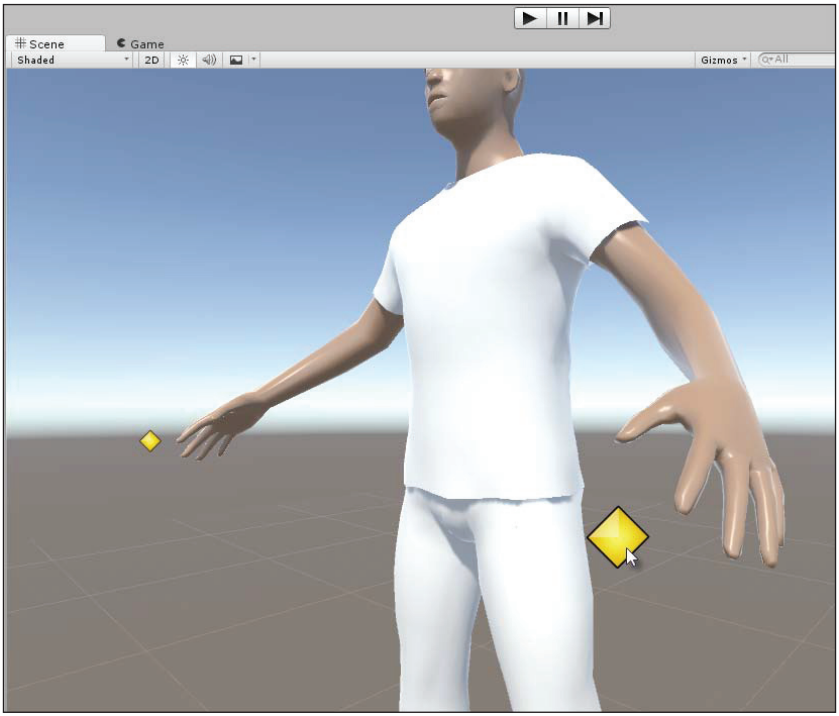

Рис. 7.12. Расположение костей инверсной кинематики

Create State → Empty из контекстного меню. Убедитесь, что пустое состояние является узлом по умолчанию. Создание актива **Animator Controller** важно как для анимации персонажа, так и для работы с инверсной кинематикой (рис. 7.13).

Убедитесь, что для актива **Animator Controller** установлен флажок **IK Pass**, позволяющий управлять инверсной кинематикой персонажа программно из скрипта. Это важно при использовании инверсной кинематики в Unity. Чтобы сделать это, нажмите на иконку в виде шестеренки для **Base Layer** в окне аниматора. Затем установите флажок **IK Pass** (рис. 7.14).

Назначьте персонажу контроллер, перетащив актив **Animator Controller** из панели проекта на меш персонажа в сцене (рис. 7.15).

Далее создайте файл скрипта и присоедините его к персонажу. Этот файл скрипта позволит нам использовать объекты управления левой и правой рук для нужд инверсной кинематики рук. Так как флажок

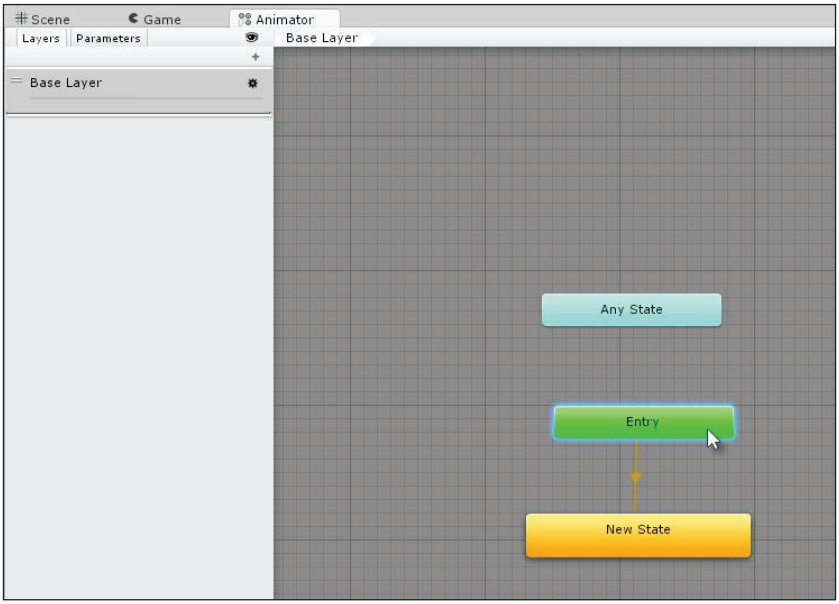

Рис. 7.13. Создание нового актива **Animator Controller**

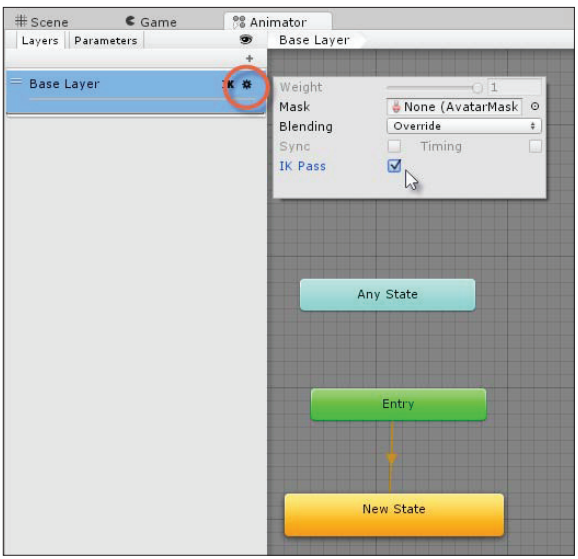

Рис. 7.14. Установка флажка **IK Pass**

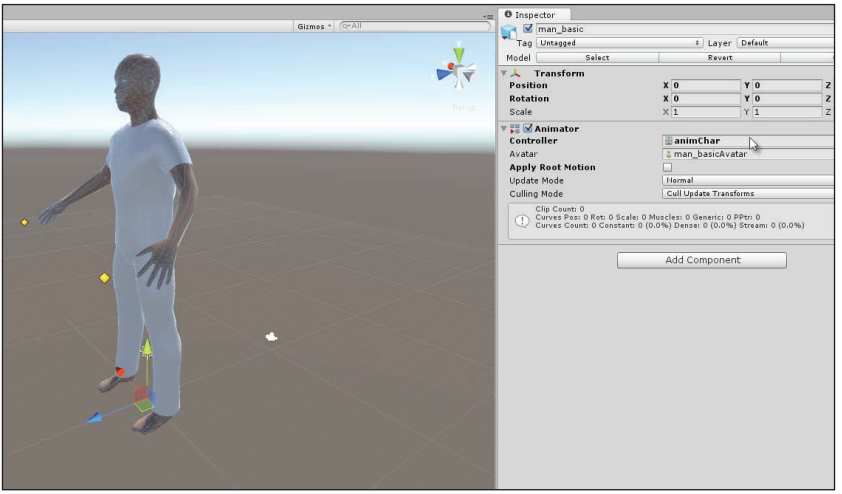

Рис. 7.15. Присоединение актива **Animator Controller** к персонажу

IK Pass установлен, актив **Animator Controller** автоматически вызовет функцию событий OnAnimatorIK для позиционирования рук скелета на указанных объектах:

```
using UnityEngine;
using System.Collections;

public class ArmIK : MonoBehaviour
{
  public float leftHandPositionWeight;
  public float leftHandRotationWeight;

  public float rightHandPositionWeight;
  public float rightHandRotationWeight;

  public Transform leftHandObj;
  public Transform rightHandObj;
  private Animator animator;

  void Start() {
    animator = GetComponent<Animator>();
  }
  void OnAnimatorIK(int layerIndex) {
    animator.SetIKPositionWeight(AvatarIKGoal.LeftHand, leftHandPosi-
tionWeight);
    animator.SetIKRotationWeight(AvatarIKGoal.LeftHand, leftHandRota-
tionWeight);
```

```
    animator.SetIKPosition(AvatarIKGoal.LeftHand, leftHandObj.position);
    animator.SetIKRotation(AvatarIKGoal.LeftHand, leftHandObj.rotation);
    animator.SetIKPositionWeight(AvatarIKGoal.RightHand, rightHandPosi-
tionWeight);
    animator.SetIKRotationWeight(AvatarIKGoal.RightHand, rightHandRota-
tionWeight);
    animator.SetIKPosition(AvatarIKGoal.RightHand, rightHandObj.posi-
tion);
    animator.SetIKRotation(AvatarIKGoal.RightHand, rightHandObj.rota-
tion);
  }
}
```

Прикрепите файл скрипта к персонажу. Затем перетащите левый и правый объекты в соответствующие слоты `Transform` в инспекторе объектов. Далее установите значения весовых коэффициентов равными 100, чтобы обеспечить влияние костей на объект.

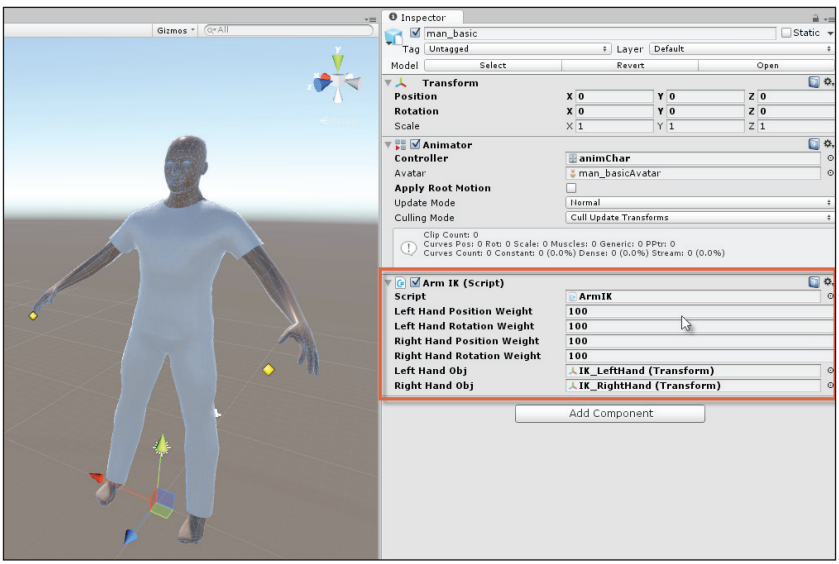

Рис. 7.16. Управление весовыми коэффициентами костей инверсной кинематики

Теперь можно протестировать игру и посмотреть на работу инверсной кинематики. При запуске игры персонаж должен находиться в стандартной позе, но его руки будут притягиваться к объектам управления рук.

Рис. 7.17. Тестирование
инверсной кинематики костей

Для тестирования передвиньте объекты управления в окне сцены, вы увидите, что руки персонажа двигаются, изгибаются и поворачиваются, чтобы соответствовать новому положению (рис. 7.18). Поздравления! Теперь вы можете настроить персонаж в системе Mecanim для анимации с использованием инверсной кинематики в реальном времени.

Анимированные текстуры

Текстуры – это изображения, которые отображаются на геометрических 3D-телах. Большинство текстур являются двумерными ста-

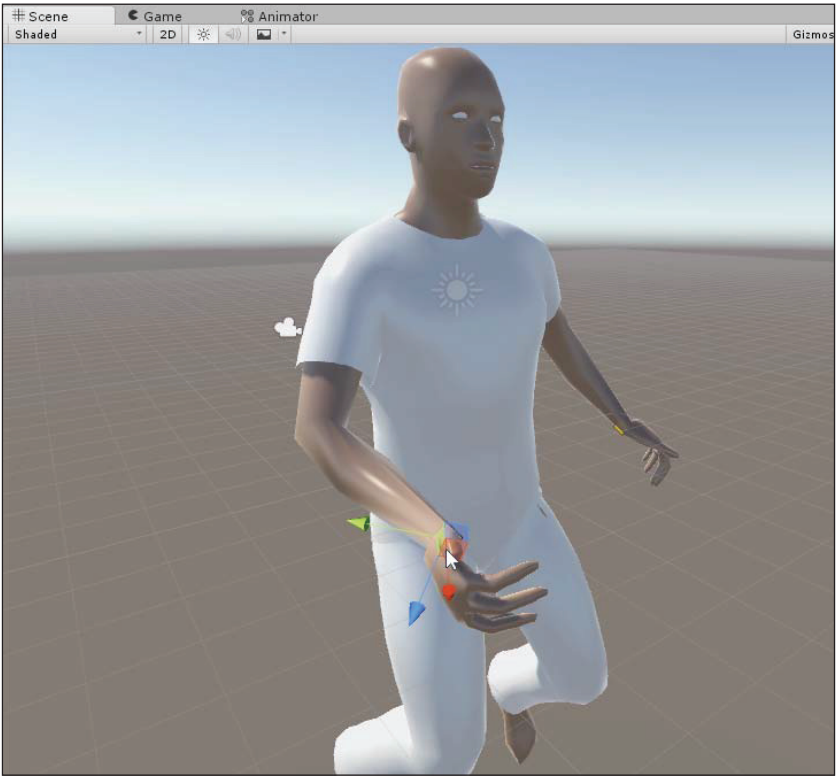

Рис. 7.18. Позиционирование рук

тическими изображениями, такими как файлы JPEG и PNG. Тем не менее Unity также поддерживает и анимированные текстуры в виде видео- или кинотекстур. Это позволит вам импортировать в проект видеофайлы и использовать их в качестве текстур. Фильмы могут воспроизводиться со звуком и отображаться в игре. Таким образом, они особенно полезны для создания заставок, списков принявших участие в разработке, воспоминаний, рекламы и других специальных эффектов. Однако анимированные текстуры – ресурс тяжеловесный, и их применение сильно снижает производительность игры. По этой причине вы должны использовать их разумно. Unity поддерживает различные форматы видео, но рекомендованным форматом является OGV (Ogg Theora video). Более подробную информацию о данном формате можно найти в Интернете по адресу http://www.theora.org/.

Если ваши фильмы имеют другой формат, а не OGV, то вы можете конвертировать их с помощью бесплатного медиаплеера VLC, который можно скачать, загрузив страницу http://www.videolan.org/vlc/index.en_GB.html. Чтобы импортировать анимированную текстуру, перетащите файл формата OGV в панель проекта Unity. После того как он импортируется, вы можете просмотреть видео, нажав кнопку **Play** на панели предварительного просмотра в инспекторе объектов.

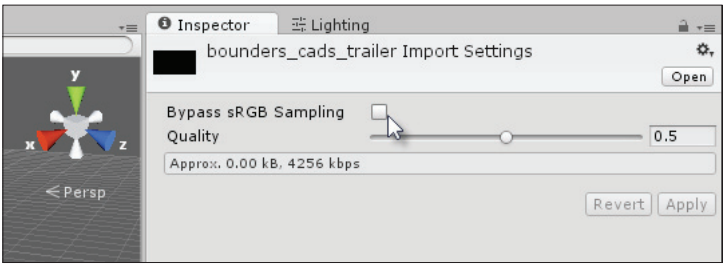

Рис. 7.19. Импорт анимированной текстуры из OGV-файла

Затем создайте объект плоской поверхности в сцене в качестве поверхности, на которой видео будет воспроизводиться во время выполнения. Чтобы создать плоскость, выполните **GameObject** →

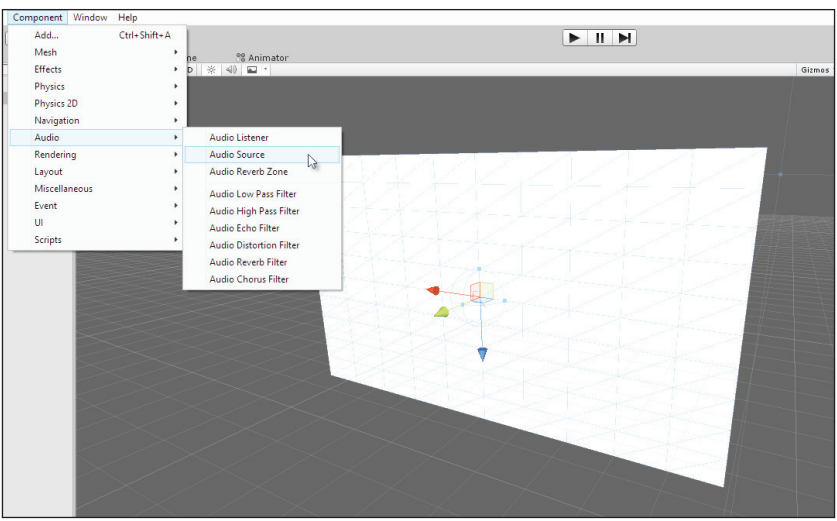

Рис. 7.20. Создание объекта плоскости
для воспроизведения видеофайла

3D Object → Plane из меню приложения. После того как она будет создана, добавьте к объекту компонент **Audio Source**, который будет отвечать за воспроизведение звука. Чтобы сделать это, выберите объект **Plane** и выполните **Component → Audio → Audio Source** из меню приложения.

Затем создайте новый материал для нанесения на меш для отображения текстуры фильма. Чтобы создать новый материал, щелкните правой кнопкой мыши в панели проекта и выполните **Create → Material** из контекстного меню. Для типа шейдеров выберите **Unlit → Texture**. Это привет к тому, что на текстуру не будет влиять освещение сцены.

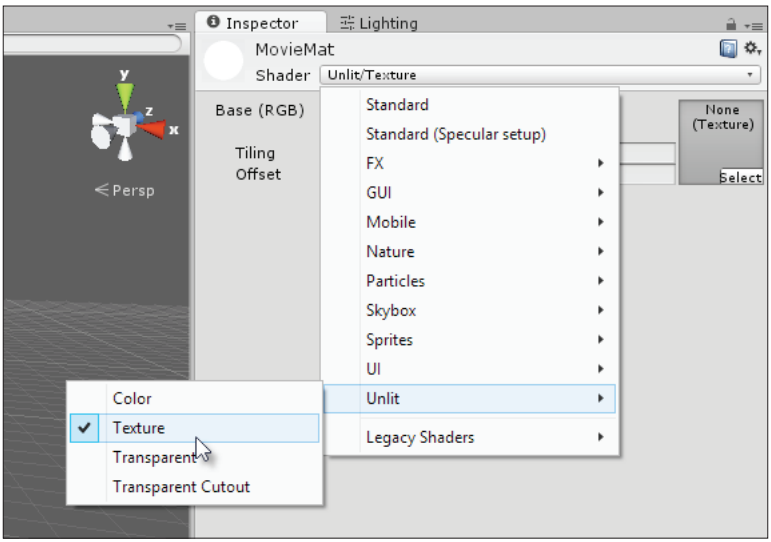

Рис. 7.21. Создание материала текстуры для воспроизведения видео

Присоедините материал к плоскости, перетащив его из панели проекта на объект **Plane** в сцене (рис. 7.22).

Далее создайте новый файл скрипта C# для воспроизведения видеотекстуры и связанного с ней звука. В него нужно будет поместить следующий пример кода:

```
using UnityEngine;
using System.Collections;

public class MoviePlay : MonoBehaviour
{
```

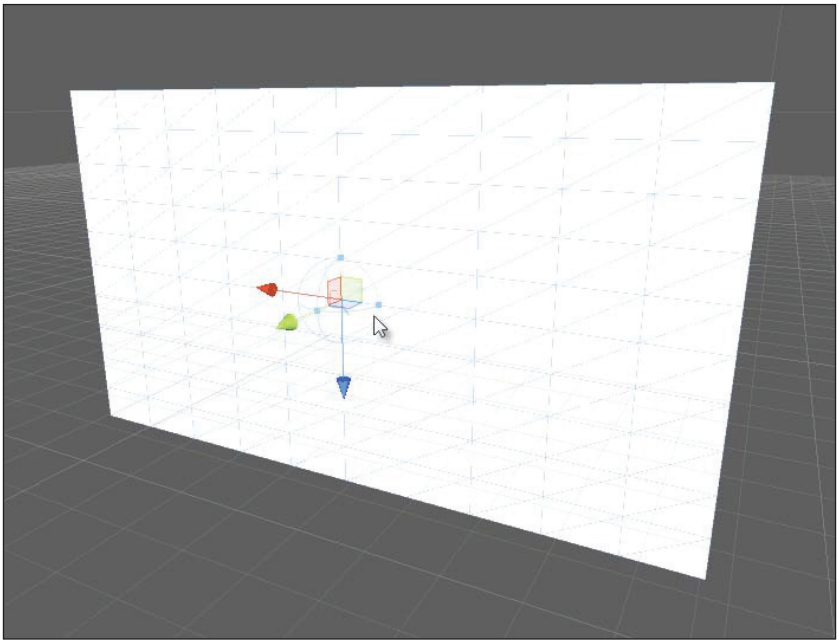

Рис. 7.22. Присоединение материала к плоскости

```
  //Reference to movie to play
  public MovieTexture Movie = null;

  // Use this for initialization
  void Start ()
  {
    //Get Mesh Renderer Component
    MeshRenderer MeshR = GetComponent<MeshRenderer>();

    //Assign movie texture
    MeshR.material.mainTexture = Movie;
    GetComponent<AudioSource>().clip = Movie.audioClip;
    GetComponent<AudioSource>().spatialBlend=0;

    Movie.Play();
    GetComponent<AudioSource>().Play();
  }
}
```

Перетащите файл скрипта на объект плоскости в сцене, а затем перетащите видеотекстуру из панели проекта на слот **Movie** компонента **MoviePlay**. Это определит текстуру, отображаемую материалом.

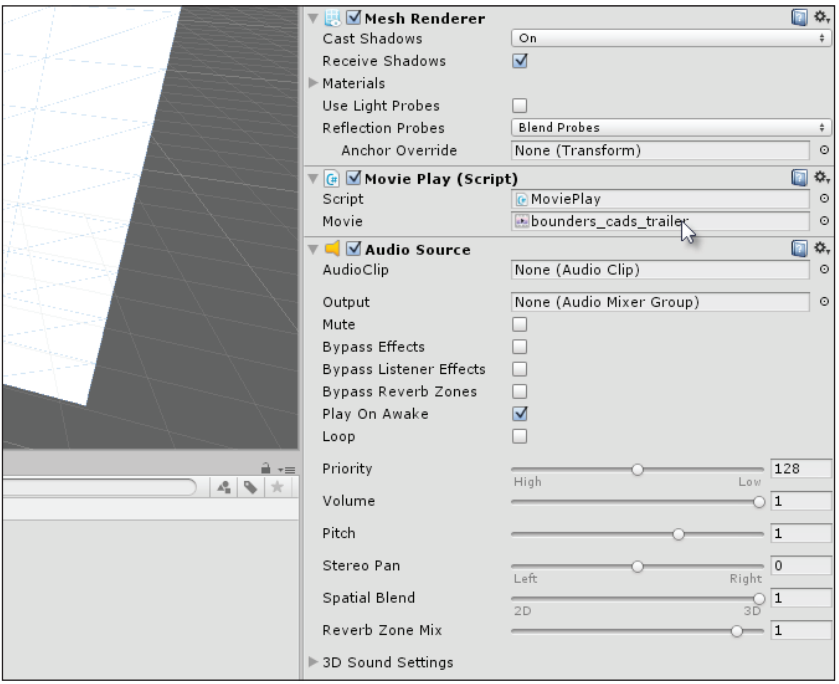

Рис. 7.23. Настройка компонента MoviePlay

Теперь нажмите кнопку **Play** панели инструментов, чтобы запустить приложение. Когда вы сделаете это, видеотекстура, назначенная плоскости, запустится автоматически вместе со звуком и музыкой (рис. 7.24). Мои поздравления! Теперь у вас есть возможность воспроизводить заставки, анимацию и другие предварительно подготовленные фрагменты видео в вашей игре.

Законченный проект для этого раздела вы можете найти в прилагаемых к этой книге файлах в папке Chapter07/MovieTextures.

Итоги

Отличная работа! Вы завершили чтение этой главы и этой книги. Сейчас вы уверенно можете использовать следующие возможности: смешивание форм для анимации превращений и переходы через состояния вершин, инверсную кинематику для создания анимированных рук и ног, легко меняющих свои положения, и анимированных

Рис. 7.24. Воспроизведение в игре анимированных текстур

текстур для создания заставок и воспроизведения других видеофайлов. Кроме того, прочитав эту книгу, вы изучили практически весь обширный набор функций анимации Unity, в том числе скрипты, кривые анимации, окно анимации, системы частиц, оснащенные скелеты, ключевые кадры, интерактивные элементы и многое другое.

Предметный указатель

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Алан Торн

Основы анимации в Unity

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Рагимов Р. Н.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 60×90 1/16 .
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 23. Тираж 200 экз.

Веб-сайт издательства: www.дмк.рф