

Cha4-Unity3d 和 C#的双剑合璧 01

在前面部分的内容中，我们主要是了解了 C#语言的基本语法，以及对编程开发来说一些非常重要的概念。

从这一课的内容开始，我们将学习如何在实际的游戏开发中使用 C#编写游戏脚本。当然，考虑到我们还有很多关于 Unity 的基础知识没有接触过，所以只会用到一些最基本的用法。此外，为了让大家真正的理解和掌握，我们将继续通过实际的小游戏示例来展示。

还等什么呢？让我们开始吧~

个人微信号：iseedo

微信公众号：vrlife

01 Unity 和 C#的双剑合璧-Unity 游戏脚本剖析

在之前的内容中，我们重点了解了 C#语言本身的特点，那么在实际的 Unity3d 游戏中，C#又是如何发挥作用的呢？

首先我们来大概剖析一下 Unity 的游戏脚本构成，从而为进一步学习打好基础。

当然，我们还是用实际的小示例游戏来说明。

开始前的准备-游戏策划

在开发任何一款游戏之前，首先要做的就是策划和设计。

我们人生的第一款游戏是科幻风格的，而接下来的这款游戏，就可以考虑换一种奇幻风格的了~

1.游戏背景和世界观设定：

在维斯卡诺星球上有一片美丽的大陆，那里居住着神奇的魔法生物，过着和平安宁的生活。但是有一天从地底岩洞中出现一个巨大的时空裂缝，冒出来很多奇形怪状的怪物。作为精灵族的勇士，你需要击败这些怪物，从宝箱中集齐九大神器碎片，然后回族中的圣坛合成精灵族的镇族神器，彻底将怪物驱逐出这个时空。

2.游戏规则：

在游戏场景中漫游寻找宝箱，击败怪物，打开宝箱并找到神器碎片。

好了，既然游戏已经策划完成，让我们开始动手吧。

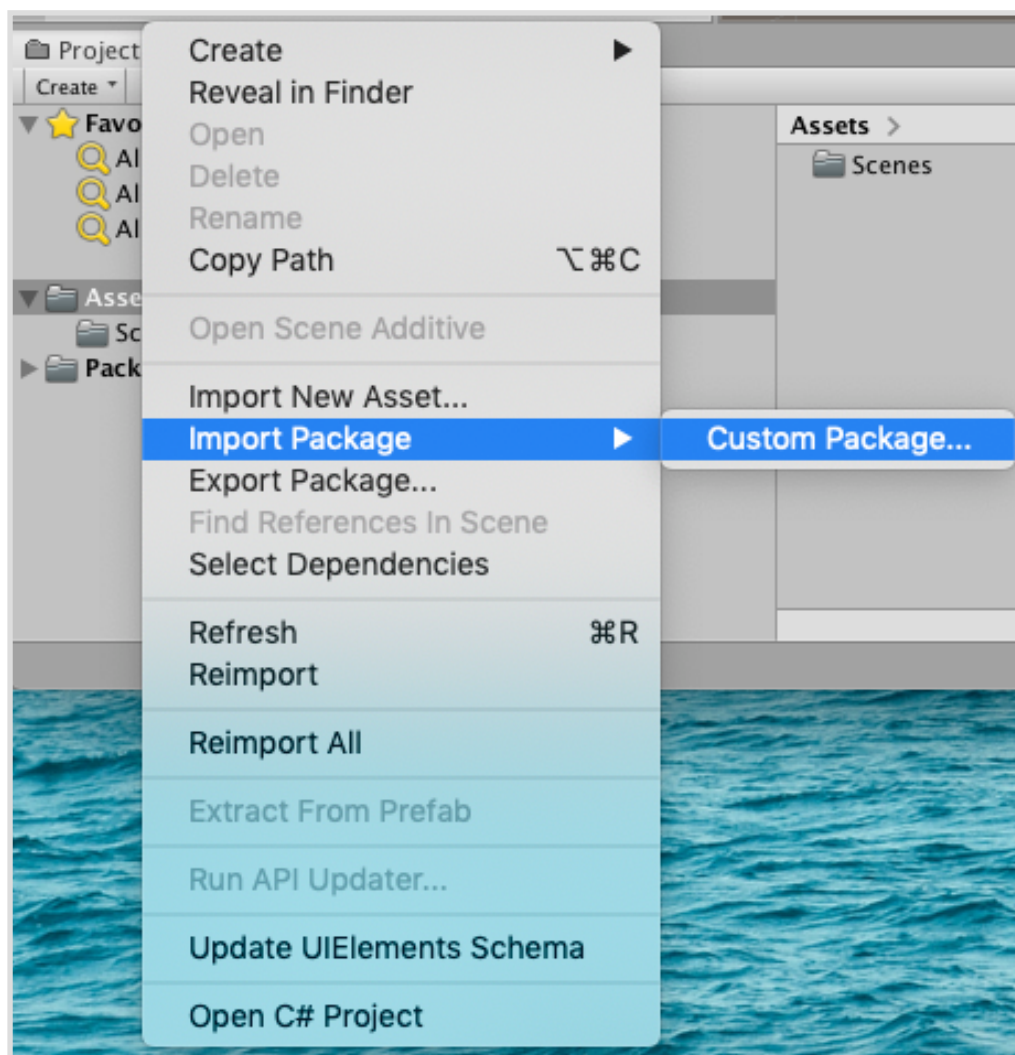
Step1.创建新项目

打开 Unity Hub，点击 New 创建一个全新的项目，把它命名为 FairyLand，点击 Create Project 创建项目。

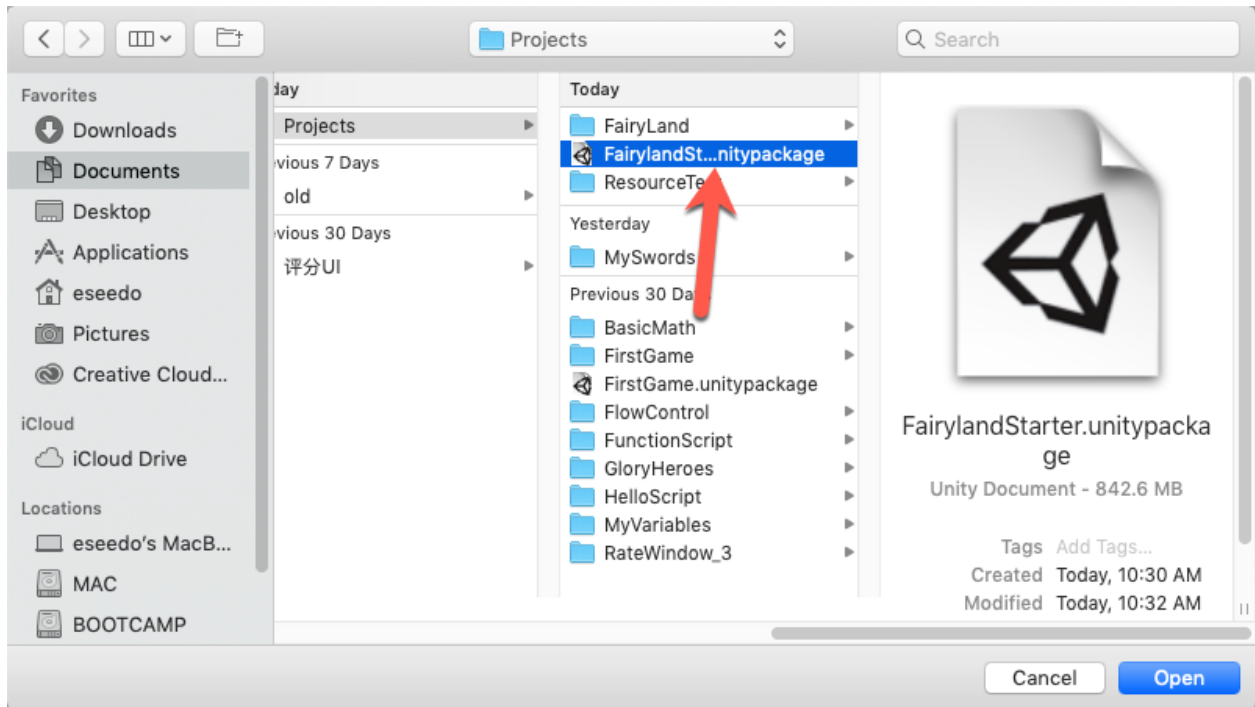
Step2.导入游戏资源

为了方便起见，我为这款游戏提前准备好了一些游戏资源，包括场景，角色，动画和音效等。

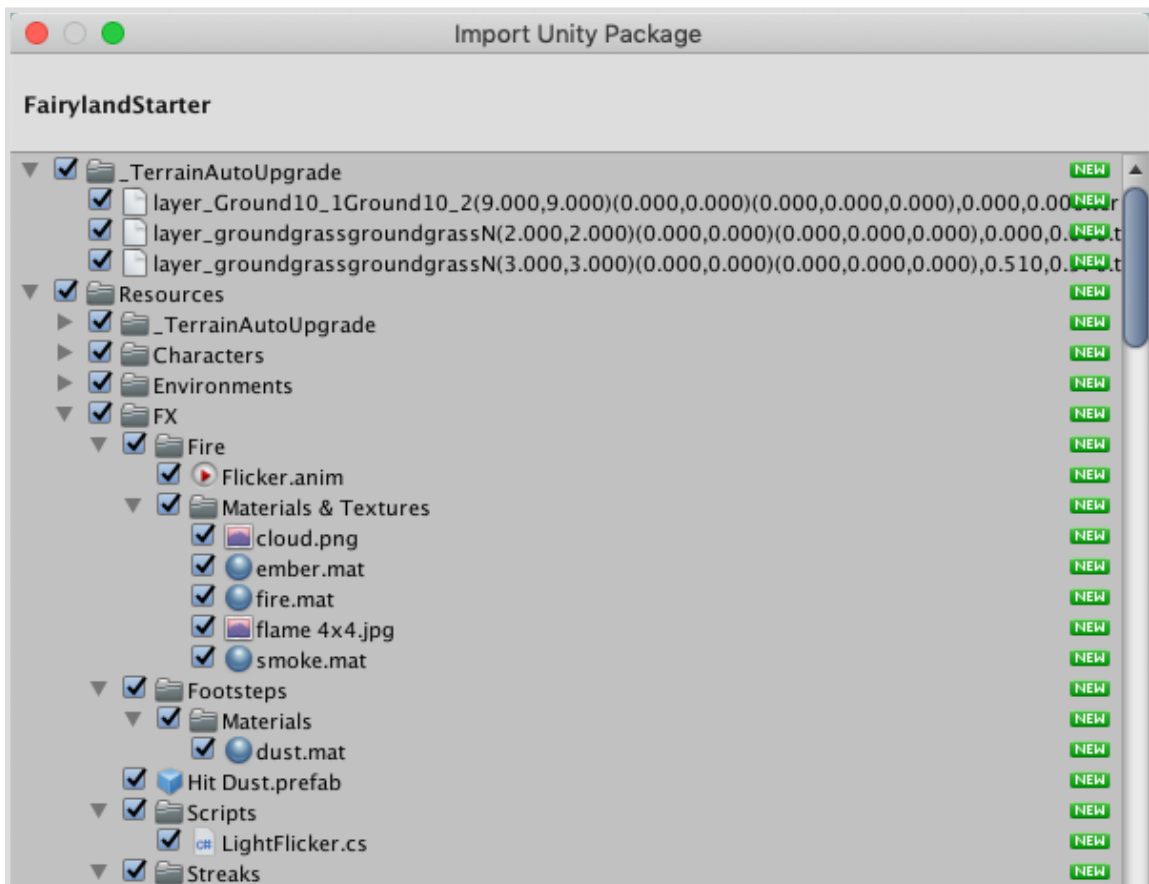
在 Unity 编辑器中右键单击 Assets，选择 Import Package- Custom Package...



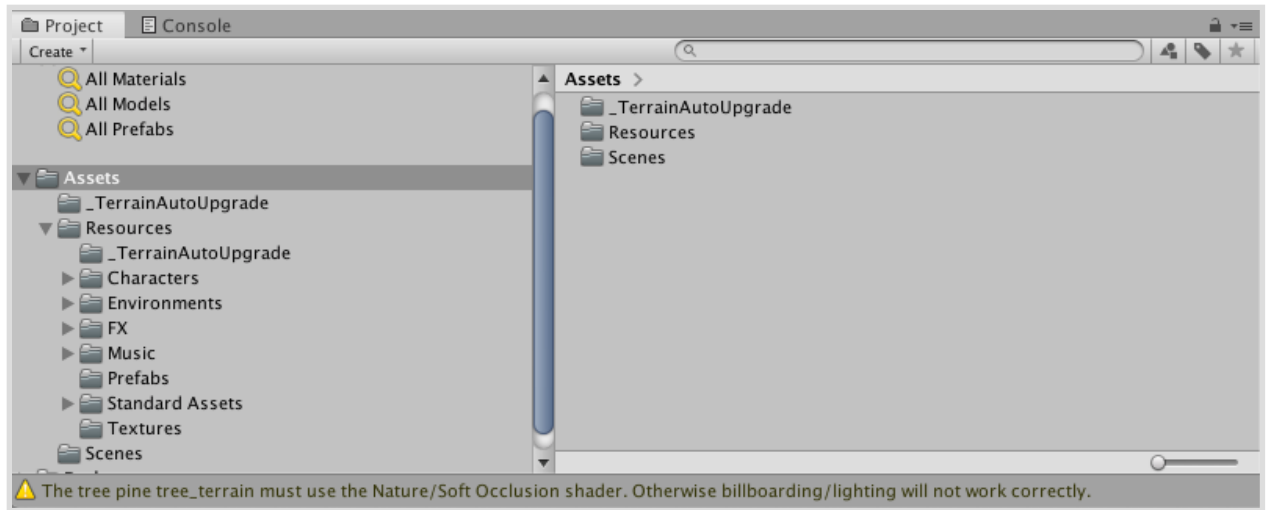
找到 FairyLandStarter.unitypackage 文件，点击 Open。当然，具体的存放位置取决于你自己。



在弹出的对话框中点击 All，然后点击 Import 导入项目中。



导入完成后的 Project 视图是这样的：



Step3 创建游戏脚本并关联游戏对象

好了，项目的准备工作已经就绪，接下来该是剖析 Unity 游戏脚本的时候了。

在 Project 视图中右键单击 Assets，选择 Create-Folder，创建一个新的文件夹，并命名为_Scripts。

在 Hierarchy 视图中点击 Create- Create Empty,创建一个新的空白游戏对象 GameObject。点击选中 GameObject，在 Inspector 视图中点击 Add Component，选择 New Script，取名为 ExampleScript，然后点击 Create and Add。可以看到，现在 GameObject 就和一个名为 Example Script(Script)的组件关联在一起了。

当然，回到 Project 视图，会看到刚才所创建的脚本文件并没有出现在_Scripts 文件夹中。是因为通过这种方式创建的游戏脚本默认在 Assets 的根目录下。所以我们需要手动把 ExampleScript 拖动到 _Scripts 文件夹中。

ok,一切就绪，接下来可以进入代码时间了~

双击 ExampleScript，在 Visual Studio 中将其打开。

在之前的学习中，我们已经很多次看到默认创建的脚本了。但是这一次，我们会逐行来剖析一下，每一行代码到底是干嘛用的~

为了方便起见，请大家直接根据编辑器中代码前的行编号来查看。

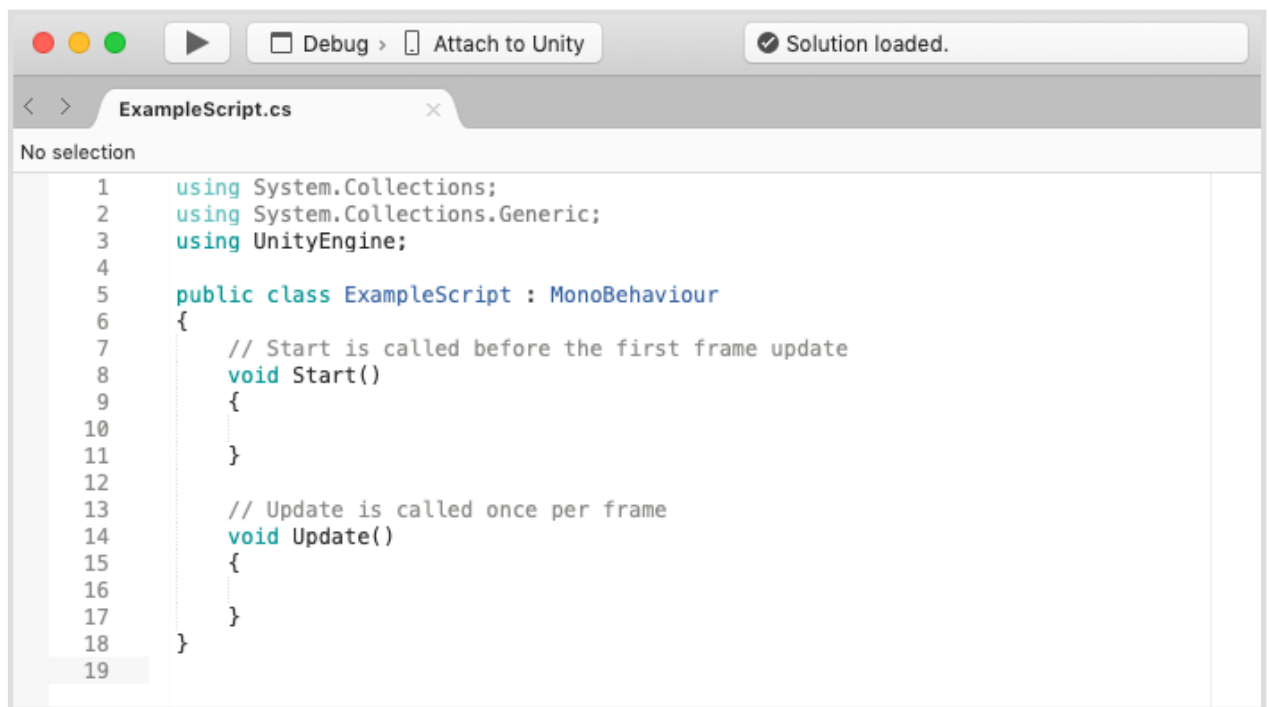
```
1. using System.Collections;
```

其实这一行代码和接下来的两行代码作用类似，都是为了在脚本文件中引用特定的 namespace (命名空间)。

那么问题来了，什么是 namespace (命名空间)？

在 C# 中，namespace 用来将很多类组织在一起。

比如：



在不使用命名空间的前提下输出一行信息，可能会使用下面的代码：

```
System.Console.WriteLine("Hello World!");
```

其中 System 就是一个命名空间，而 Console 是该命名空间中的一个类。如果我们使用 using 关键字，那么这一行代码就可以简化了。

```
using System;
```

```
Console.WriteLine("Hello World!");
```

除了使用系统所提供的命名空间，我们也可以声明自己的命名空间，来控制类和方法名称的范围，这里就先不具体举例讲了~

```
namespace MyNameSpace
```

```
{
```

```
class MyClass
```

```
{
```

```
public void MyMethod()
```

```
{
```

```
    System.Console.WriteLine("MyMethod inside MyNameSpace~");
```

```
}
```

```
}  
  
}
```

命名空间具有以下属性：

- (1) 使用 namespace 组织大型代码项目。
- (2) 通过使用 . 运算符分隔 namespace。
- (3) using 指令可免去为每个类指定命名空间的名称。
- (4) 大的 namespace 中还可以有子 namespace。

考虑到 C#语言是由微软开发出来的，建议大家可以看看官方的详细解释~

<https://docs.microsoft.com/zh-cn/dotnet/csharp/programming-guide/namespaces/index>

好了，有了这些理论基础铺垫，我们再来看看这行代码的作用：

```
using System.Collections;
```

简单来说，这个命名空间中包含了 C#中可使用的集合类和相关的接口，这些接口和类可以用来定义各种对象（比如列表、队列、位数组、哈希表和字典等）。

注意 System.Collections 这个命名空间不是 Unity3d 游戏所特有的，所以如果有童鞋后面有机会尝试.NET 开发，那么也会碰到的~

至于 System.Collections 里面具体包含了哪些东西，这个就有点复杂了。考虑到大家都是小白，先不用这么深入，但是感兴趣的童鞋也可以参考微软的官方文档：

<https://docs.microsoft.com/en-us/dotnet/api/system.collections?view=netframework-4.8>

然后看紧接着的这行代码：

```
2. using System.Collections.Generic;
```

好了，聪明的童鞋应该可以猜到，这里我们导入了又一个命名空间，而且它是刚才的

System.Collections 命名空间的子命名空间~

System.Collections.Generic 命名空间包含定义泛型集合的接口和类，泛型集合允许用户创建强类型集合，它能提供比非泛型强类型集合更好的类型安全性和性能。

关于什么是泛型，这里先不多讲，但后面很快我们就会遇到使用泛型的情况~

关于 System.Collections.Generic，感兴趣的童鞋可以参考：

<https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic?view=netframework-4.8>

那么问题来了，既然 System.Collections.Generic 命名空间中的接口和类理论上都包含在

System.Collections 这个命名空间中，为什么还要多此一举呢？

举个例子吧，可能不是很贴近，但大家或许有个大概的概念。

比如我住在广东省深圳市南山区的人民路 13 号。

那么，假如 System 对应的命名空间是广东省，那么 System.Collections 对应的命名空间就是广东省深圳市，而 System.Collections.Generic 对应的命名空间是广东省深圳市南山区。

这样，当我说自己住在人民路 13 号的时候，就必须先引用 System.Collections.Generic 这个命名空间。

反过来，为什么有了 System.Collections.Generic，还要有 System.Collections 这个命名空间的引用呢？这个就更简单了。因为或许 System.Collections（广东省深圳市）的宝安区有个龙舟路，而南山区并没有这样一条路。

或许大家看了还是一脸懵逼，但是没关系，在学习的过程中我会反复灌输这些基本的概念，直到大家完全可以理解为止。

好了，紧接着这两行代码之后的是这样一行代码：

3. `using UnityEngine;`

如果说之前的两个命名空间是 C# 开发所通用的，那么 UnityEngine 这个命名空间中包含的就是开发 Unity 游戏所特有的类。

通过引用 UnityEngine 这个命名空间，我们的 ExampleScript.cs 才成为了真正的游戏脚本。

UnityEngine 可以说是编写 Unity 游戏脚本的灵魂所在，其中包含了跟游戏脚本开发相关的一切，包括 AI，对 iOS 和 Android 的支持，音频处理，性能诊断分析，事件系统，Jobs 系统，网络系统，渲染系统，场景管理，序列化，社交平台，2D 精灵对象，UI，视频，XR 支持等等。

感兴趣的童鞋可以在浏览器中打开以下链接：

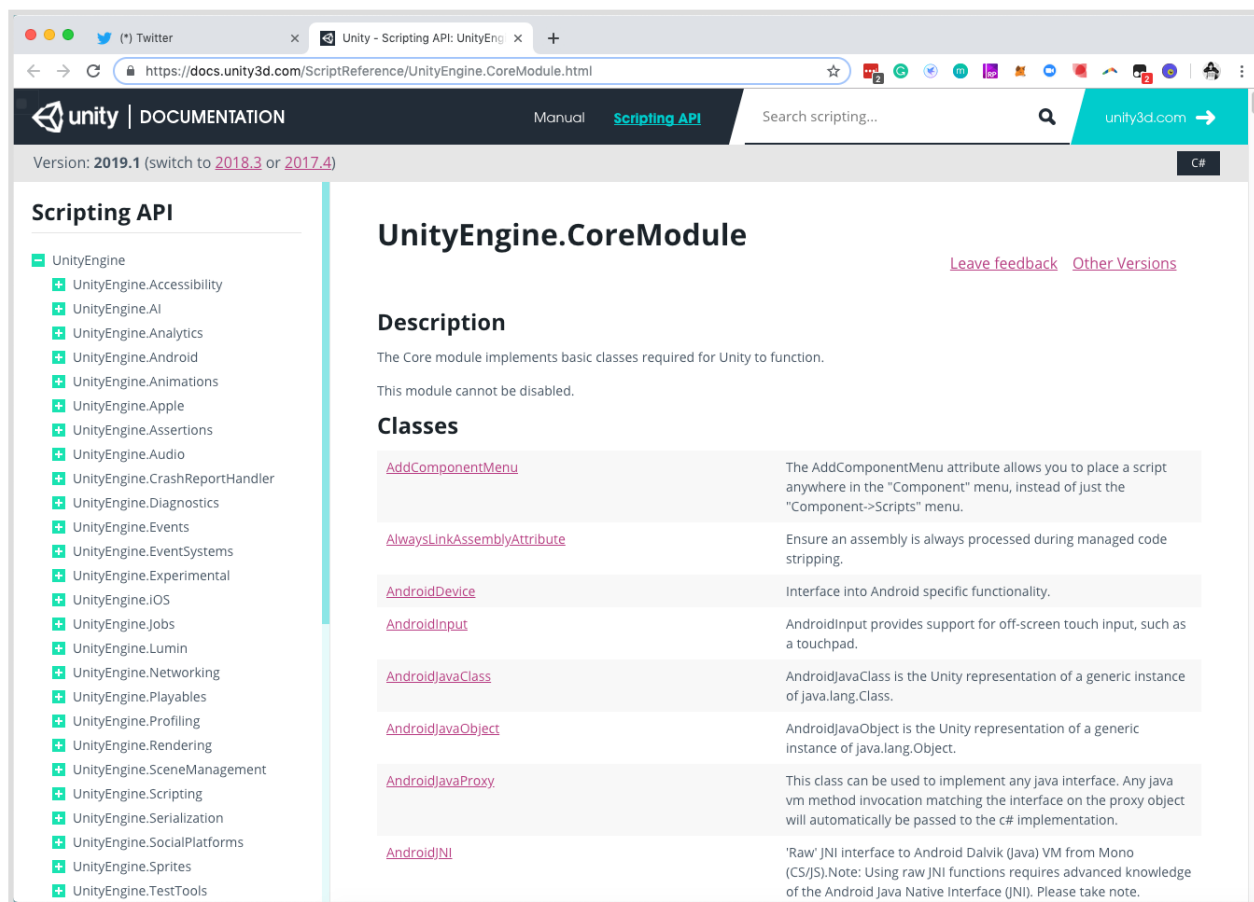
<https://docs.unity3d.com/ScriptReference/UnityEngine.CoreModule.html>

另外提醒大家的是，在我们学习 Unity3d 开发的过程中，官方文档的 Manual 和 Scripting API 是最大的得力助手~其次才是 stack overflow, google，百度，讨论社区啥的。

再往下看：

```
5.public class ExampleScript: MonoBehaviour
```

这行代码的作用比较直接，我们定义了一个 public 类型的类，名为 ExampleScript，该类继承自 MonoBehaviour。



如果细心的童鞋可能会发现，在 Visual Studio 的编辑器中，代码有着不同的色彩。

比如 using System.Collections;这行代码中，

using 是绿色的，而 System.Collections 则是灰色的。

默认情况下，Visual Studio 会把 C#语言的关键字设置为绿色的。

什么是关键字？也就是一门编程语言为自己预留的词。这些词对这门编程语言来说有一些极其重要和特殊的作用，开发者在定义自己的变量、函数、方法和类的时候是禁止使用这些关键字的~

不光是 C#语言如此，其它的所有编程语言，从晦涩难懂的汇编语言到简单易懂的 Python 甚至是 Scratch，莫不如是。

而默认情况下，类似 `System.Collections` 这种所引用的系统命名空间则是灰色的~

从第三行代码 `using UnityEngine;` 开始，我们终于见到了正常的黑色字体。默认情况下，在 Unity 游戏脚本中，黑色字体通常代表 Unity 系统的相关类、方法、函数等。

回到 `public class ExampleScript : MonoBehaviour` 这行代码，这里用淡蓝色的字体代表类的名称~

好了，我们已经知道 `public class` 是系统预留的关键字，意思是我们要创建一个 `public` 类型的类。而 `ExampleScript` 则是类的名称，这个也好理解。那么 `MonoBehaviour` 究竟是个什么东西？

简单来说，Unity 中的所有游戏脚本，要是想跟游戏对象关联在一起，就必须继承自 `MonoBehaviour`，它是 Unity 游戏脚本类的基类。

默认情况下，我们在 Unity 编辑器中所创建的所有游戏脚本都会自动继承自 `MonoBehaviour`，除非手动删除~（在之前的内容中，我们曾经尝试过）

问题来了，`MonoBehaviour` 并非 C# 的系统类，那么 Visual Studio 作为一个第三方的编辑器，是如何正确识别的呢？

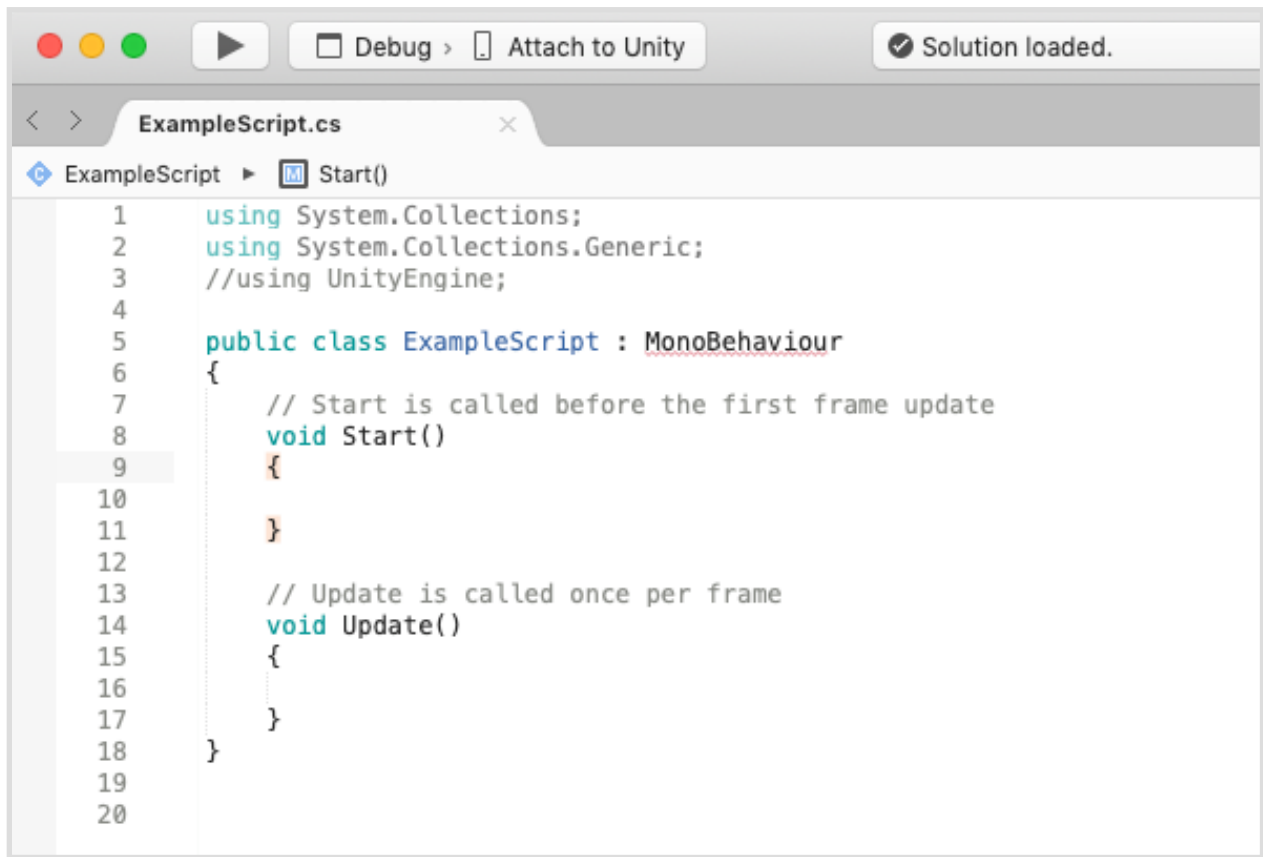
这个时候，第三行代码的作用就体现了。

```
using UnityEngine;
```

通过使用这行代码，我们引用了 Unity 的核心系统类库，让 Visual Studio 知道 `MonoBehaviour` 究竟是个什么东西。

大家可以手动把 `using UnityEngine;` 这行代码注释一下看看

可以看到，一旦这行代码被注释掉，紧接着定义 `ExampleScript` 类的这行代码中，`MonoBehaviour`



The screenshot shows a Visual Studio window with a C# script named `ExampleScript.cs`. The script contains the following code:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 //using UnityEngine;
4
5 public class ExampleScript : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
19
20
```

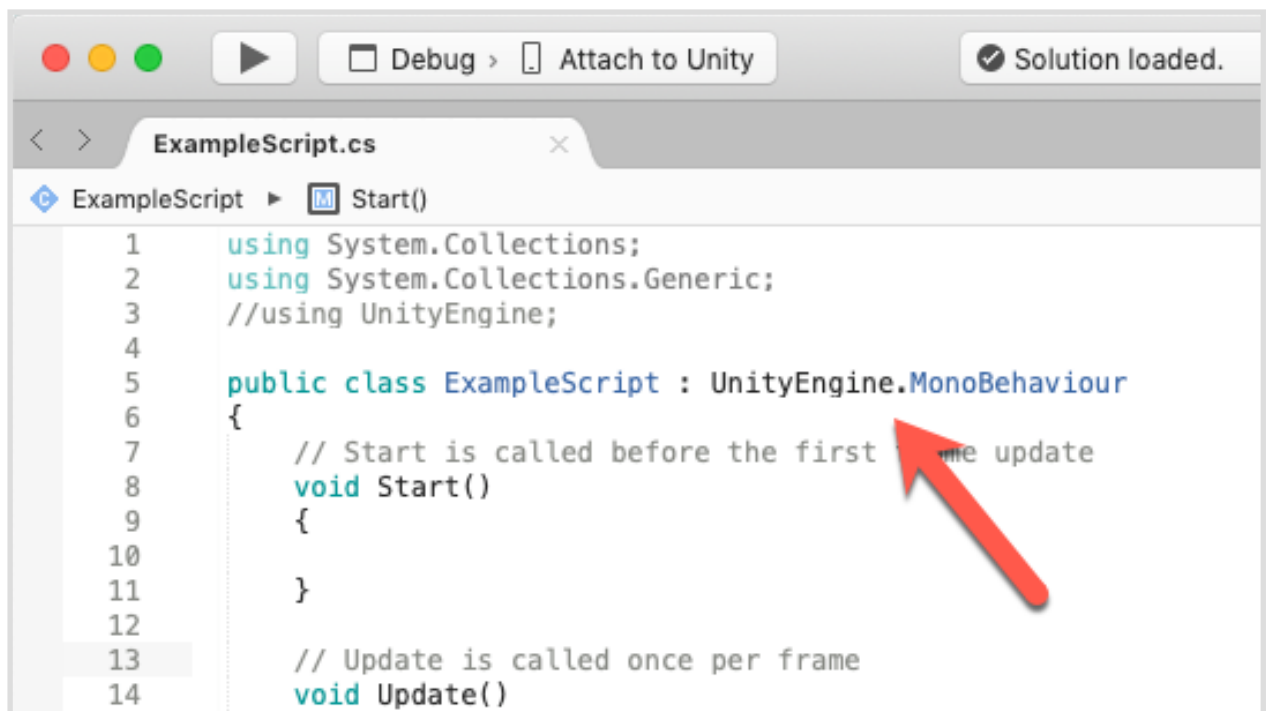
The `UnityEngine.MonoBehaviour` type is not recognized, indicated by a red squiggly line under `MonoBehaviour` on line 5.

的下方出现了红色的波浪线，代表系统根本不认识什么是 `MonoBehaviour`~

此时，我们有两种选择，一种是取消注释，而另一种则是在 `MonoBehaviour` 的前面加上 `UnityEngine`：

当然，正常情况下我们只需要取消注释就好了，而这，恰好说明了命名空间的作用。

说白了，你可以说自己是文科状元，但一定要限制在某某省某某市某某县某某学校，出了这一亩



The screenshot shows the same Visual Studio window, but the script has been corrected. The `using UnityEngine;` line has been added on line 3, and the `MonoBehaviour` type is now fully qualified as `UnityEngine.MonoBehaviour` on line 5. A red arrow points to the `UnityEngine` namespace prefix.

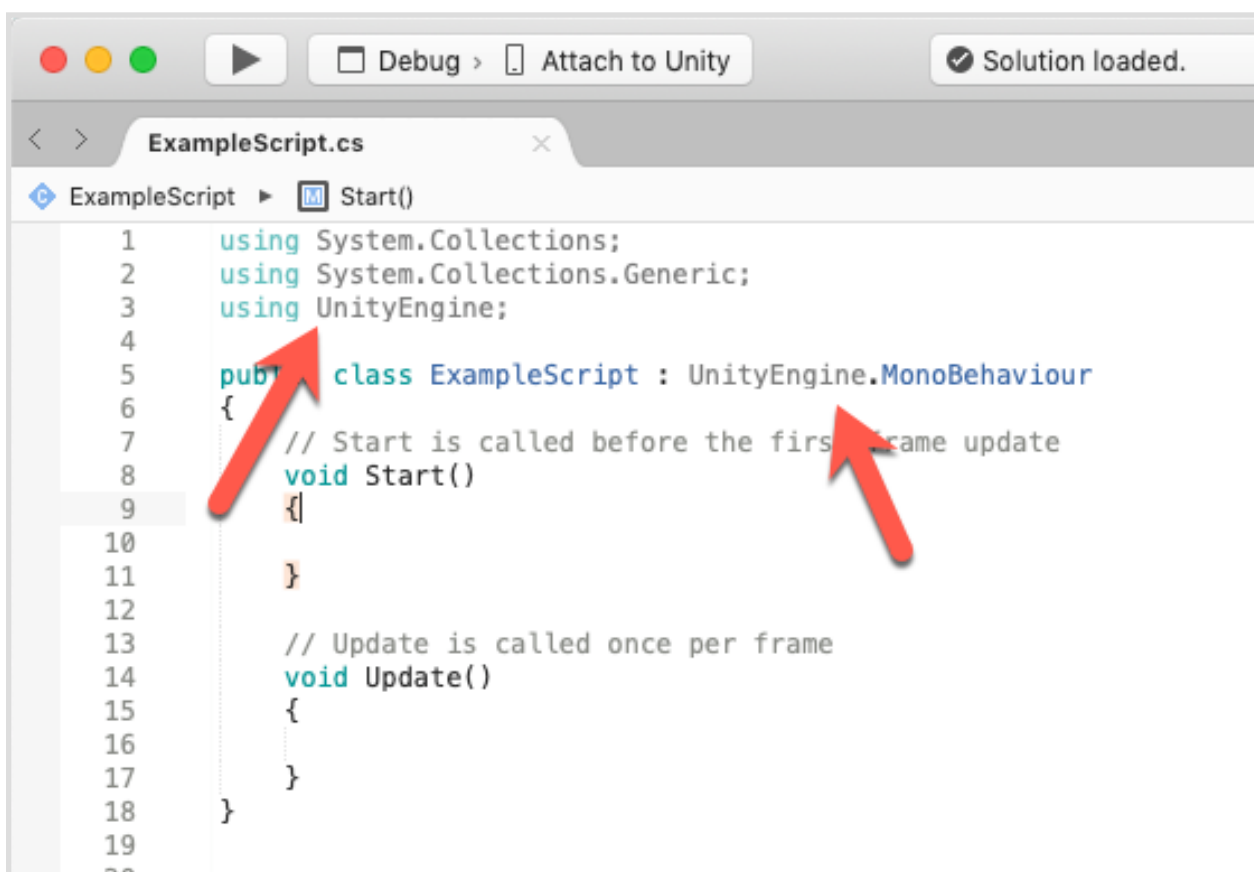
```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class ExampleScript : UnityEngine.MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
19
20
```

三分地，还敢这么说自己，谁都不会理会的~

有些童鞋可能是杠精转世，或许会问，如果我前面引用了命名空间，然后这里仍然加上前缀，会不会起冲突？

答案是，显然不会。

如果你每天闲得无聊靠数毛爷爷来打发时光，那你大可以这样做，毕竟动动手也算是一种身体运动啊~



相比大家对什么是命名空间，以及命名空间是如何起作用的，已经有了比较直观的理解，那么接下来我们再来复习一下什么是类。

之前我们提到过，类就是一个容器，它里面放置的是变量和方法。通过使用类，可以更好的组织代码，实现规模宏大威力惊人的游戏项目~

这里需要特别提醒大家的是，这里我们所定义的类的名称 `ExampleScript` 和游戏脚本的文件名 `ExampleScript.cs` 去掉后缀后必须是完全一致的，否则 Unity 是不会认的~

虽然默认情况下这个不是问题，但总有些手贱的童鞋一不小心修改了脚本的文件名，或者修改了类的名称，导致两个名字对不上了，那么这个时候就会崩了~

在游戏开发的过程中，灵活有效的使用类，可以极大的提升开发效率。在之前的学习中，我们了解过类的继承概念，并以《王者荣耀》为例做了说明。

在实际的开发过程中，除了使用继承，我们还可以用到另一种思路，也就是组合 (composition)。比如游戏中可能我们需要定义一个 `Player` 类代表玩家，这个 `Player` 类有很多属性，同时还具备一系列的技能和方法。但是如果把所有的属性和方法都写在一个类的代码中，会让这个类变得异常复杂。那么我们就可以考虑使用组合，也就是把一个复杂的类拆开成若干个小一些的类，每个类都用来完成某些特定的任务。比如 `PlayerMovement` 类专门用来处理玩家运动，`PlayerShooting` 类专门用来处理玩家的射击，等等。

不过这些思路如果只是纸上谈兵的话，大家可能也很难理解，我们后面还是用实战案例来具体说明~

OK，看懂了以上的代码，在所定义的 `ExampleScript` 类的大括号中，默认情况下会有两个 Unity 自动提供的系统方法 (函数)，通常我们将其称之为事件函数。

我们可以在大括号中添加自定义的变量和方法，这些在后面的内容中逐渐展开。

通过系统提供的注释，我们知道 `Start` 方法会在游戏开始的时候调用，而 `Update` 方法则会在游戏的每一帧调用。

```
void Start(){}
```

在上面的代码中，`void` 代表 `Start` 方法不返回任何数值，只是执行某些特定的任务。

```
void Update(){}
```

同样的，`Update` 方法也不会返回任何数值，只是执行某些特定的任务。

Unity 的系统方法 (事件函数) 对于我们开发游戏脚本非常重要，在下一课的内容中，会进行更加详细的解释和说明。

好了，看到这里，大家应该可以基本理解一个新创建的游戏脚本中每一行代码的作用了。大家可以尝试做一些修改，或是添加自己的内容，看看会发生些什么~

让我们下一课再见~