Содержание

Об авторе **10**

О технических рецензентах **11**

Предисловие **13**

Глава 1. Основы C# Unity **19**

Почему C#? 20
Создание файлов скриптов 21
Присоединение скриптов 24
Переменные 25
Условные операторы 28
    Оператор if 29
    Оператор switch 31
Массивы 34
Циклы 38
    Цикл foreach 38
    Цикл for 39
    Цикл while 41
    Бесконечные циклы 42
Функции 43
События 46
Классы и объектно-ориентированное программирование 47
Классы и наследование 50
Классы и полиморфизм 52
Свойства в C# 56
Комментирование 59
Визуализация переменных 61
Оператор ? 64
Методы SendMessage и BroadcastMessage 64
Итоги 67

Глава 2. Отладка **68**

Ошибки компиляции и консоль 69
Отладка с помощью Debug.log – определяемые программистом сообщения 73
Переопределение метода ToString 75

Визуальная отладка 80
Регистрация ошибок 83
Отладка с помощью редактора 87
Использование профилирования 89
Отладка с помощью MonoDevelop – начало 94
Отладка с помощью MonoDevelop – окно Watch 100
Отладка с помощью MonoDevelop – продолжить и по шагам 104
Отладка с помощью MonoDevelop – стек вызовов 106
Отладка с помощью MonoDevelop – окно Immediate 108
Отладка с помощью MonoDevelop – точки останова с условием 109
Отладка с помощью MonoDevelop – точки трассировки 111
Итоги 114

Глава 3. Синглтоны, статические члены, объекты игры и игровые миры 115

Объекты игры 115
Интерактивность компонента 117
Функция GetComponent 119
Получение нескольких компонентов 121
Компоненты и сообщения 122
Объекты игры и игровой мир 123
Поиск объектов игры 123
Сравнение объектов 126
Получение ближайшего объекта 126
Поиск любого объекта определенного типа 127
Отсутствие препятствий между объектами игры 128
Доступ к иерархии объектов 130
Игровой мир, время и обновление 131
Правило № 1 – важность событий обновления кадров 133
Правило № 2 – движение должно основываться на времени 134
Неуничтожаемые объекты 135
Синглтоны и статические переменные 137
Итоги 141

Глава 4. Событийное программирование 142

События 143
Управление событиями 147
Основы управления событиями с помощью интерфейсов 148
Создание класса EventManager 151

Директивы #region и #endregion для свертывания кода в MonoDevelop ... 156
Использование EventManager ... 157
Альтернативный способ, основанный на делегировании ... 158
События класса MonoBehaviour ... 162
События мыши и сенсорного ввода ... 163
Фокус приложения и остановка на паузу ... 166
Итоги ... 168

Глава 5. Камеры, рендеринг и сцены ... 171

Гизмо камеры ... 171
Быть на виду ... 174
Определение видимости объекта ... 176
Подробнее о видимости ... 177
Проверка области отсечения – рендеры ... 178
Проверка области отсечения – точки ... 179
Проверка области отсечения – окклюзия ... 180
Видимость камерой – впереди или позади ... 181
Ортографические камеры ... 182
Рендеринг камеры и постобработка ... 186
Дрожание камеры ... 193
Камеры и анимация ... 195
Сопровождающие камеры ... 195
Камеры и кривые ... 197
Траектория камеры – iTween ... 200
Итоги ... 203

Глава 6. Работа с фреймворком Mono ... 205

Списки и коллекции ... 206
Класс List ... 207
Класс Dictionary ... 210
Класс Stack ... 211
Интерфейсы IEnumerable и IEnumerator ... 213
Перебор врагов с помощью интерфейса IEnumerator ... 214
Строки и регулярные выражения ... 219
Null, пустые строки и пробелы ... 219
Сравнение строк ... 220
Форматирование строк ... 222
Цикл по строке ... 222

Создание строк ... 223
Поиск в строках ... 223
Регулярные выражения ... 223
Неограниченное количество аргументов ... 225
Язык интегрированных запросов ... 226
Linq и регулярные выражения ... 229
Работа с активами текстовых данных ... 230
Текстовые активы – статическая загрузка ... 230
Текстовые активы – загрузка из локальных файлов ... 232
Текстовые активы – загрузка из INI-файлов ... 233
Текстовые активы – загрузка из CSV-файлов ... 235
Текстовые активы – загрузка из Интернета ... 235
Итоги ... 236

Глава 7. Искусственный интеллект ... 237

Искусственный интеллект в играх ... 238
Начало проекта ... 239
Впекание навигационного меша ... 241
Создание агента NPC ... 246
Конечные автоматы в Mecanim ... 248
Конечный автомат состояний в C# – начало ... 255
Создание состояния Idle ... 256
Создание состояния Patrol ... 260
Создание состояния Chase ... 264
Создание состояния Attack ... 266
Создание состояния Seek-Health (или состояния побега) ... 267
Итоги ... 270

Глава 8. Настройка редактора Unity ... 272

Пакетное переименование ... 272
Атрибуты C# и отражение ... 278
Смешивание цветов ... 282
Отображение свойств ... 287
Локализация ... 293
Итоги ... 301

Глава 9. Работа с текстурами, моделями и 2D ... 303

Скайбокс ... 303
Создание меша из скрипта ... 310

Анимация UV-координат – скроллинг текстур 316
Рисование на текстуре 319
Шаг 1 – создание шейдера смешивания текстур..... 320
Шаг 2 – создание скрипта рисования текстуры 323
Шаг 3 – настройка текстуры рисования 330
Итоги 334

Глава 10. Управление исходными текстами и другие подсказки 336

Git – управление исходными текстами..... 336
Шаг № 1 – загрузка 338
Шаг № 2 – встраивание в проект Unity..... 339
Шаг № 3 – настройка Unity для управления исходным кодом 341
Шаг № 4 – создание хранилища 342
Шаг № 5 – игнорирование файлов 343
Шаг № 6 – создание первой фиксации 344
Шаг № 7 – изменение файлов 346
Шаг № 8 – получение файлов из хранилища 348
Шаг № 9 – просмотр хранилища 350
Папка ресурсов и внешние файлы 352
Пакеты активов и внешние файлы 354
Постоянные данные и сохранение игры 357
Итоги 361

Предметный указатель 363

Об авторе

Алан Торн (Alan Thorn), разработчик игр, независимый программист и писатель, с более чем 13-летним опытом работы, живущий в Лондоне. В 2010 году основал компанию Wax Lyrical Games и является создателем удостоенной многочисленными наградами игры *Baron Wittard: Nemesis of Ragnarok*. Автор 10 видеокурсов и 11 книг по разработке игр, в том числе *Unity 4 Fundamentals: Get Started at Making Games with Unity*, *Focal Press*, *UDK Game Development* и *Pro Unity Game Development with C#*, *Apress*. Кроме того, он приглашенный лектор курса *Game Design & Development Masters Program* в Национальной школе кино и телевидения.

Алан участвовал как независимый разработчик в более чем 500 проектах по созданию игр, симуляторов, игровых киосков, «серьезных» игр, программ расширения реальности для игровых студий, музеев и тематических парков по всему миру. В настоящее время работает над подающей большие надежды приключенческой игрой *Mega Bad Code* для настольных компьютеров и мобильных устройств. Алан обожает графику. Увлекается философией, йогой и пешими прогулками по сельской местности. Адрес его электронной почты directx_user_interfaces@hotmail.com.

О технических рецензентах

Дилан Агис (Dylan Agis), программист и дизайнер игр, в настоящее время участвует, как независимый разработчик, в нескольких проектах и в то же время развивает несколько собственных. Он имеет большой опыт работы в C++ и C#, а также в Unity и любит решать проблемы.

> Я хотел бы поблагодарить издательство Packt Publishing за предоставленную мне возможность ознакомиться с книгой и автора за интересное чтение.

Джон П. Доран (John P. Doran), дизайнер игр, он занимается созданием игр более чем 10 лет. Участвовал в разработке множества игр в командах, численностью от него одного до 70, в учебных, ультрасовременных и профессиональных проектах.

Ранее он работал в компании LucasArts над созданием игры *Star Wars: 1313* в качестве дизайнера-стажера, единственный дизайнер junior в команде дизайнеров senior. Он был также ведущим инструктором DigiPen®-Ubisoft® Campus Game Programming Program, обучая студентов-выпускников продвинутому программированию игр по интенсивной программе.

Джон – в настоящее время технический дизайнер в департаменте DigiPen's Research & Development. Кроме того, он также и преподаватель и обучает студентов по нескольким предметам, читая лекции по разработке игр, в том числе по C++, Unreal, Flash, Unity, и многому другому.

Он был техническим рецензентом девяти книг по разработке игр и является автором книг *Unity Game Development Blueprints*, *Getting Started with UDK*, *UDK Game Development [Video]*, and *Mastering UDK Game Development HOTSHOT*, все они изданы Packt Publishing. Он также соавтор *UDK iOS Game Development Beginner's Guide*, *Packt Publishing*.

Алессандро Моти (Alessandro Mochi) играет в видеоигры со времен эры Amstrad и NES на всех устройствах: компьютере, консоли и мобильном телефоне. Большие и маленькие видеоигры – его любовь и страсть. Компьютерные ролевые игры (RPG), стратегии, экшен-платформы... ничто не может избежать его внимания.

Профессиональное поле деятельности – информационные технологии, диплом с отличием менеджера проектов, свободно говорит на испанском, итальянском и английском языках, глубокое знание многих программ. Всегда приветствует новые вызовы.

В настоящее время внештатный дизайнер и программист, помогает молодым разработчикам превратить свои идеи в реальность. Хотя он часто путешествует по всему миру, его по-прежнему легко найти через его портфолио на www.amochi-portfolio.com.

Райан Уоткинс (Ryan Watkins**)** любит веселиться. Его можно найти на LinkedIn в www.linkedin.com/in/ryanswatkins.

Предисловие

Книга «*Создание скриптов в Unity*» содержит только существенные сведения и посвящена освоению ряда продвинутых, нетрадиционных и мощных методов, применяемых в написании скриптов для игр на C# в Unity. Это делает книгу очень актуальной, потому что хотя и существует достаточно много литературы для начинающих и учебных пособий по Unity, сравнительно немного в них сказано о наиболее современных темах в выделенном и структурированном виде. Автор книги предполагает, что вы уже знакомы с основами Unity, такими как импорт активов, проектирование уровней, картами освещения и основами написания скриптов на C# или JavaScript. Книга с самого начала посвящена рассмотрению практических задач и примеров того, как творчески использовать скрипты для достижения сложных целей, которые включают в себя такие вопросы, как отладка, искусственный интеллект, настраиваемая визуализация, расширение редактора, анимация движения и многое другое. Главная цель заключается не в рассмотрении абстрактных принципов и теоретических основ, а в том, чтобы показать, как теория на практике, в реальных примерах, поможет вам получить максимум от вашего знания программирования для создания качественных игр, которые не только работают, но и работают оптимально. Чтобы получить максимальную отдачу от этой книги, прочтите каждую ее главу в приведенном в книге порядке, от начала до конца, и при чтении используйте обобщение и абстрактное мышление. То есть посмотрите на каждую главу как на конкретный пример и демонстрацию более общих принципов, которые сохраняются во времени и пространстве, их можно выделить из конкретного контекста, в котором я их использовал, и повторно применить в ином месте, согласно вашим потребностям. Короче говоря, рассматривайте приведенные здесь сведения вне связи с конкретными примерами и выбранной мной тематикой, а как весьма актуальные знания для ваших собственных проектов. Итак, давайте начнем.

Что охватывает эта книга

Глава 1 «*Основы C# Unity*» кратко напоминает основы написания скриптов Unity. Она не является полным или всеобъемлющим руководством по основам языка C#. Скорее, эта глава представляет собой курс повышения квалификации для тех, кто ранее уже изучил основы, но, возможно, не писал скрипты некоторое время и кто был

бы благодарен за краткое напоминание, перед тем как начать работу в следующих главах. Если вы знакомы с основами написания скриптов (такими как классы, наследование, свойства и полиморфизм), то можете пропустить эту главу.

Глава 2 «*Отладка*» глубоко исследует процесс отладки. Возможность писать надежный и эффективный код зависит от вашей способности успешно находить и исправить ошибки при их возникновении. Это делает отладку очень важным умением. В этой главе мы не только рассмотрим основы, но и опишем отладку в интерфейсе MonoDevelop, а также установим полезную систему регистрации ошибок.

Глава 3 «*Синглтоны, статические члены, объекты игры и игровые миры*» рассматривает широкий спектр возможностей для доступа, изменения и управления игровыми объектами. В частности, мы познакомимся с шаблоном синглтона для построения глобальных уничтожаемых объектов, а также со многими другими методами поиска, перечисления, сортировки и размещения объектов. Скрипты в Unity посредством манипулирования объектами в едином игровом мире или пространстве координат обеспечивают реалистичность игр.

Глава 4 «*Событийное программирование*» рассматривает управляемое событиями программирование как важный подход к перестройке архитектуры вашей игры, ведущей к ее оптимизации. Сняв тяжелую нагрузку с часто вызываемых событий обновления и передав ее в вызываемые только при необходимости управляемые события, мы высвободим много ценного времени для решения других задач.

Глава 5 «*Камеры, рендеринг и сцены*» глубоко исследует работу камер, подробно описывает их архитектуру и настройку рендеринга. Мы изучим проверку попадания внутрь области отсечения, вопросы отбраковки, прямую видимость, ортогональную проекцию, глубину, слои, эффекты постобработки и прочее.

Глава 6 «Работа с фреймворком Mono» исследует обширную библиотеку Mono и некоторые из ее самых полезных классов: словари, списки и стеки, другие функции и понятия, такие как строки, регулярные выражения и Linq. К концу этой главы вы сможете быстро и эффективно работать с большими объемами данных.

Глава 7 «*Искусственный интеллект*» содержит пример практического применения почти всего описанного ранее в одном проекте, связанном с разработкой искусственного интеллекта, а точнее создание умного врага, который выполняет широкий спектр действий: ходьба, погоня, патрулирование, нападение, бегство и поиск аптечек для восстановления здоровья. При создании этого персонажа мы

рассмотрим вопросы прямой видимости, обнаружения и прокладки маршрутов.

Глава 8 «*Настройка редактора Unity*» посвящена редактору Unity, функционал которого отвечает многим потребностям, но иногда вам потребуется нечто большее, чего в нем нет. В этой главе мы рассмотрим, как создать редактор классов для настройки самого редактора, чтобы сделать свою работу удобнее и эффективнее. Мы создадим индивидуальные свойства в инспекторе объектов и полнофункциональную систему локализации для разработки многоязычных игр.

Глава 9 «*Работа с текстурами, моделями и 2D*» содержит описание многих возможностей при работе с 2D-элементами, такими как спрайты, текстуры и элементы графического интерфейса. 2D-элементы играют важную роль и в 3D-играх, и здесь мы рассмотрим проблемы работы в 2D и эффективные способы их решения.

Глава 10 «*Управление исходными текстами и другие подсказки*» завершает книгу. Она содержит ряд подсказок, которые не вписываются в какую-либо конкретную категорию, но в целом очень важны. Мы рассмотрим полезные навыки при написании кода, советы по поддержанию чистоты кода, сериализацию данных, интеграцию исходных кодов, контроль версий и т. п.

Что вам понадобится при чтении этой книги

Эта книга, как следует из ее названия, связана с работой в приложении Unity, следовательно, для ее чтения вам понадобится установленная копия приложения Unity. Приложение Unity поставляется со всем необходимым, в том числе со встроенным редактором кода, для выполнения всех описанных в книге примеров. Дистрибутив Unity может быть загружен с сайта http://unity3d.com/. Приложение Unity поддерживает две основные лицензии, бесплатную и профессиональную. Бесплатная лицензия ограничивает доступ к некоторым функциям, однако предоставляет доступ к обширному набору основных функций. В целом большинство глав и примеров в этой книге соответствуют бесплатной версии, это означает, что вы, как правило, можете при выполнении примеров пользоваться бесплатной версией. Тем не менее некоторые главы и примеры требуют наличия профессиональной версии.

Для кого эта книга

Эта книга предназначена для студентов, преподавателей и специалистов, знакомых с основами Unity, а также с основами написания скриптов. Познакомились ли вы с Unity недавно или вы опытный пользователь, у этой книги найдется, что предложить вам, чтобы помочь усовершенствовать методы вашей работы в области разработки игр.

Соглашения

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

Фрагменты кода в тексте, имена таблиц базы данных, имена папок, имена файлов, расширения файлов, адреса страниц в Интернете, пользовательский ввод и указатели Twitter будут выглядеть так: «После создания нового файла скрипта он будет создан в папке `Project` с расширением `.cs`».

Блок программного кода будет приведен в следующем виде:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MyNewScript : MonoBehaviour
05 {
```

Когда мы хотим обратить ваше внимание на конкретный фрагмент блока кода, то соответствующие его строки или элементы будут выделены жирным шрифтом:

```
//We should hide this object if its Y position is above 100 units
bool ShouldHideObject = (transform.position.y > 100) ? true : false;
//Update object visibility
gameObject.SetActive(!ShouldHideObject);
```

Новые термины и **важные слова** будут выделены жирным шрифтом. Слова, которые вы видите на экране, например в меню или в диалоговых окнах, будут выглядеть в тексте так: «Одним из способов является выбор опции **Assets → Create → C# Script** из меню приложения».


Предупреждения или важные сообщения будут выделены подобным образом.


Подсказки и трюки будут выглядеть так.

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.

Загрузка цветных иллюстраций к этой книге

Мы также предоставляем в ваше распоряжение файл формата PDF, который содержит цветные изображения рисунков 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/0655OT_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 этой книги. Если вы нашли опечатку, пожалуйста, сообщите о ней, для этого посетите сайт http://www.packtpub.com/submit-errata, выберите вашу книгу, щелкните по ссылке **Errata Submission Form** и введите описание опечатки. Как только ваше сообщение об опечатке будет проверено, ваше обращение будет принято и опечатка добавлена в общий список опечаток.

Общий список опечаток можно просмотреть, выбрав нужную книгу, на http://www.packtpub.com/support. Требуемая информация будет показана в разделе **Errata**.

Пиратство

Пиратство в отношении авторского права в Интернете является общей проблемой всех издателей. В Packt мы относимся к защите ав-

торских 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

Основы C# Unity

Эта книга о создании скриптов Unity, в частности о создании скриптов на языке C# для разработки игр в Unity. Перед тем как двигаться дальше, необходимо дать определение и квалифицировать понятие создания скриптов. Под созданием скриптов я имею в виду то, что эта книга поможет вам совершить переход от промежуточного теоретического знания к более свободному, практическому и продвинутому овладению скриптами. Здесь ключевым словом является «свободное». С самого начала обучения любого языка программирования неизменно в центре внимания оказывается синтаксис языка, его правила и законы, то есть формальная часть языка. Они включают в себя такие понятия, как переменные, циклы и функции. Однако по мере набора программистом опыта его внимание смещается от самого языка к творческим способам применения языка для решения реальных проблем. Фокус смещается от ориентированных на сам язык задач к вопросам контекстно-зависимого применения. Следовательно, большая часть этой книги не будет посвящена формальному синтаксису языка C#.

В следующих главах я буду считать, что вы уже знакомы с основами языка C#. Вместо основ в книге уже пойдет речь о конкретных применениях и реальных примерах использования C#. Однако сначала в этой главе основное внимание будет уделено именно основам C#. И это не случайно. Эта глава кратко охватит все базовые понятия C#, нужные вам для продуктивной работы в последующих главах. Я настоятельно рекомендую вам прочесть ее от начала до конца независимо от вашего опыта. Она предназначена прежде всего читателям, которые поверхностно знакомы с C#, но стремятся углубить свои знания. Тем не менее она может быть полезной и опытным разработчикам для закрепления существующих знаний и, возможно, для приобретения новых, свежих идей. В этой главе я кратко опишу основы C# с нуля, шаг за шагом. Я буду излагать так, как будто вы уже знакомы с основами программирования, может быть, и на другом языке, но никогда не сталкивались с C#. Итак, начнем.

Почему C#?

Когда дело доходит до скриптов для Unity, перед началом работы над новой игрой всегда возникает вопрос, какой язык выбрать, потому что Unity предлагает выбор. Официально существует два варианта выбора: C# или Javascript. Тем не менее идет дискуссия о том, как JavaScript более правильно называть – «JavaScript» или «UnityScript», – благодаря ряду специфичных приспособлений для Unity, сделанных в языке. Это не должно нас сейчас волновать. Вопрос в том, какой язык должен быть выбран для вашего проекта. Кроме того, может показаться, что у нас есть и еще один вариант выбора – мы можем выбрать оба языка и писать одни файлы скриптов на одном языке, а другие файлы скриптов на другом языке, таким образом эффективно совмещать языки. Это, конечно, технически возможно. Unity не запрещает нам так поступить. Тем не менее это плохо, потому что подобная практика, как правило, приводит к путанице, а также к конфликтам при компиляции, это все равно, что использовать в одном расчете расстояния в милях и километрах.

Рекомендуемый подход состоит в том, чтобы выбрать один из трех языков и применять его последовательно во всем вашем проекте в качестве главного языка. Это сделает процесс работы более эффективным, но это означает, что должен быть выбран один язык, а от других придется отказаться. В этой книге выбран язык C#. Почему? Во-первых, это не потому, что язык C# лучше, чем другие. На мой взгляд, нет абсолютно «лучшего» языка программирования, и нет абсолютно «худшего» языка программирования. Каждый язык имеет свои достоинства и недостатки, и все языки одинаково хорошо применимы для создания игр в Unity. Основная причина в том, что C# является, пожалуй, наиболее широко используемым и поддерживаемым языком в Unity, поскольку он позволяет большинству разработчиков применить в Unity уже имеющиеся у них знания. Большинство учебников по Unity ориентированы на C#, потому что он часто применяется в других областях разработки приложений. Язык C# исторически привязан к структуре .NET, которая используется в Unity (под именем Mono), и язык C# напоминает C++, который очень популярен у разработчиков игр. Кроме того, изучив язык C#, вы обнаружите, что ваши знания и умения востребованы спросом на программистов Unity в современной игровой индустрии. Таким образом, я выбрал C#, чтобы обеспечить этой книге более широкую аудиторию и позволить вам дополнительно использовать обширный набор уже сущест-

вующих учебников и литературы. Этот выбор в дальнейшем позволит лучше применить знания, полученные при чтении данной книги.

Создание файлов скриптов

Если вам нужно определить логику вашей игры или поведение ее персонажей, то вам необходимы скрипты. Написание скрипта в Unity начинается с создания нового файла скрипта, который является стандартным текстовым файлом, добавленным в проект. Этот файл содержит инструкции программного кода, каждая из которых является командой для Unity. Как уже упоминалось, программный код может быть написан на одном из языков: C#, JavaScript или Boo. В этой книге будет применяться язык C#. В Unity есть несколько способов для создания файла скрипта.

Один из них состоит в выборе опции **Assets → Create → C# Script** из меню приложения, как это показано на рис. 1.1.

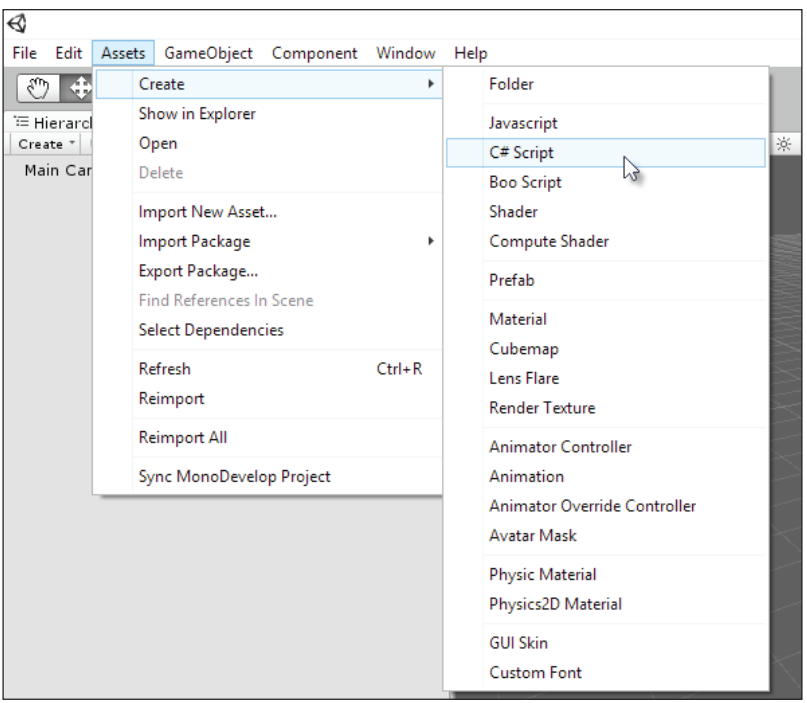

Рис. 1.1. Создание файла скрипта из меню приложения

Другой способ – это щелчок правой кнопкой мыши на пустом пространстве в любом месте панели проекта и выбор опции **C# Script** в меню **Create** из контекстного меню, как показано на рис. 1.2. При этом актив скрипта создается в открытой в данный момент папке.

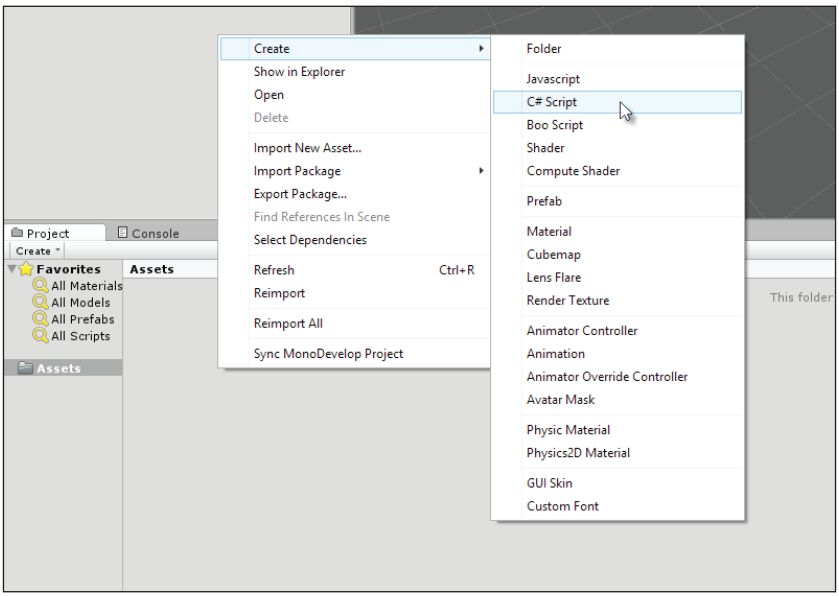

Рис. 1.2. Создание файла скрипта
из контекстного меню панели проекта

После создания новый файл скрипта с расширением .cs (сокращение от C Sharp) будет сгенерирован в папке Project. Имя файла особенно важно, и его смена будет иметь серьезные последствия для корректности файла скрипта, потому что Unity использует имя файла для задания имени класса C#, который создан в файле. Классы будут рассмотрены более подробно ниже в этой же главе. Короче говоря, не забудьте дать файлу уникальное и значимое имя.

Под уникальным именем мы имеем в виду то, что ни один другой файл скрипта в проекте не должен иметь то же имя, независимо от того, в какой папке он находится. Все файлы скриптов должны иметь уникальное имя в рамках всего проекта. Имя должно быть значимым, явно выражая, для чего предназначен ваш скрипт. Кроме того, существуют правила, регулирующие правильность файлов скриптов,

а также имен классов в C#. Формальное определение этих правил можно найти в Интернете по адресу http://msdn.microsoft.com/en-us/library/aa664670%28VS.71%29.aspx. Короче говоря, имя файла должно начинаться только с буквы или символа подчеркивания (цифры для первого символа не подходят), и имя не должно включать в себя пробелов, их рекомендуется заменять символами подчеркивания _ (рис. 1.3).

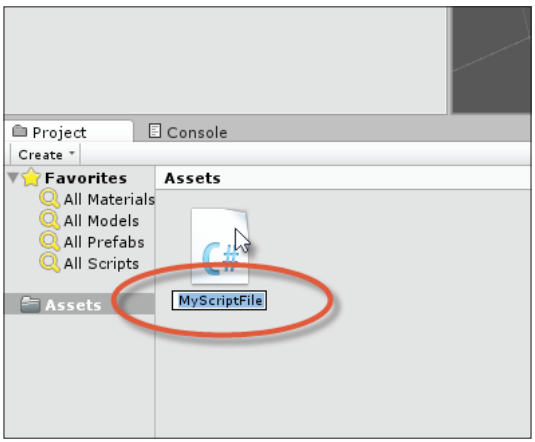

Рис. 1.3. Имя вашего файла скрипта должно быть уникальным и соответствовать принятым в C# соглашениям об именах классов

Файлы скриптов Unity могут быть открыты и их содержимое просмотрено в любом текстовом редакторе или IDE, в том числе в Visual Studio или Notepad ++, но Unity предоставляет бесплатный открытый редактор исходного кода MonoDevelop. Эта программа является частью основного пакета Unity и входит в установочный дистрибутив, но не может быть загружена отдельно. При двойном щелчке по файлу скрипта в панели проекта Unity автоматически откроет файл в MonoDevelop. Если позже вы решите, может быть в силу необходимости, переименовать файл скрипта, то вы должны будете переименовать и класс C# в файле, чтобы его имя в точности соответствовало новому имени файла, как показано на рис. 1.4. Если этого не сделать, то это приведет к ошибке при компиляции кода и проблеме при прикреплении файла скрипта к вашим объектам.

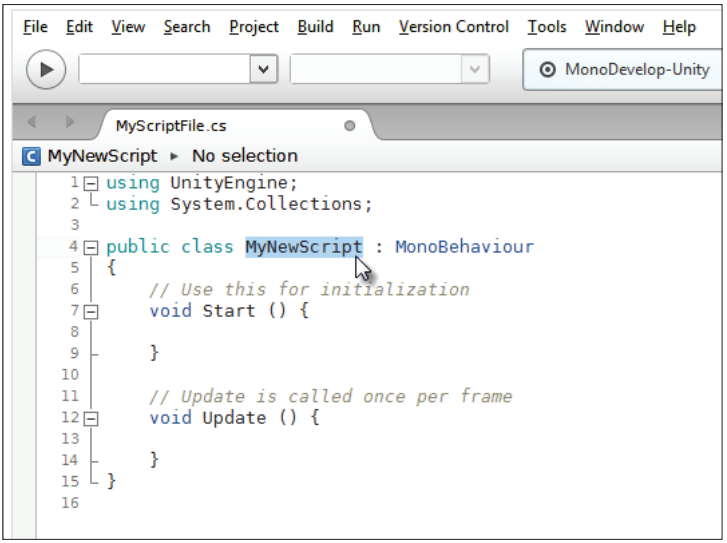

Рис. 1.4. Переименование класса для соответствия имени файла

Компиляция кода

Чтобы скомпилировать код в Unity, вам просто нужно в MonoDevelop сохранить файл скрипта, выбрав опцию **Save** в меню **File** из меню приложения (или нажав **Ctrl**+**S** на клавиатуре), а затем вернуться в главный редактор Unity. При получении фокуса окном **Unity** Unity автоматически распознает изменения кода в файлах и перекомпилирует код. Если при компиляции возникнут ошибки, то игру невозможно будет запустить, и в окне консоли будет выведено сообщение об ошибках. Если компиляция прошла успешно, вам не нужно будет ничего делать для запуска игры, кроме нажатия на клавишу **Play** в панели инструментов редактора, и игра будет запущена в тестовом режиме. Имейте в виду, что если вы забыли сохранить файл из MonoDevelop после внесения изменений в код, то Unity будет по-прежнему использовать старую версию вашего кода. По этой причине, а также в целях резервного копирования важно сохранять ваши коды регулярно, так что не забывайте нажимать **Ctrl**+**S** для сохранения в MonoDevelop.

Присоединение скриптов

Каждый файл скрипта в Unity определяет один основной класс, который можно клонировать, создавая экземпляры. Он представляет собой совокупность связанных между собой переменных, функций и событий (с которыми мы скоро познакомимся). Формально файл

скрипта подобен любому другому виду активов в Unity, такому как меши или аудиофайлы. В частности, он ждет своей очереди в папке проекта и ничего не делает, пока не будет добавлен к определенной сцене (точнее, добавлен к объекту в качестве компонента), где он оживет во время выполнения сцены. Далее, скрипты, будучи по своей сути логическими или математическими активами, не добавляются к сцене как материальные, независимые объекты, как, например, меши. Вы не увидите и не услышите их непосредственно, потому что они не имеют никакого видимого или слышимого отражения. Вместо этого они при добавлении к объектам игры в виде компонентов определяют поведение этих объектов. Процесс вовлечения скрипта в работу как отдельного компонента на конкретном объекте называют созданием его экземпляра. Конечно, из одного файла скрипта можно создать экземпляры для нескольких объектов, если их поведение должно быть похожим, это позволяет не создавать отдельный файл скрипта для каждого объекта, например когда несколько вражеских персонажей должны иметь одинаковый искусственный интеллект. В идеале назначение скрипта – в том, чтобы определить некую абстрактную формулу или модель поведения объекта, которая может быть успешно повторно применена ко многим подобным объектам во всевозможных обстоятельствах. Чтобы добавить файл скрипта к объекту, перетащите его из панели проекта на нужный объект в сцене. Экземпляр скрипта станет компонентом, и его общедоступные переменные станут видны в инспекторе объектов при выборе объекта, как показано на рис. 1.5.

Более подробно переменные будут описаны в следующем разделе.

Более подробную информацию о создании и использовании скриптов в Unity можно найти в Интернете по адресу http://docs.unity3d.com/412/Documentation/ Manual/Scripting.html.

Переменные

Возможно, основным понятием в программировании вообще и в языке C# в частности является переменная. Переменные часто сопоставляют буквам, используемым в алгебре для записи числовых величин, например *X*, *Y* и *Z* или *a*, *b* и *c*. Если вам нужно отслеживать информацию, такую как имя игрока, счет, положение, ориентация, количество боеприпасов, здоровье и множество других видов количественной информации (выраженных существительными), то переменные помогут вам в этом. Переменная представляет собой единичный блок

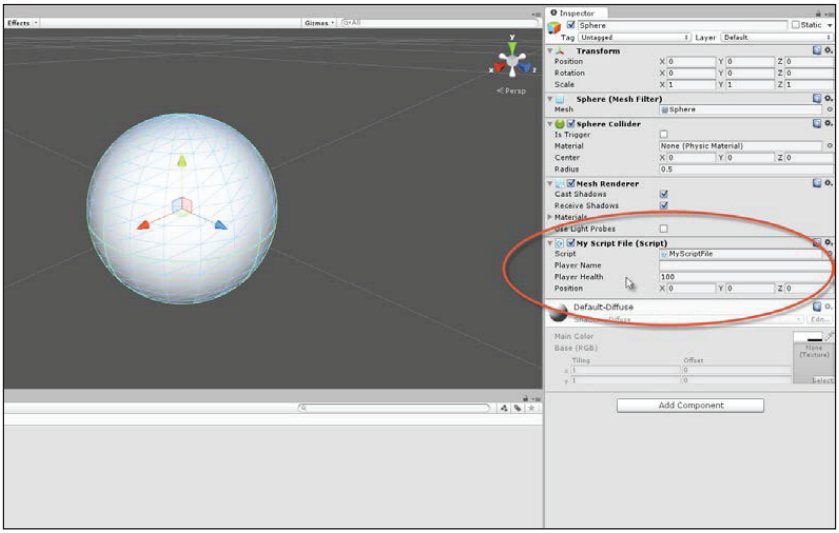

Рис. 1.5. Присоединение скрипта к объекту игры как компонента

информации. Это значит, что для хранения нескольких единиц информации вам потребуется несколько переменных, по одной переменной для каждой единицы информации. Кроме того, каждый блок будет иметь определенный тип или вид. Например, имя игрока представляет собой последовательность букв, таких как «Джон», «Том» или «Давид». Здоровье игрока, напротив, задается в виде цифровых данных, таких как 100 процентов (1) или 50 процентов (0,5), в зависимости от того, какой игроку уже нанесен ущерб. Итак, каждая переменная обязательно имеет тип данных. В C# переменные создаются с помощью специального рода синтаксиса или грамматики. Рассмотрим следующий пример кода 1-1, в котором приведен код файла скрипта, содержащий класс с тем же названием `MyNewScript`, в котором объявлены три различные переменные с областью видимости внутри класса, все разного вида. Слово «объявить» означает, что мы как программисты сообщаем компилятору C# о необходимых переменных:

Пример кода 1-1

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MyNewScript : MonoBehaviour
05 {
```

```
06   public string PlayerName = "";
07   public int PlayerHealth = 100;
08   public Vector3 Position = Vector3.zero;
09
10   // Use this for initialization
11   void Start () {
12
13   }
14
15   // Update is called once per frame
16   void Update () {
17
18   }
19 }
```

Типы данных переменных
Каждая переменная имеет тип данных. Несколько наиболее употребительных из них – это: `int`, `float`, `bool`, `string` и `Vector3`. Ниже представлено несколько примеров переменных этих типов:
`int` (целое число) = –3, –2, –1, 0, 1, 2, 3...
`float` (число с плавающей точкой или десятичное) = –3.0, –2.5, 0.0, 1.7, 3.9...
`bool` (булева или `true`/`false`) = `true` или `false` (1 или 0)
`string` (строка символов) = «hello world», «a», «another word...»
`Vector3` (векторное значение) = (0, 0, 0), (10, 5, 0)...

Заметим для строк кода 06–08 примера 1-1, что каждой переменной присваивается начальное значение и явно указан ее тип данных как `int` (целое число), `string` (строка) и `Vector3`, который представляет собой координаты точки в 3D-пространстве (а также и направление, как мы это увидим ниже). Это не полный список всевозможных типов данных, а только самые распространенные из них, список типов данных будет зависеть от вашего проекта (и вы сможете создавать свои собственные типы!). В этой книге мы будем работать с наиболее распространенными типами данных, так что вы познакомитесь со множеством примеров с их использованием. И наконец, каждая строка объявления переменной начинается с ключевого слова `public`. Как правило, переменные могут быть либо `public`, либо `private` (существует еще один их вид `protected`, но он здесь не рассматривается). Значения общедоступных `public` переменных можно будет изменять и в инспекторе объектов Unity (как мы скоро это увидим, а как это выглядит, можно увидеть на предыдущем рисунке), а также они будут доступными из других классов.

Переменные названы так потому, что их значения могут отличаться (или меняться) в разные моменты времени. Конечно, они не изменяются произвольным и непредсказуемым образом. Они меняются тогда, когда мы явно изменяем их: либо путем прямого назначения

в коде, либо из инспектора объектов, либо с помощью методов и вызовов функций. Они могут быть изменены как непосредственно, так и косвенно. Переменным могут быть присвоены значения непосредственно, например следующим образом:

```
PlayerName = "NewName";
```

Значения им также могут быть назначены косвенно с помощью выражений, то есть операторов, чье окончательное значение должно быть рассчитано до его назначения переменной следующим образом:

```
//Variable will result to 50, because: 100 x 0.5 = 50
PlayerHealth = 100 * 0.5;
```

Область видимости переменных

Каждая переменная объявлена с неявной областью видимости. Диапазон ее видимости определяет время жизни переменной, то есть область внутри исходного файла, где на переменную можно будет успешно сослаться и получить к ней доступ. Эта область определяется местом, где эта переменная объявлена. Для переменных, объявленных в примере кода 1-1, областью видимости является класс, потому что они объявлены в начале класса и вне всяких функций. Это значит, что они доступны во всем классе и (будучи public) доступны и из других классов. Переменные также могут быть объявлены внутри определенных функций. Их называют локальными переменными, потому что область их применения ограничена функцией, то есть локальная переменная не может быть доступна за пределами функции, в которой она была объявлена. Классы и функции будут рассмотрены позже в этой же главе.

Более подробную информацию о переменных и их использовании в C# можно найти по адресу http://msdn.microsoft.com/en-us/library/aa691160%28v=vs.71%29.aspx.

Условные операторы

Значения переменных могут быть изменены во множестве разных ситуаций: игрок поменял свою позицию, враги были уничтожены, при смене уровня и т. д. и т. п. Следовательно, вам необходимо часто проверять значения переменных при выполнении ваших скриптов и выполнять различные наборы действий в зависимости от их текущих значений. Например, если значение переменной `PlayerHealth`, определяющей здоровье игрока, достигнет 0 процентов, вы должны выполнить фрагмент кода смерти игрока, а если значение переменной `PlayerHealth` стало равно 20 процентам, вы, возможно, только отобразите предупреждение. В этом конкретном примере значение переменной `PlayerHealth` развернет скрипт в указанном направлении. Язык C# предлагает два основных условных оператора для обеспече-

ния такого ветвления программного кода. Это операторы `if` и `Switch`. Оба они очень полезны.

Оператор if

Оператор `if` имеет различные формы. Основная форма проверяет условие и выполняет следующий за ним блок кода, если и только если это условие истинно, то есть значение его равно `true`. Рассмотрим следующий пример кода 1-2:

Пример кода 1-2

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class MyScriptFile : MonoBehaviour
05 {
06   public string PlayerName = "";
07   public int PlayerHealth = 100;
08   public Vector3 Position = Vector3.zero;
09
10   // Use this for initialization
11   void Start () {
12   }
13
14   // Update is called once per frame
15   void Update ()
16   {
17     //Check player health - the braces symbol {} are option
       for one-line if-statements
18     if(PlayerHealth == 100)
19     {
20       Debug.log ("Player has full health");
21     }
22   }
23 }
```

Этот код запускается на выполнение, как и все другие виды кодов в Unity, после нажатия кнопки **Play** на панели инструментов и выполняется столько времени, сколько экземпляр файла сценария остается связанным с объектом в активной сцене. Оператор `if` в строке 18 непрерывно проверяет текущее значение переменной `PlayerHealth` класса. Если переменная `PlayerHealth` в точности равна (==) 100, то будет выполнен код внутри скобок `{}` (строки 19–21). Это сработает, потому что условие перед проверкой будет приведено к значению логического типа: либо `true`, либо `false`; условный оператор на самом деле проверяет, что условие (`PlayerHealth == 100`) равно `true`