3 Погружение в переменные, типы и методы

Поначалу в освоении любого языка программирования возникает фундаментальная проблема — вы понимаете набираемые слова, но не их назначение. Обычно в таких ситуациях возникает парадокс, но программирование — особый случай.

C# написан на английском. Несоответствие между словами, которые вы используете в своей жизни, и кодом в Visual Studio возникает из-за отсутствия контекста, который всегда разный. Вы знаете, как произносить и записывать слова, используемые в C#, но не знаете, где, когда, почему и, что наиболее важно, как именно они превращаются в язык программирования.

В этой главе мы отойдем от теории программирования и сделаем первые шаги на нашем пути к реальному программированию. Мы поговорим о принятых правилах форматирования, методах отладки и составлении более сложных примеров переменных и методов. Нам нужно осветить много вопросов, но к моменту, когда вы дойдете до контрольных вопросов, мы рассмотрим такие высокоуровневые вопросы, как:

- пишем на C# правильно;
- отладка вашего кода;
- объявление переменных;
- использование модификаторов доступа;
- понятие области видимости переменных;
- работа с методами;
- анализ распространенных методов Unity.

Пишем на C# правильно

Строки кода работают как предложения, то есть в них должен быть некий разделительный или конечный символ. Каждая строка C#, называемая оператором, обязательно **должна** заканчиваться точкой с запятой, чтобы компилятор мог понять, где заканчивается одна команда и начинается другая.

Однако есть одна особенность, о которой вам нужно знать. В отличие от обычного письменного языка, с которым вы знакомы, оператор C# не обязательно должен располагаться в одной строке. Пробелы и символы переноса строки компилятором игнорируются. Например, простую переменную можно объявить так:

```
public int firstName = "Harrison";
```

а можно и так:

```
public
int
firstName
=
"Harrison";
```

Оба эти фрагмента кода одинаково приемлемы для Visual Studio, но второй вариант крайне не одобряется сообществом разработчиков программного обеспечения, поскольку такой код становится чрезвычайно трудным для чтения. Идея состоит в том, что писать программы нужно как можно эффективнее и понятнее.

Бывают моменты, когда оператор получается слишком длинным и не умещается в одной строке, но это редкость. Но если так произошло, то отформатируйте данный оператор таким образом, чтобы и другие люди могли его понять, и не забудьте точку с запятой.

Второе правило форматирования, которое нужно записать у себя на подкорке, — использование фигурных скобок. У всех методов, классов и интерфейсов после их объявления должна идти пара фигурных скобок. Мы поговорим о каждом из этих элементов более подробно позже,

но продумать стандартное форматирование нужно заблаговременно. Традиционная практика в C# — писать каждую скобку на новой строке, как показано ниже:

```
public void MethodName()
{

}
```

Однако, когда вы создаете в редакторе Unity новый сценарий или открываете документацию Unity, первая фигурная скобка оказывается в той же строке, что и объявление:

```
public void MethodName() {

}
```

Убиваться из-за этого не стоит, но важно соблюдать принятые принципы. В чистом коде C# всегда каждая скобка занимает новую строку, в то время как примеры C#, связанные с Unity и разработкой игр, чаще оформлены так, как показано во втором образце.

Хороший, последовательный стиль форматирования имеет огромное значение, когда вы только начинаете программировать, равно как и возможность наконец увидеть плоды своей работы. В следующем разделе мы поговорим о том, как выводить переменные и информацию прямо в консоли Unity.

Отладка кода

Когда мы будем работать над практическими примерами, нам понадобится способ распечатать информацию и отзывы на панели консоли в редакторе Unity. В программировании это называется отладкой, и в C# и в Unity есть вспомогательные методы, упрощающие данный процесс для разработчиков. Каждый раз, когда я попрошу вас отладить или вывести что-либо на экран, используйте один из методов, которые описаны ниже.

- Для простого текста или отдельных переменных задействуйте стандартный метод `Debug.Log()`. Текст должен быть заключен в кавычки,

а переменные можно использовать напрямую, без добавления символов. Например:

```
Debug.Log("Text goes here.");
Debug.Log(yourVariable);
```

- Для более сложной отладки используйте метод `Debug.LogFormat()`. Он позволяет размещать переменные внутри выводимого текста через заполнители. Они отмечены парой фигурных скобок, каждая из которых содержит индекс. Индекс — это обычное число, начинающееся с 0 и последовательно увеличивающееся на 1.

В следующем примере заполнитель `{0}` заменяется значением переменной `variable1`, `{1}` — значением `variable2` и т. д.:

```
Debug.LogFormat("Text goes here, add {0} and {1} as variable
placeholders", variable1, variable2);
```

Вы могли заметить, что в наших методах отладки мы используем точечную нотацию, и это не случайность! `Debug` — класс, а `Log()` и `LogFormat()` — методы из данного класса, которые мы можем задействовать. Подробнее об этом — в конце главы.

Теперь, зная о возможностях отладки, мы можем смело двигаться дальше и глубже погрузиться в то, как объявляются переменные, а также в различные способы использования синтаксиса.

Объявление переменных

В предыдущей главе мы посмотрели, как записываются переменные, и коснулись высокоуровневого функционала, который они предоставляют. Однако нам все еще не хватает синтаксиса, который позволил бы реализовать этот функционал. Переменные не просто появляются в верхней части сценария C#. Их нужно объявлять в соответствии с определенными правилами и требованиями. На самом базовом уровне оператор переменной должен удовлетворять следующим требованиям:

- необходимо указать тип данных, которые будет хранить переменная;
- переменная должна иметь уникальное имя;

- если мы присваиваем переменной значение, то оно должно соответствовать указанному типу;
- объявление переменной должно заканчиваться точкой с запятой.

Совокупность этих правил дает следующий синтаксис:

```
dataType uniqueName = value;
```

Переменным нужны уникальные имена, чтобы избежать конфликтов со словами, уже используемыми в самом C#, которые называются ключевыми. Вы можете найти полный список защищенных ключевых слов, пройдя по ссылке docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/.

Этот синтаксис прост, аккуратен и эффективен. Однако язык программирования не был бы полезен в долгосрочной перспективе, если бы существовал только один способ создать что-то используемое столь часто, как переменные. В сложных приложениях и играх встречаются разные варианты применения и сценарии, каждый из которых написан с использованием уникального синтаксиса C#.

Объявление типа и значения

Наиболее распространенный сценарий создания переменных — тот, в котором вся необходимая информация указывается сразу при объявлении. Например, если бы мы знали возраст игрока, то написали бы нечто такое:

```
int currentAge = 32;
```

В этой строке выполнены все основные требования:

- указан тип данных — `int` (сокращение от `integer`);
- используется уникальное имя `currentAge`;
- `32` — целое число, соответствующее указанному типу данных;
- оператор заканчивается точкой с запятой.

Однако бывают случаи, в которых нужно объявить переменную, не зная ее значения заранее. Об этом мы поговорим в следующем подразделе.

Объявление типа без значения

Рассмотрим другой случай: вам известны тип данных, которые вы хотите хранить в переменной, и ее имя, а значение неизвестно. Оно будет вычислено и присвоено где-нибудь еще, но вам все равно нужно объявить переменную в начале сценария.

Эта ситуация идеально подходит для вот такого объявления:

```
int currentAge;
```

Здесь определены только тип (`int`) и уникальное имя (`currentAge`), и такой оператор тоже работает и соответствует правилам. Без присвоенного значения переменная получит значение по умолчанию, соответствующее типу переменной. В данном случае `currentAge` станет равно `0`, что соответствует типу `int`. Когда фактическое значение будет известно, его можно легко установить в отдельном операторе, указав имя переменной и присвоив ей значение:

```
currentAge = 32;
```

Полный список всех типов C# и их значений по умолчанию можно найти по адресу docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/builtin-types/default-values.

Здесь вы можете спросить, почему у этих переменных не было ключевого слова `public`, называемого *модификатором доступа*, которое мы видели в предыдущих примерах сценариев. Дело в том, что на тот момент у нас не было почвы для разговора о данном модификаторе. Теперь пришло время вернуться к этому моменту.

Использование модификаторов доступа

Итак, базовый синтаксис нам понятен, поэтому перейдем к более тонким подробностям операторов переменных. Поскольку мы читаем код слева направо, имеет смысл начать подробное изучение переменных с ключевого слова, которое традиционно идет первым, — с модификатора доступа.

Взгляните на переменные, которые мы использовали в предыдущей главе в сценарии `LearningCurve`, и вы увидите, что перед их объявлениями добавлено дополнительное ключевое слово `public`. Это модификатор доступа к переменной, то есть некий параметр безопасности, определяющий, кто и что может получить доступ к информации о переменной.

Любая переменная, не имеющая модификатора public, по умолчанию получает модификатор private и не будет отображаться на панели Inspector в Unity.

Если мы добавим модификатор, то рецепт синтаксиса, который мы собрали в начале этой главы, будет выглядеть следующим образом:

```
accessModifier dataType uniqueName = value;
```

Хотя явные модификаторы доступа при объявлении переменной не всегда нужны, начинающим программистам было бы хорошо завести привычку их добавлять. Это лишнее слово значительно повышает читаемость и профессионализм вашего кода.

Выбор уровня безопасности

В C# есть четыре основных модификатора доступа, но вам, как новичкам, чаще всего предстоит работать с двумя из них:

- `public` — переменная доступна для любого сценария без ограничений;
- `private` — переменная доступна только в классе, в котором создана (он называется содержащим классом (containing class)).

Любая переменная без модификатора доступа по умолчанию является частной.

Есть еще два более сложных модификатора:

- `protected` — доступна из содержащего класса или производных от него типов;
- `internal` — доступна только в текущей сборке.

Каждый из этих модификаторов предполагает свои варианты использования, но пока мы не дошли до более сложных глав, о модификаторах `protected` и `internal` можно временно забыть.

Существует еще два комбинированных модификатора, но в этой книге мы не будем их использовать. Подробнее о них можно почитать, пройдя по ссылке docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/access-modifiers.

Теперь опробуем эти модификаторы доступа!

Время действовать. Делаем переменную частной

Как и в реальной жизни, одни данные необходимо защищать, а другие можно передавать конкретным людям. Если в вашем проекте нет необходимости изменять переменную на панели Inspector или обращаться к ней из других сценариев, то такой переменной можно задать модификатор `private`.

Подредактируем сценарий `LearningCurve`, выполнив следующие действия.

1. Измените модификатор доступа переменной `currentAge` с `public` на `private` и сохраните файл.
2. Вернитесь в Unity, выберите объект `Main Camera` и посмотрите, что изменилось в разделе `LearningCurve`.

Поскольку переменная `currentAge` теперь является частной, она больше не отображается на панели Inspector и доступна только в сценарии `LearningCurve` (рис. 3.1). Если мы нажмем кнопку Play, то сценарий будет работать точно так же, как и раньше.

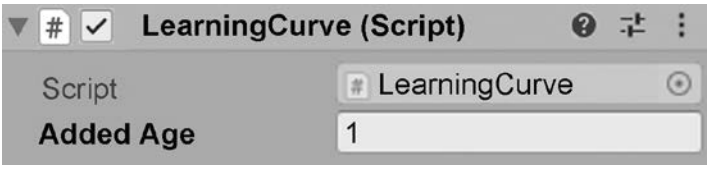

Рис. 3.1

Мы уже много чего знаем о переменных, но нам все еще нужно узнать больше о том, какие типы данных они могут хранить. И в следующем разделе мы поговорим именно о типах данных.

Работа с типами

Присвоение переменной определенного типа — важный выбор, влияющий в дальнейшем на каждое ее взаимодействие на протяжении всего ее жизненного цикла. Поскольку C# — так называемый строго типизированный или типобезопасный язык, абсолютно каждая переменная должна иметь определенный тип данных. Это значит, что существуют четкие правила при выполнении операций с определенными типами, а также правила преобразования одного типа переменной в другой.

Простые встроенные типы

Все типы данных в C# наследуются (или являются *производными*, если говорить как программисты) от общего предка: `System.Object`. Эта иерархия, называемая **системой общих типов** (Common Type System, CTS), означает, что разные типы имеют много общих функций. В табл. 3.1 представлены некоторые из наиболее распространенных вариантов типов данных и хранимые ими значения.

Таблица 3.1

Тип	Содержимое переменной
`Int`	Целые числа, например 42
`Float`	Число с плавающей запятой, например 3,14
`String`	Набор символов в кавычках, например «абыр валг»
`Bool`	Логическое значение — true или false

Помимо указания на значения, которые может хранить переменная, типы содержат дополнительную информацию о себе:

- необходимое для хранения пространство;
- минимальные и максимальные значения;

- допустимые операции;
- расположение в памяти;
- доступные методы;
- базовый (производный) тип.

Если этот объем информации кажется вам слишком большим, то сделайте глубокий вдох. Работа со всеми типами, которые есть в C#, — прекрасный пример использования документации вместо запоминания. Довольно скоро применение даже самых сложных пользовательских типов станет для вас простым и родным.

Вы можете найти полный список всех встроенных типов C# и их спецификации по адресу docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/types/.

Чтобы не застрять с этим огромным списком типов, лучше поэкспериментировать с ними. В конце концов, лучший способ узнать нечто новое — это сделать что-то, сломать, а затем научиться исправлять.

Время действовать. Экспериментируем с типами

Откройте сценарий `LearningCurve` и добавьте в него новую переменную каждого типа, которые мы перечислили выше. Имена и значения можете выбрать сами, но не забудьте добавить им модификатор `public`, чтобы они появились на панели Inspector. Если вам нужно вдохновение, то взгляните на мой код, который показан на рис. 3.2.

При работе со строковыми типами присваиваемое текстовое значение должно находиться внутри пары двойных кавычек, а значения с плавающей запятой должны заканчиваться строчной буквой f — в примере это видно.

Теперь мы видим все использованные типы переменных. Обратите внимание на переменную типа `bool`, которая отображается в Unity в виде флажка (если галочка есть — переменная получает значение `true`, если нет, то `false`) (рис. 3.3).