Roll A Ball

——熟悉Unity软件的基本操作

**提示**：在本实验文档中存在三种不同颜色的悬浮框：

1. 橙色悬浮框：伴随**注意**字样，你需要仔细阅读该悬浮框中的内容，并严格遵守。若不遵守这类信息，有可能会对自己与他人的实验进度，仪器甚至是人身安全造成影响。
2. 绿色悬浮框：伴随**提示**字样，该类信息将为你阅读实验文档以及进行实验提供必要的帮助，你不必严格遵守，但通常该类信息会帮助你更轻松地完成实验。
3. 蓝色悬浮框：伴随**问题**字样，你需要仔细阅读这些问题，并试图做出合理的解释，有的问题需要你在实验过程中找到答案。

## 实验目的

1. **学会创建新的Unity项目与游戏场景**
2. **学会添加游戏对象**
3. **学会基本的脚本语言C#，并通过编写脚本来达到各种效果**
4. **实现简单的计分板UI**
5. **学会发布项目**

## 实验软件/硬件

1. **Unity 2018.4.17f1 LTS版本。**

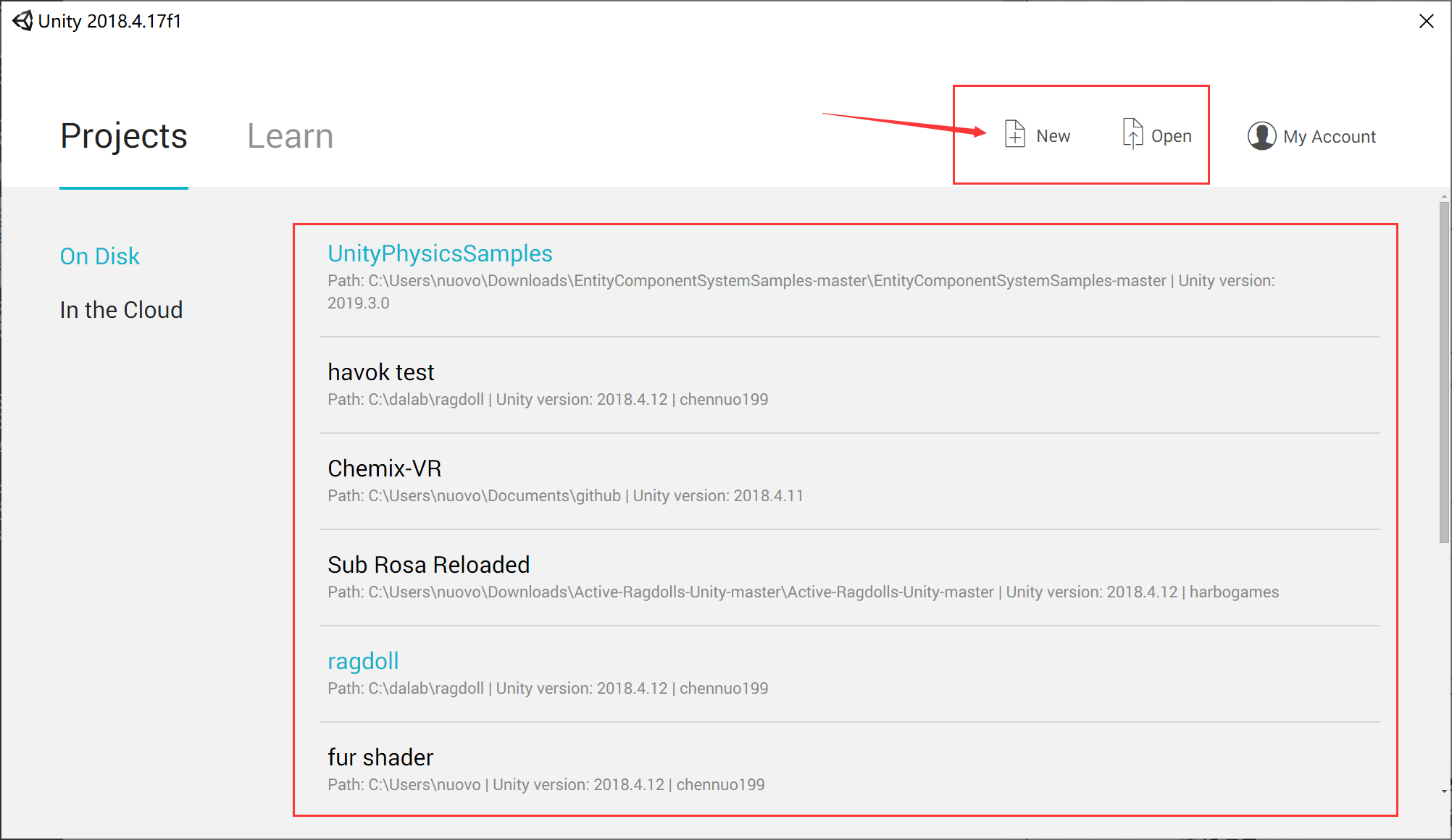
为了大家以后团队作业的兼容性、引擎稳定性的考虑，请大家统一使用Unity 2018.4.17f1 LTS版本。在本课程的 页面tab下有安装Unity的指引（<https://oc.sjtu.edu.cn/courses/17606/pages/installing-unity?module_item_id=26597>）。

## 实验内容

### 1 创建Unity项目与项目场景（此部分5分）

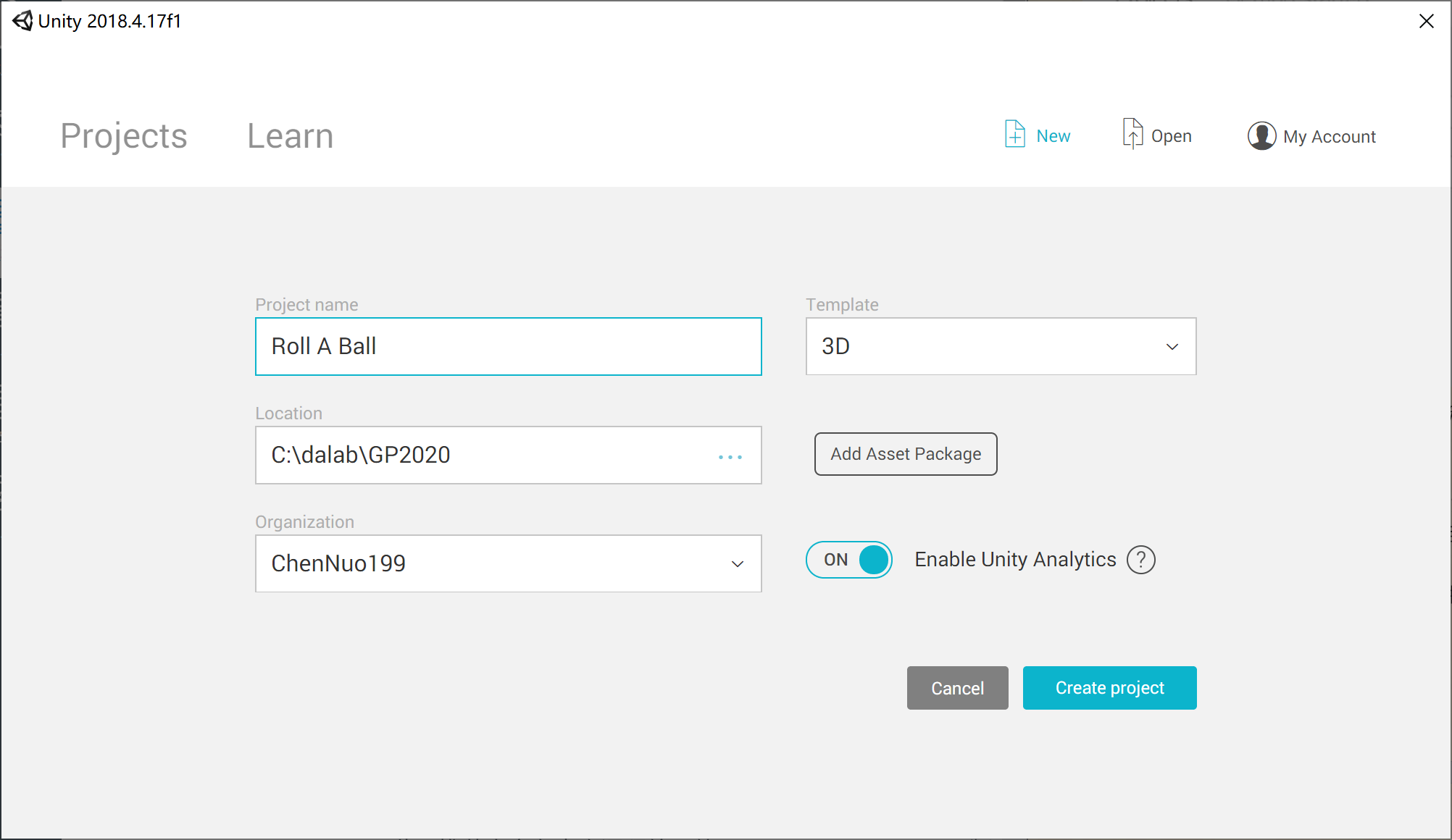
#### 1.1 创建Unity项目

首先点击桌面上的Unity图标，你会看到图1所示的Unity界面，其右上角存在New和Open两个选项，同时，你也可以看到中间存在多个项目条目。点击Open可以指定打开一个项目，点击New可以打开一个新项目，而点击中间的对应条目，则会对应打开这个最近使用过的项目。这里我们点击New来新建一个项目。



1. *初始Unity界面示意图*

点击后，出现图2所示的界面，我们输入项目名称、存放路径以及项目模式，这里我们要建立3D的项目，所以对应选择3D。然后点击Create project按钮创建新项目。这之后等待一小段时间使Unity初始化项目。



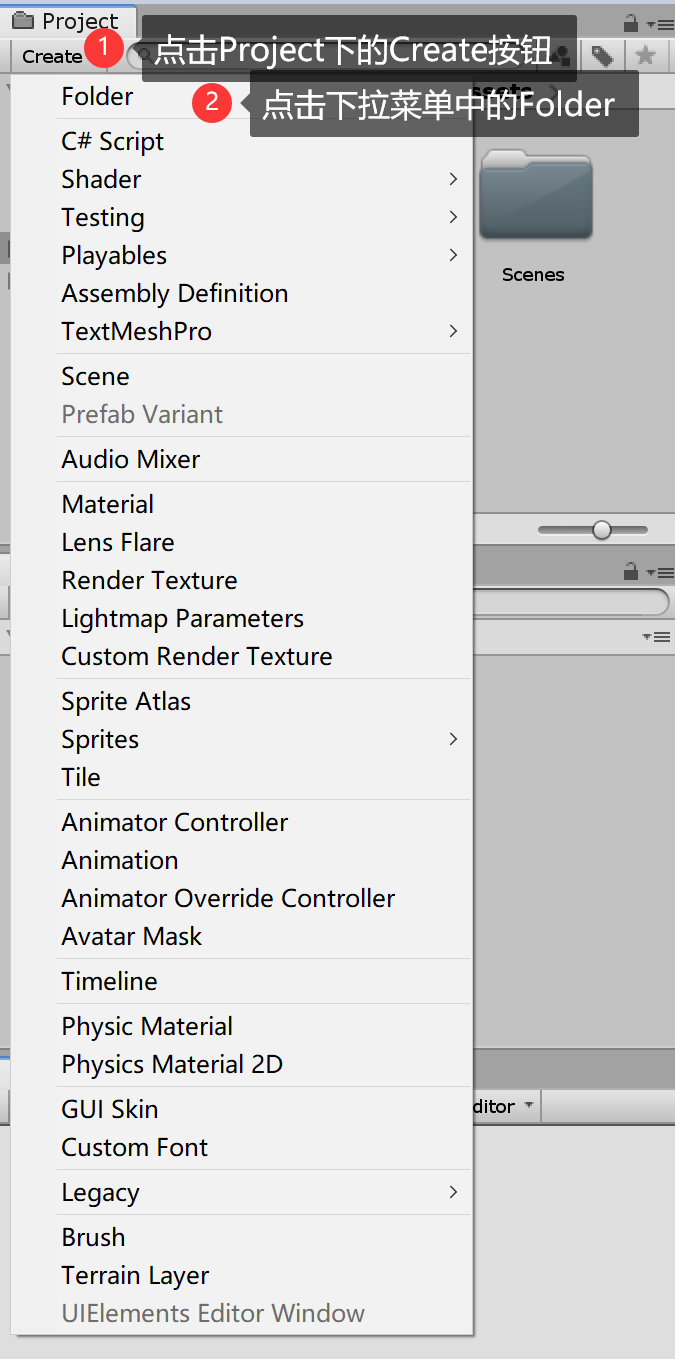
1. *新建工程与设置*

#### 1.2 创建项目场景

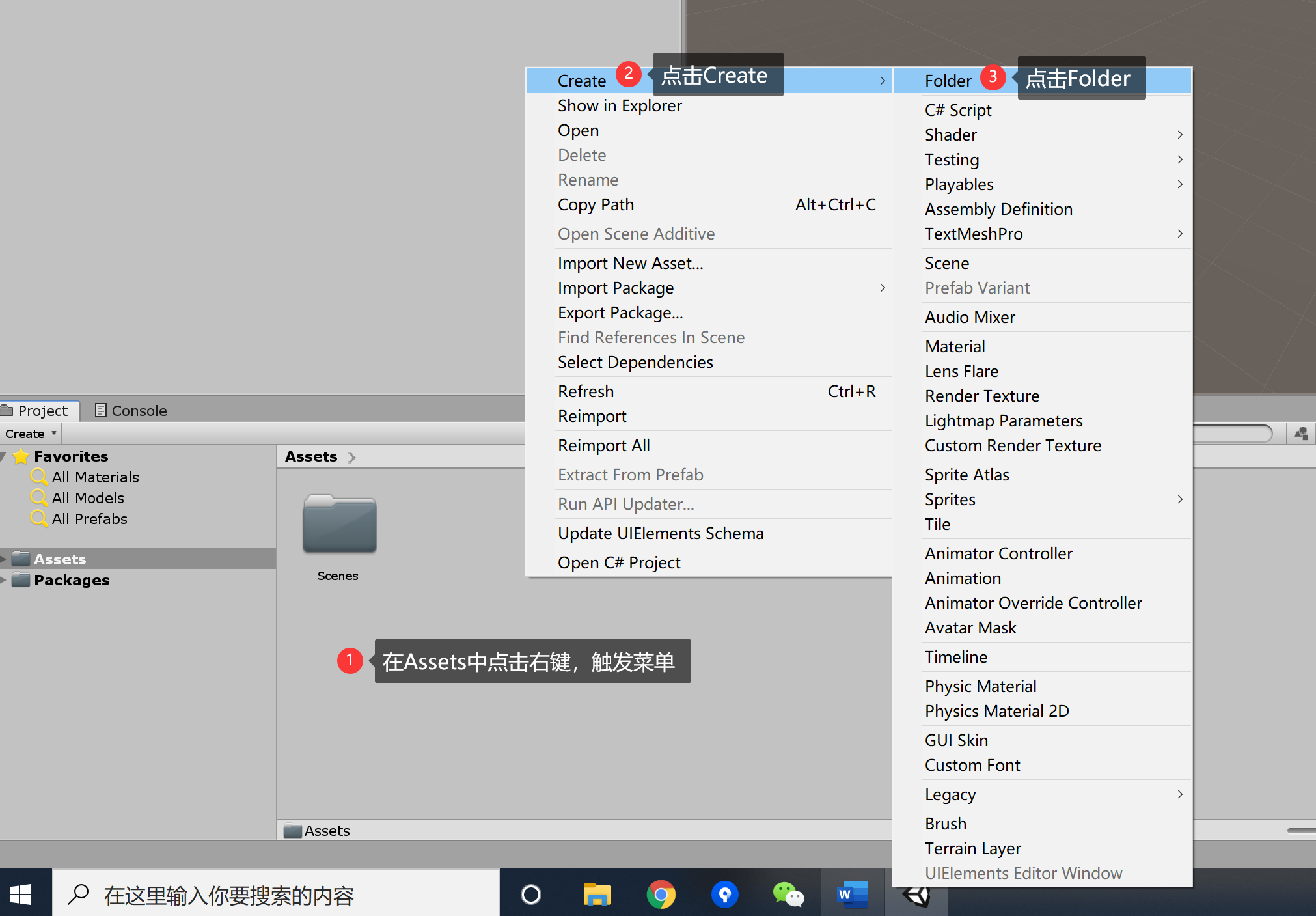
Unity中，所有的项目资源文件都放在Assets文件夹下面，为了便于管理项目资源文件，我们一定要对项目资源进行合理的分类，这一点很重要，尤其是后期项目资源文件多起来的时候，合理的分类能帮你更好的管理资源，所以我们首先创建一个名为Scenes的文件夹，来存放项目场景文件。

创建文件夹（其他一些素材或文件的创建同样）的方法有2种：

1. 通过Create按钮创建，我们在Project窗口中，选中Asset文件夹，然后点击Create弹出下拉菜单，在菜单中选中Folder，如图3所示：



1. *创建文件夹方法其一*
2. 在Assets中通过右键弹出创建菜单，如图4所示：



1. *创建文件夹方法其二*

然后，我们将我们当前的场景保存下来，通过File->Save Scene来保存当前游戏场景，你也可以通过快捷键【Cmd + S】来保存。（PS：windows用户快捷键为【Ctrl + S】）。给Scene命名和制定存放路径，这里我们将场景文件保存在Scenes文件夹下面，我们要从一开始就养成良好的命名习惯。

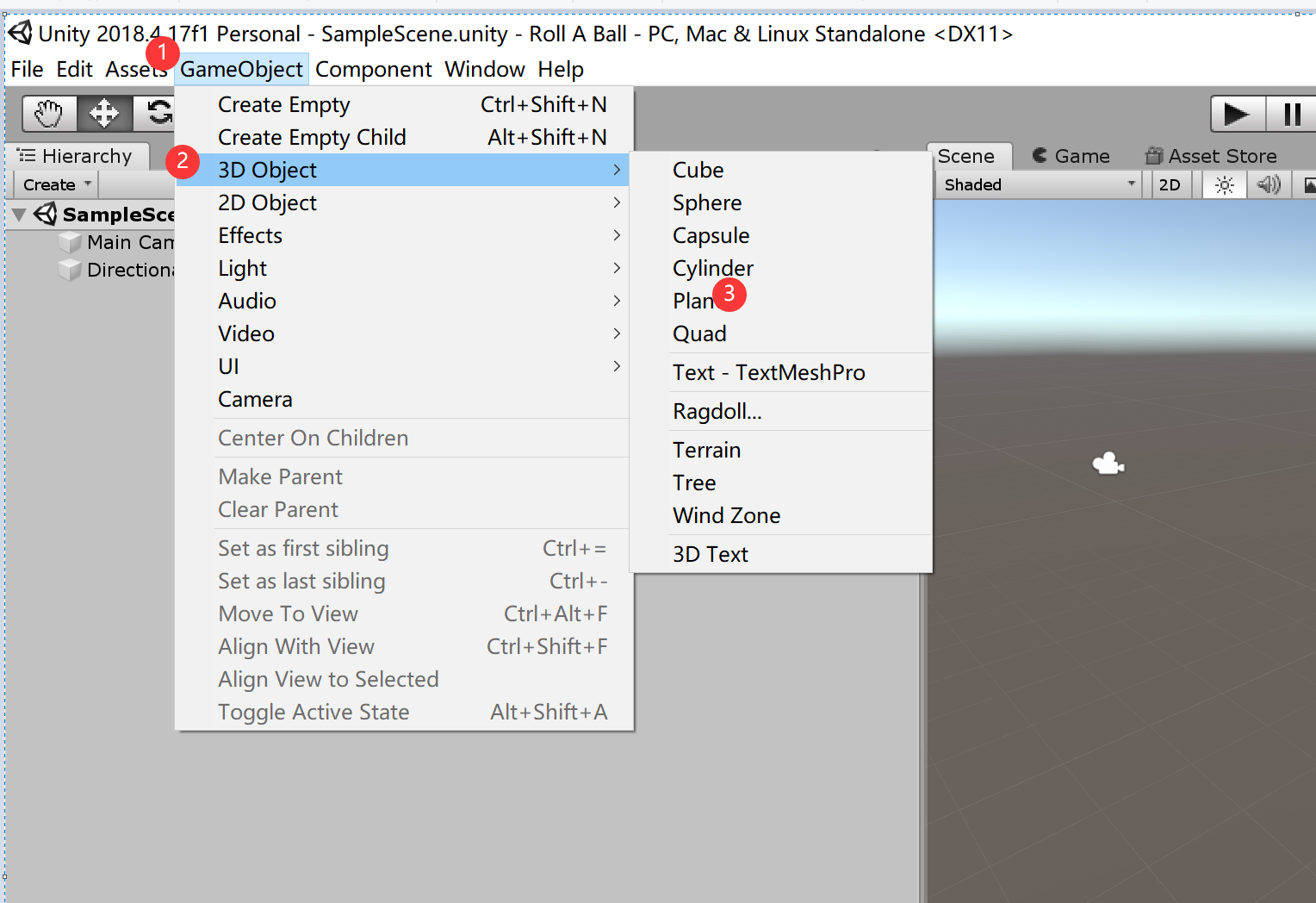
### 2 添加基本物体（此部分10分）

#### 2.1 添加一个平面

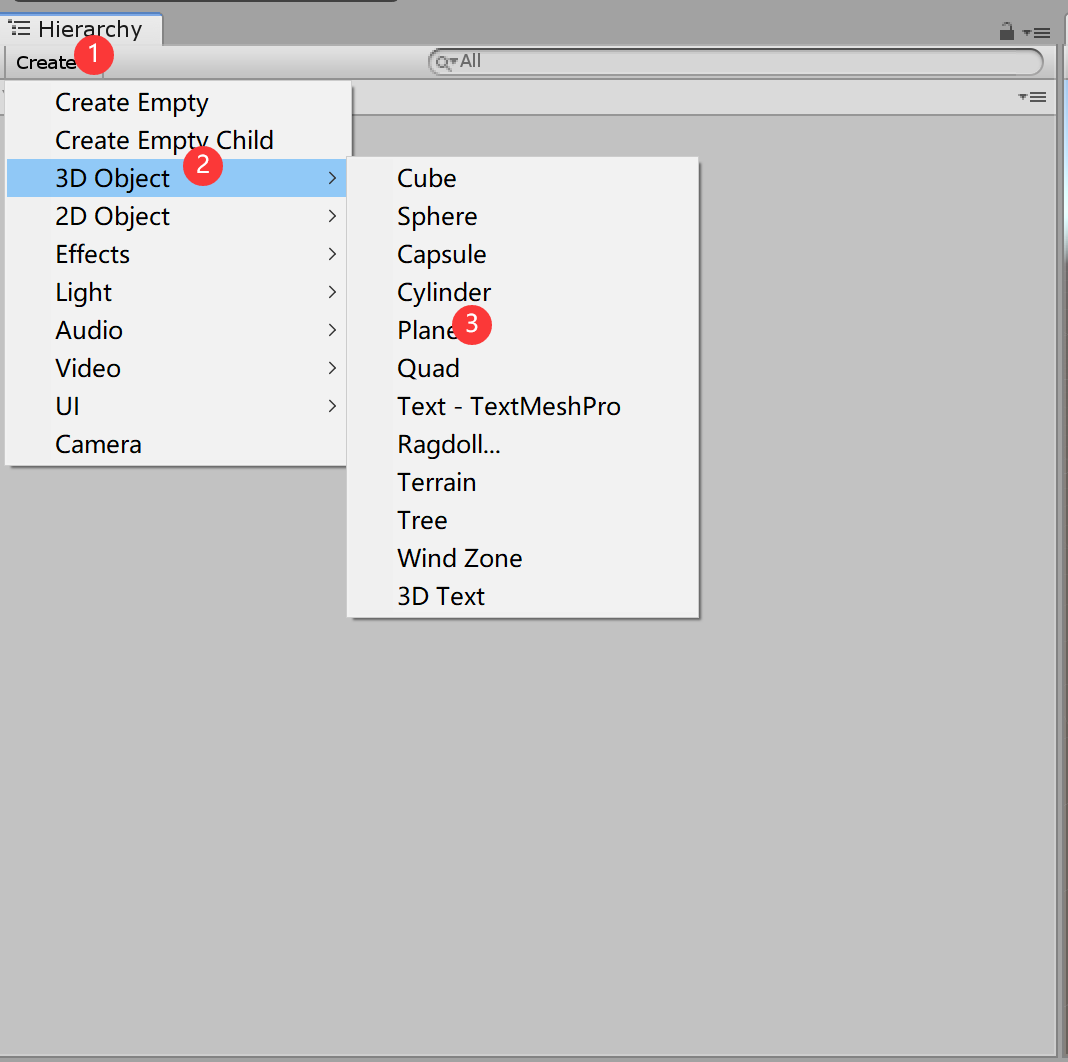
首先，我们要做一个什么样的项目？在本教程中，我们希望实现一个简单的游戏，来教会大家一些Unity的基本操作。在这个游戏中，玩家控制一个小球，通过上下左右的移动来触碰方块，触碰完所有的方块，即宣告游戏结束。这个游戏看似简单，实则涵盖了不少Unity的核心功能，接下来，我们将一步步地完成这个游戏。在这一章中，我们将先从物体的建造开始。

保存了游戏场景后，我们就可以开始在场景中添加一些我们所需要的游戏对象了，我们第一个需要添加的就是进行的平台，这里我们使用的是unity自带的Plane对象，添加一个游戏对象（GameObject）有3种方法，与前面的文件夹创建相似：

1. 通过菜单GameObject->3D Object->Plane来创建：

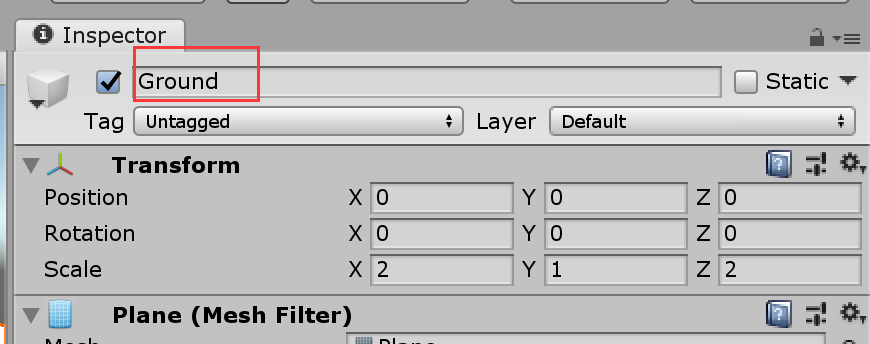


1. *添加平面*
2. 通过Hierarchy窗口中Create按钮来创建；



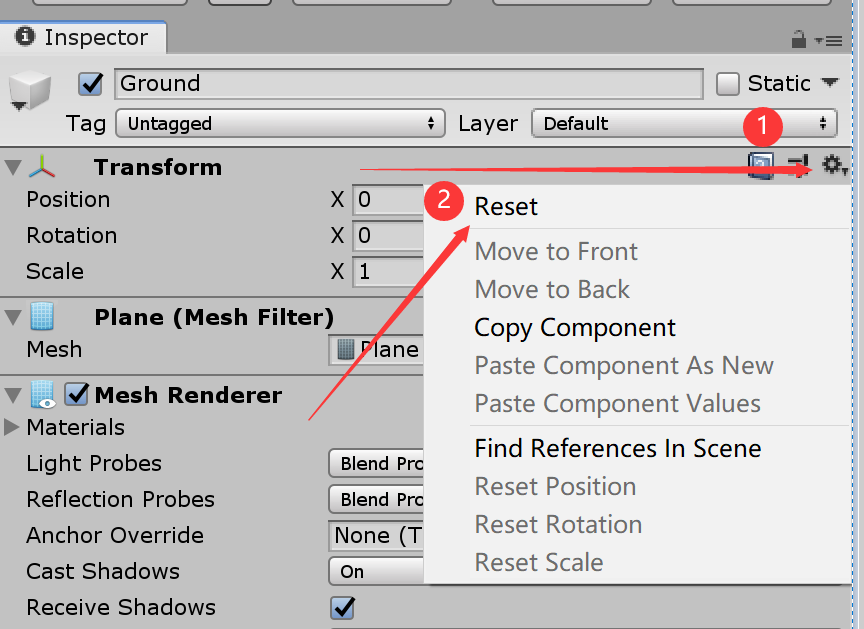
1. *添加平面*
2. 在Hierarchy窗口中单击右键会弹出菜单，选择3D Object->Plane来创建。

平面对象创建成功后，我们最好给它起一个好的名字，来解释它的作用。在Inspector窗口中最上方修改其名字。我们将平台的名称修改为Ground，来表示这是一个地面。



1. *修改名称*

名字修改完成后，我们还需要做一个很重要的事情（官方强力推荐），我们把平面的Transform属性给重置（reset）一下，我们在Inspector窗口中找到Transform组件，然后点击它右上角的小齿轮，选择reset选项，如图8：



1. *重置Transform组件*

然后我们调整以下Ground的大小。在本例中，我们把平台的Scale X和Z的数值设置为2。

**提示**：在Scene窗口中，我们可以直观地观看到我们的修改结果对某个个体，乃至整个环境产生的影响。这里有3个很实用的快捷键可以使用：

1. W：切换到移动模式，该模式下可以任意拖动物体，改变它的位置
2. E：切换到旋转模式，该模式下可以任意旋转平台，改变它的角度
3. R：切换到缩放模式，该模式下可以任意缩放平台，改变它的大小

当然，最精准的方法是输入数值，我们可以修改Transform组件的对应数值（Position：位置，Rotation：旋转，Scale：缩放）来修改物体，例如上面就是通过修改Scale的X与Z将平面拉长至开始的两倍。

#### 2.2 添加球体

有了平面后，我们还需要一个球体来作为我们的主控制物体。添加球体的方式，和上面添加平面的方式一样，只不过我们选择的是Sphere对象。这里不加赘述。同样，在生成后，我们将球体的Transform属性重置，然后给它起一个名字。这里我们将其命名为Player，因为在这个游戏里面，这个球就代表玩家，我们希望通过方向键来控制球体移动，当然这要在后续的脚本编写中去实现。

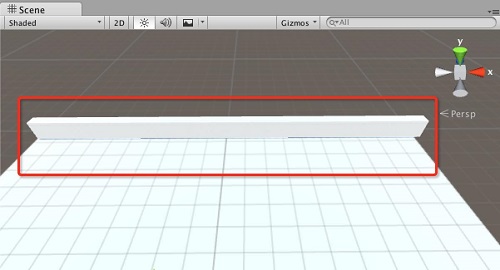
将视野定位到球体后，会发现此时球有一个半是陷入在平台中的，这时我们通过调节球的Transform中的Position的Y值为0.5，让它正好处于平台上（初始球的半径为0.5）。

**提示**：先在Hierarchy中选中一个物体，然后将**鼠标移至Scene中**，点击F快捷键。Scene窗口中的摄像机就会对准到我们选中的物体上。这是非常便捷且常用的快捷键功能，特别是在场景非常复杂的情况下。

#### 2.3 添加墙和目标方块

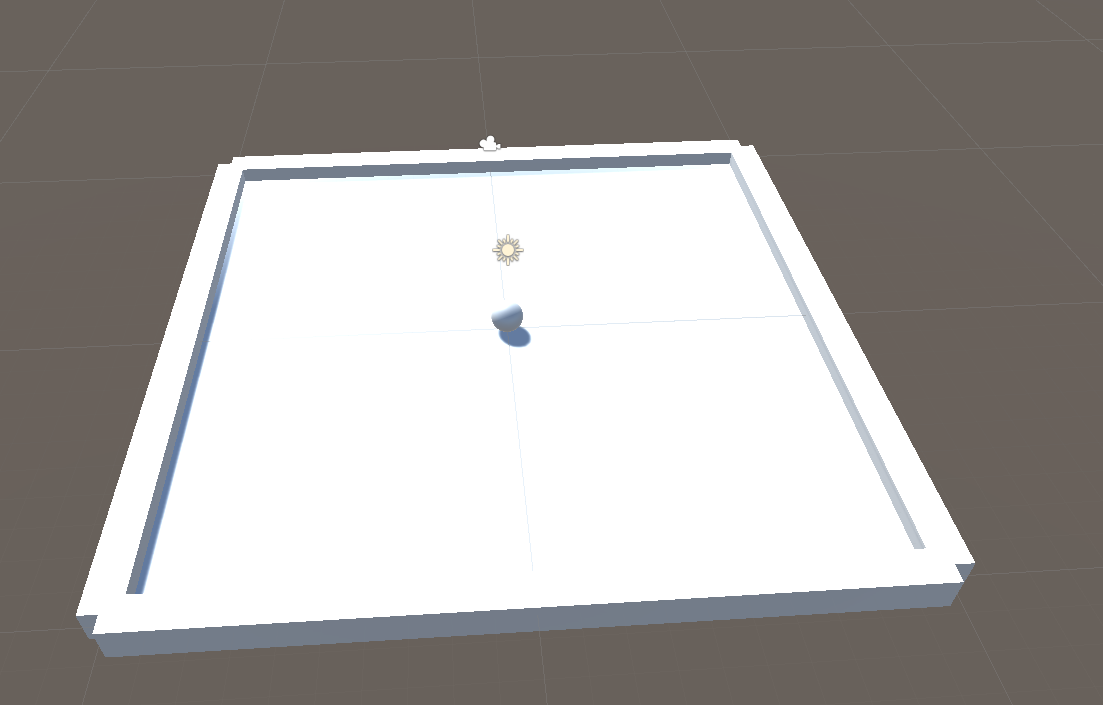
由于我们的球是要移动的，且之后我们将对它运用重力，那么就带来了一个问题，球体如果移动超过平面的边界，就会掉下去，这不太友好，所以我们接下来给平台加上一圈围墙，来防止球体跑出平面。

与前面相类似，我们新建一个cube对象，然后reset他的Transform属性，重命名为Wall，将其长度设置成和平面一样，然后将其的位置移动到平面北面的边缘，如图9：



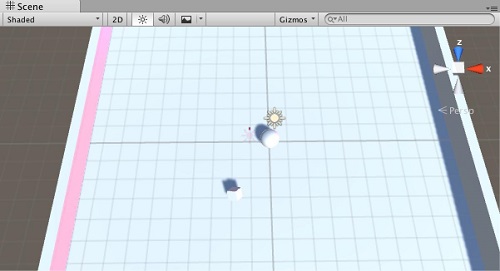
1. *将新建Cube拉伸成一堵围墙*

然后同样的，我们给平台的南面，西面，东面都制作一堵墙，以包围整个游戏空间。



1. 四面墙

有了平面，有了球体，接下来我们就需要一个方块，这个方块可以让球体碰撞并拾取。首先我们创建一个cube对象（不要忘记reset），然后我们将cube命名为PickUp，然后将其拖动到一个你觉得合适的位置，再把它的Rotation的X、Y、Z值都设置为45，比如这样：



1. *加入游戏目标——待拾取方块*

到目前为止，我们已经搭建了我们的场景（平面与围墙），主控对象（球），游戏目标（方块）。当然，你可能觉得方块的数量不够多，想多复制几个，先别着急，我们将在第三章绑定完脚本后，再使用Unity特有的预制体来快速复制方块，这里我们建造一个方块足以。

### 3 使用脚本来让场景动起来（此部分共70分）

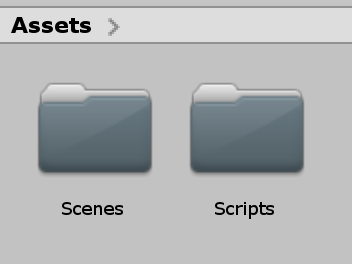
现在，我们的场景已经搭建完毕，不过令人遗憾的是，所有东西都是静止的。别急，在本章中，我们将介绍如何通过脚本来构筑“世界的法则”。

#### 3.1 如何建立一个脚本

还记得创建文件夹的方法吗（见图3与图4）？同样使用这种方法，我们可以看到紧挨着条目Folder下面的两栏分别是C# Script和Javascript，这边是脚本文件。由于使用C#脚本的人数远超于js脚本（甚至官方也以C#脚本进行示例教学），因而这里我们着重学习C#脚本，所以这里我们点击C# Script选项，然后对新建的脚本进行命名。

**提示**：在脚本命名时需要慎重。如果你细心观察，你会发现，在你命名完成后，脚本内也自动生成一段代码。这段代码声明了一个空的类，而这个类的名字与你脚本名称相同。这里就涉及了一条编译规则，与Java相同，你的类的名称必需与你的脚本名一致（当然也有例外，这里不做讨论）。因而在后期，如果你需要更改你的脚本名，你需要同时更改你的脚本名与类名。

这里，我们要再提一下开始说到的资源管理思想，为了便于我们管理游戏的资源，所以我们会创建一个Scripts文件夹，来专门存放所有的脚本文件，如图12所示：



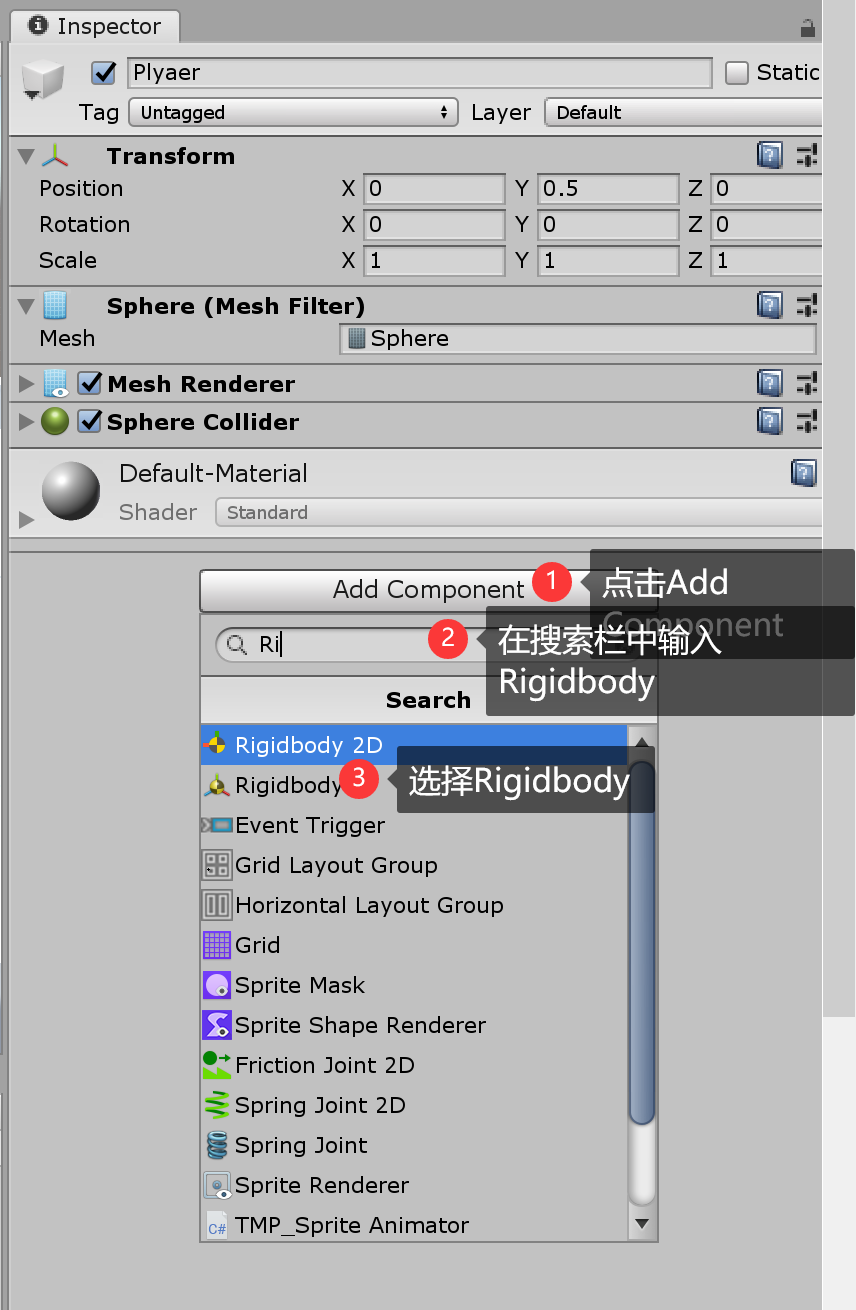
1. *新的文件夹Scripts*

#### 3.2 控制小球的移动（15分）

为了让小球能依照较为真实的物理进行移动，我们首先要为小球添加刚体属性脚本。这里其实有一个关于脚本的概念，还记得之前在Inspector上看到的一栏一栏的像Transform一样的组件吗？其实这些组件都是脚本，只不过这些脚本是Unity官方已经写好的脚本，你不能也不必要对它进行修改。不过官方脚本再多，也难以满足各种项目的千奇百怪的需求，因而，我们也可以像3.1一样自定义脚本，并实现相应的功能，以满足我们特定项目中的需求。

言归正传，我们要添加的刚体属性脚本是较为常用的Unity官方脚本。绑定刚体的物体可以参与到整个Unity物理引擎的模拟中，包括但不限于与其他带有碰撞体（Collider）的物体进行碰撞，添加重力，对物体施加力、冲量等。

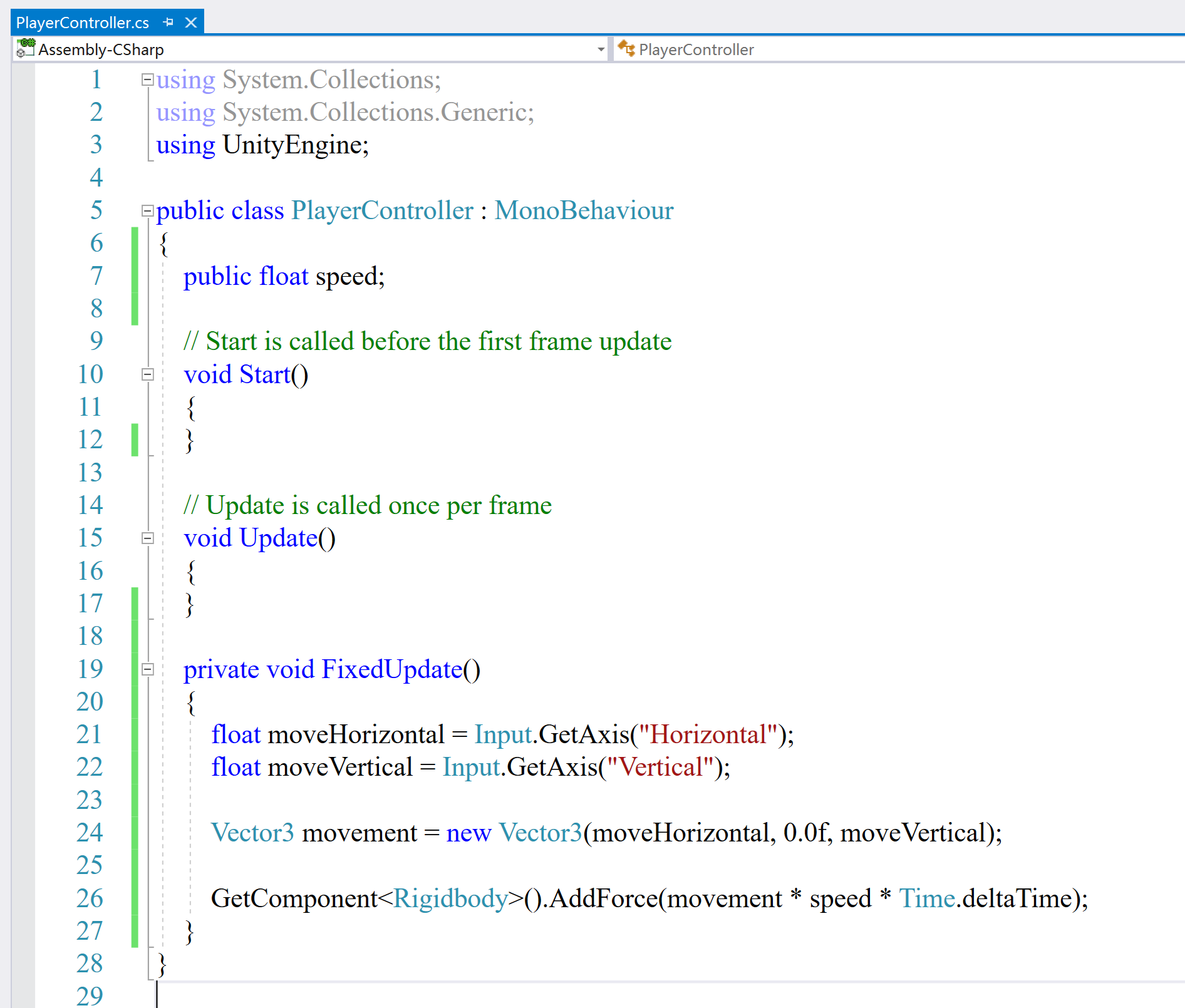
添加方式很简单，选中球体，在Inspector上点击最后的Add Component按钮，然后输入Rigidbody即可找到.（注意是Rigidbody，不是Rigidbody 2D）如图13所示：



1. *加入官方自带的刚体（Rigidbody）脚本*

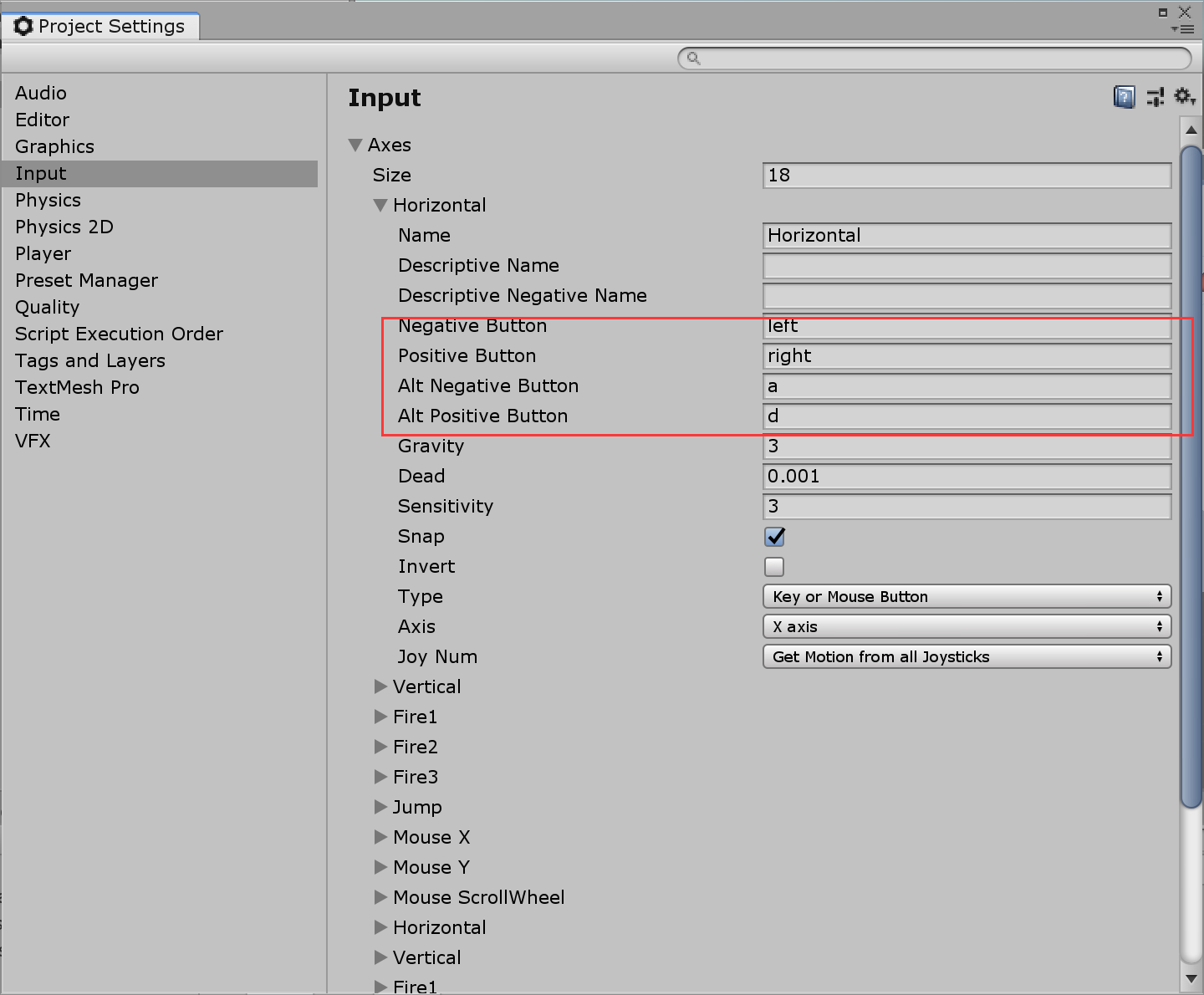
这时我们的球可以与其它物体碰撞了，准备工作算是完成，下面我们将通过自定义脚本的方式控制球的移动了。

我们先根据3.1提到的方法创建脚本，命名为PlayerController，打开该脚本，并遵照图14输入代码内容。



1. *PlayerController脚本内容*

这里针对代码做出解释：Time.deltaTime代表的是帧与帧的间隔时间；Input为Unity的输入组件，以获取所有设备的输入信息（鼠标，键盘，手柄等各种输入设备），通过菜单Edit->Project Settings->Input可以找到相应的信息；GetAxis()为获取轴信息的函数。我们打开Input，找到并展开Horizontal词条，可以看到如图15中的内容：



1. *Input组件示意图*

这里我们看到Horizontal词条对应的输入为left/right键按下的程度（left/right即键盘左右键），或是AD键按下的程度。因而这里第一句GetAxis其实就是获取上述按键按下的程度（这里全左为-1，全右为1）。同理第二句GetAxis其实就是获取上下按键或WS按键按下的程度（这里全上为1，全下为-1）

然后定义一个三维的变量，将水平的轴数值（左右键/AD键）赋给x轴，竖直的轴数值（上下键/WS键）赋给z轴，y轴不变（y代表高度，球只在平面上滚动）。然后使用刚体组件下的AddForce函数为球施加力度，力度大小与定义的三维变量成正比。

然后，我们把PlayerController脚本绑定在球上，这里可以像搜索Rigidbody一样搜索到我们的新的PlayerController脚本。也可以直接将PlayerController本体直接拖拽到Hierarchy对应的小球下或小球对应的Inspector上。

这里注意代码中还有一个speed变量，指代的是变化速率，我们没有在代码中初始化这个变量。不过由于这个值为公有变量（public），因而在绑定脚本后，我们可以在Inspector上直接看到它（如图16）。因此我们可以直接通过Inspector设置其初始值，方便调试，这里设置其为500。



1. *在Inspector上的PlayerController脚本区域出现speed变量*

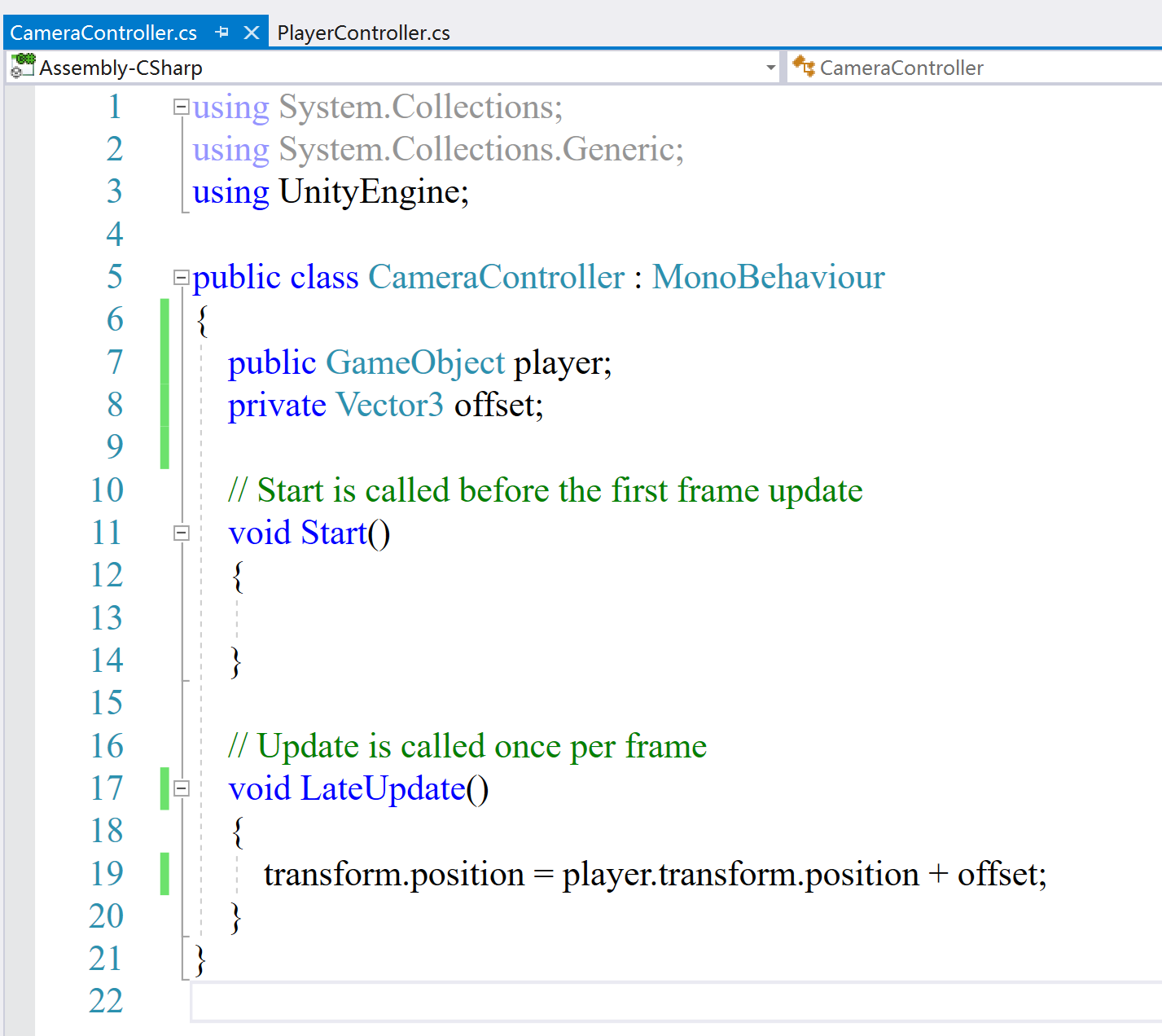
到目前位置，这部分已经完成了，点击播放键，尝试使用WSAD或上下左右键控制小球的移动。

#### 3.3 让摄像头跟随球体移动（15分）

在移动球时，我们发现球很容易移出我们的视野，这是因为摄像机始终固定在原处。所以，我们希望编写一个脚本，来实时更新摄像机的位置，以跟随球的移动。

首先，我们先拖动摄像机的位置和角度，来处于一个合适的视角，本例中，我们将摄像头（Hierarchy中的Main Camera）的Position的Y和Z分别调至5和-6，然后把其的Rotation的X值调至45，这样摄像机的视角就变成了45°俯视角。

然后我们新建一个名为CameraController的脚本，并写入如图17的代码：



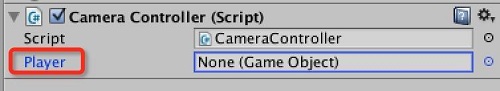
1. *CameraController脚本代码*

我们一句句来解读脚本的含义：

1. 首先，我们定义了一个public的GameObject变量，这个变量就是用来告诉程序，摄像机会跟随哪个GameObject来移动。还记得之前我们提到的public的float变量speed，这个player也会像speed一样出现在Inspector上，以便初始化。
2. 然后我们定义一个private的Vector3的变量，这个变量我们主要用来记录摄像机初始的位置，用作之后更新摄像机的位置的偏移向量。这里我们不需要手动初始化它，也就不需要让它显示在Inspector上，所以这里定义为private私有变量。
3. 【需要自己编写这部分代码】然后我们在Start方法中，将摄像机的初始position数值赋给offset变量，意思是在开始前，用offset变量记录下摄像机的初始位置。
4. 最后我们在LateUpdate()方法中，把摄像机的位置实时变更为球体的位置加上最开始的偏移位置。这样就可以保证摄像机实时地处在相对于球的一个固定的偏移位置上了。

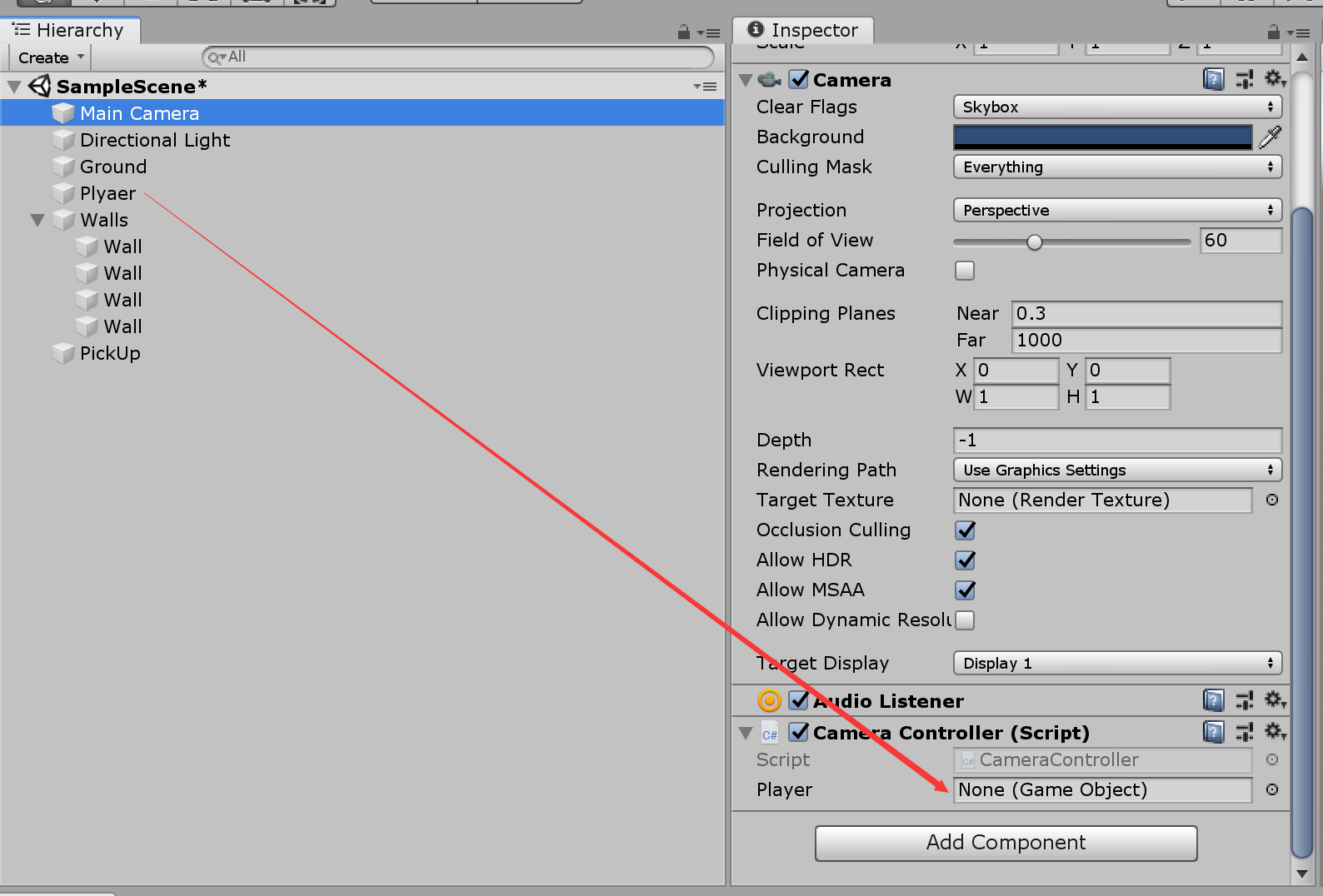
这里其实脚本中使用Update()方法一样可以达到效果，但是我们使用LateUpdate()（最后更新）效率会更高一点。

写完脚本内容后，我们保存一下脚本，把脚本绑定在Main camera上，然后回到Unity编辑器界面。与前面说的speed相同，我们会发现摄像机的脚本组件中，多了一个Player属性，如图18：



1. *公有GameObject类型变量Player*

这就是我们刚刚在脚本中定义的public类型的属性，然后我们把球体拖入到Player属性中，如图19所示：



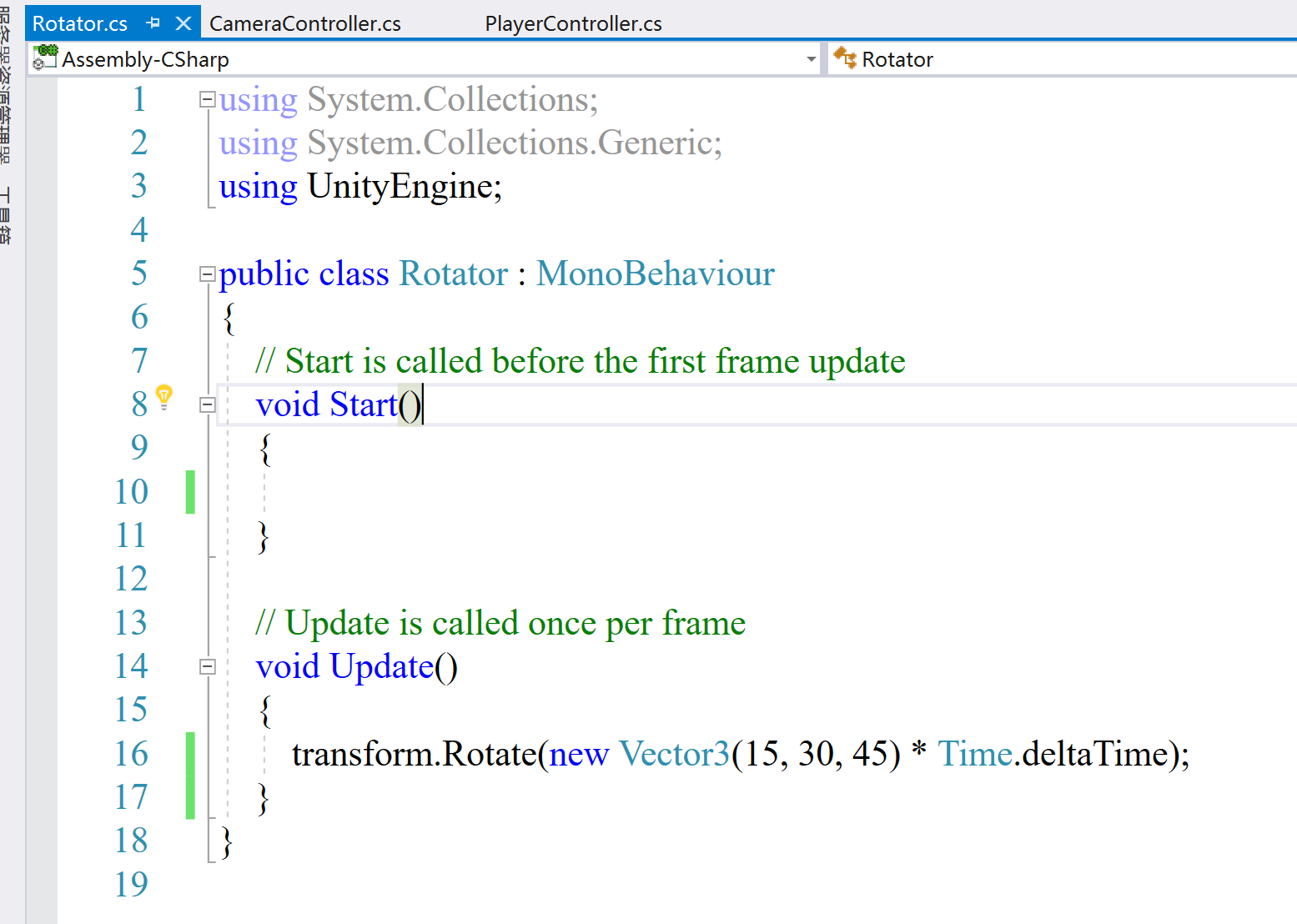
1. *拖拽小球Player至CameraController的Player属性上*

接下来我们运行一下游戏，就可以发现摄像机现在可以正常跟随球体移动了。

#### 3.4 让目标方块旋转起来（15+10分，脚本部分15分，prefab部分10分）

在游戏中，目标方块一动不动似乎有些令人乏味，于是我们希望通过一个脚本让方块旋转起来。

创建一个名为Rotator的脚本，然后按照图20所示输入代码：



1. *Rotator脚本代码*

里面我们就添加了一句话，就是红线标记的，这里new Vector3 (15, 30, 45) 是一种欧拉角（Euler）的表示方式，在Unity中相当于将一个物体先沿y轴旋转30度，再沿x轴旋转15度，最后沿z轴旋转45度。Time.deltaTime代表的是帧与帧的间隔时间。将两者相乘，代表以[(15, 30, 45) \* Time.deltaTime/帧]的速率自动旋转物体，等同于以[(15, 30, 45) /s]的速率自动旋转物体。

保存脚本，然后把Rotator脚本绑定在方块上，运行一下，就可以看到方块自己旋转起来了。

当然，之前我们也说了，为了让游戏丰富起来，我们只有一个方块是不够的，因此我们需要多创建几个方块，在这之前呢，我们需要先制作一个Prefab（预制件），制作一个预制件有以下好处：

1. 一次制作，重复使用：可以在任何项目中使用Prefab；
2. 一次修改，全部变化：我们修改Prefab的属性后，所有使用该Prefab的对象属性都会跟着发生变化。

而制作Prefab的方法也非常简单，我们为了资源管理方便，先在Assets文件夹下面创建一个Prefabs文件夹来存放Prefab文件，然后拖动Pickup对象到Prefabs中，一个Prefab就创建好了。

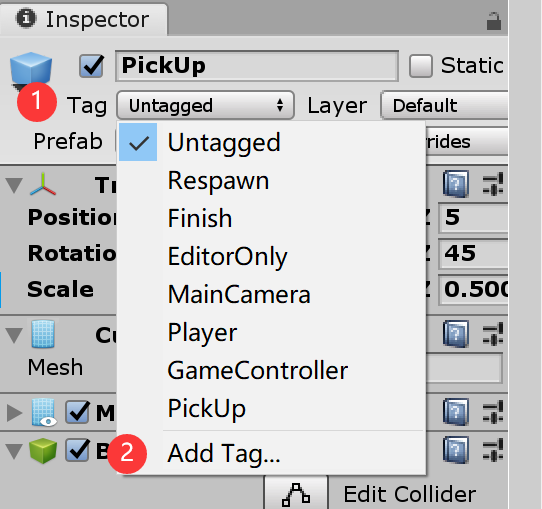
然后我们可以通过Prefab创建新的PickUp对象，直接拖拽Prefab到场景中合适的位置（可以通过Transform设置其位置），我们会看到，除了我们修改的位置信息，这些PickUp方块的其他属性完全一致，包括其绑定的脚本。这里我们生成总共8个PickUp（含原来的方块）。

#### 3.5 拾取方块（15分）

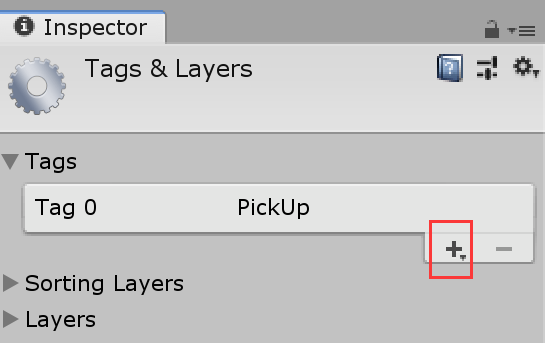
接下来我们实现球体拾取方块的功能，我们通过查API得知有一个OnTriggerEnter的方法可以用来在球体碰到物体的时候调用，光有这个方法还不够，我们还需判断球体碰到的是什么东西，我们不能碰到任何东西都拾取，所以我们需要给拾取的方块增加一个标记（Tag），只有碰到特定标记的物体（即方块）的时候，我们才去拾取它。

因为我们场景里面一共有8个方块，如果一个个的设置标记，就有点太麻烦了，还记得上面我们提到的Prefab的第2点好处吗？对的，我们只需要给Prefab设置标记就可以了。

首先我们选中PickUp的Prefab，然后在Inspector的Tag里面新增一个Tag，我们命名为PickUp。



1. *选择添加标签的选项*

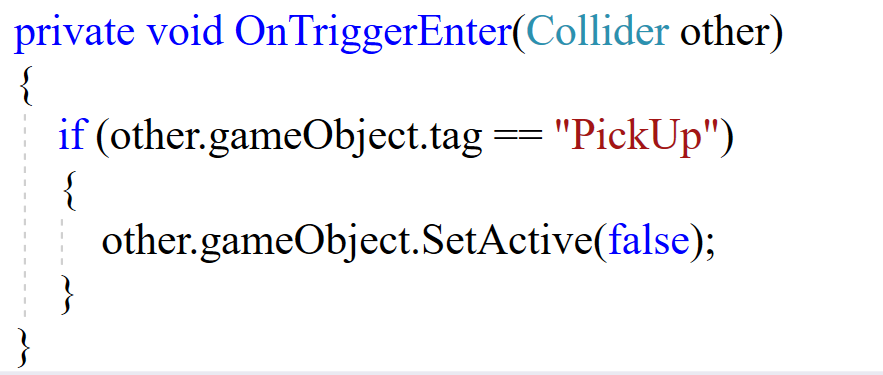


1. *添加Pick标签*

当然，这只是添加了一个PickUp标签，再次点击Prefab的Tag，我们就能看到新添加的标签PickUp了，选择它。

然后，我们的准备工作并没有结束，OnTriggerEnter只对特殊的Trigger碰撞体起作用。所以这里我们需要选中Prafab，找到它的BoxColiider脚本，将它下面的is Trigger属性勾选上。

到此准备工作就绪，打开之前写的PlayerController脚本，新增图23的代码：



1. *PlayerController脚本新增代码*

现在我们来解释以下这段代码。首先OnTriggerEnter是MonoBehavior（到目前为止，我们所有的脚本都默认继承于这个类）的自带函数。该函数会在绑定的物体触碰到一个Triggle碰撞器时（在这里是我们的方块）被自动调用，并返回一个Collider类型变量来指明碰撞的信息。

然后根据Collider返回的碰撞信息，我们判断碰撞物体的Tag是否为PickUp，如果是，则说明我们真的碰撞到了方块，那么我们就让方块消失（代表已经拾取了该方块）。这里SetActive(false)代表让方块不显示，并不代表删除方块，你可以在任意时刻通过调用SetActive(true)让它重新显示。

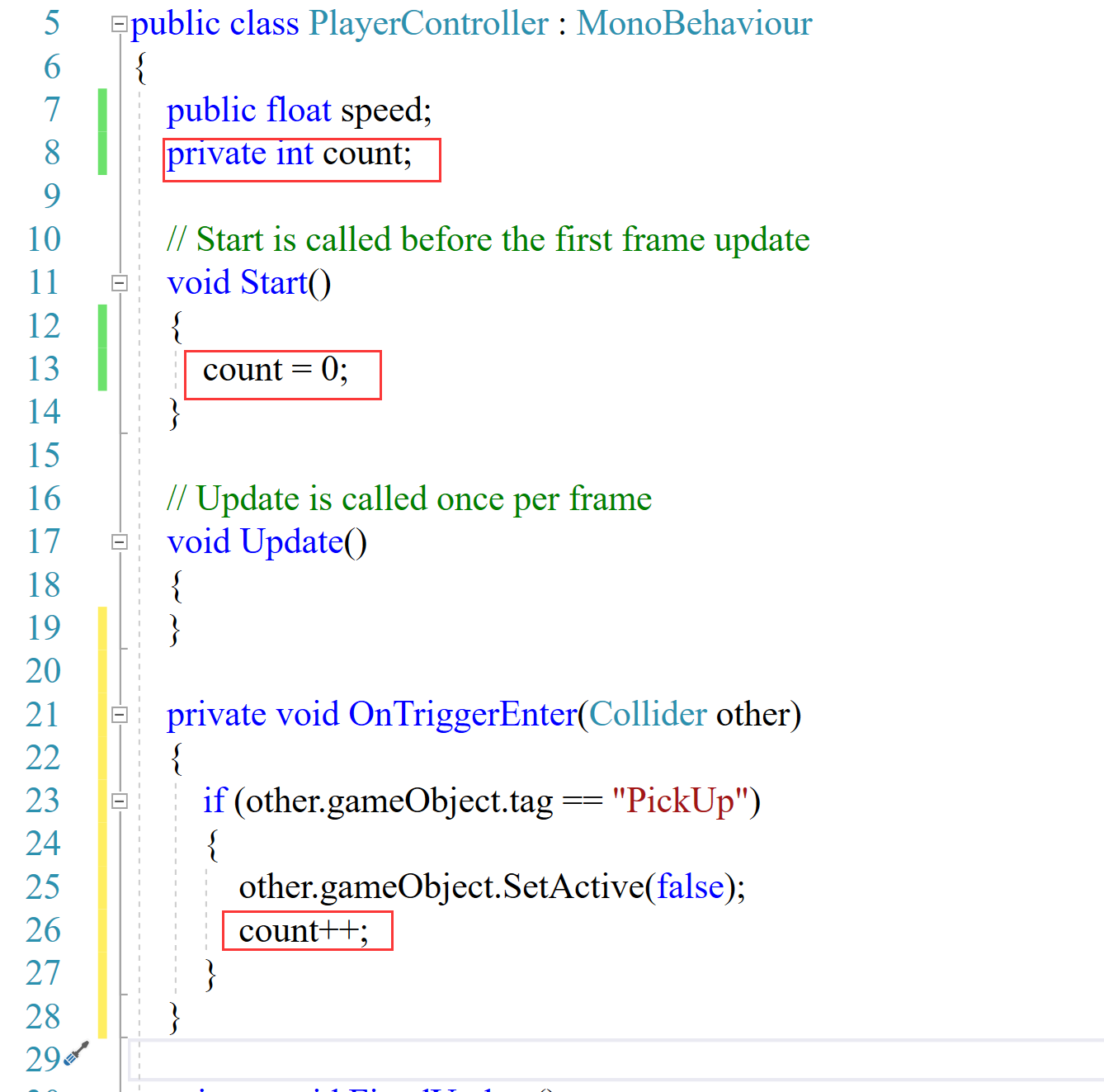
运行程序，控制小球去触碰方块，方块就会消失，至此核心功能基本完成。

### 4 制作一个计分表（15分）

至此，我们的游戏基本可以玩了，但是我们还需给玩家增加一点刺激，做一个计分板来鼓励玩家多拾取方块，那么实现一个计分板我们需要以下元素：

1. 一个用于保存玩家的分数的变量
2. 一个用于显示分数的UI

首先，我们修改PlayerController脚本，如图24所示：

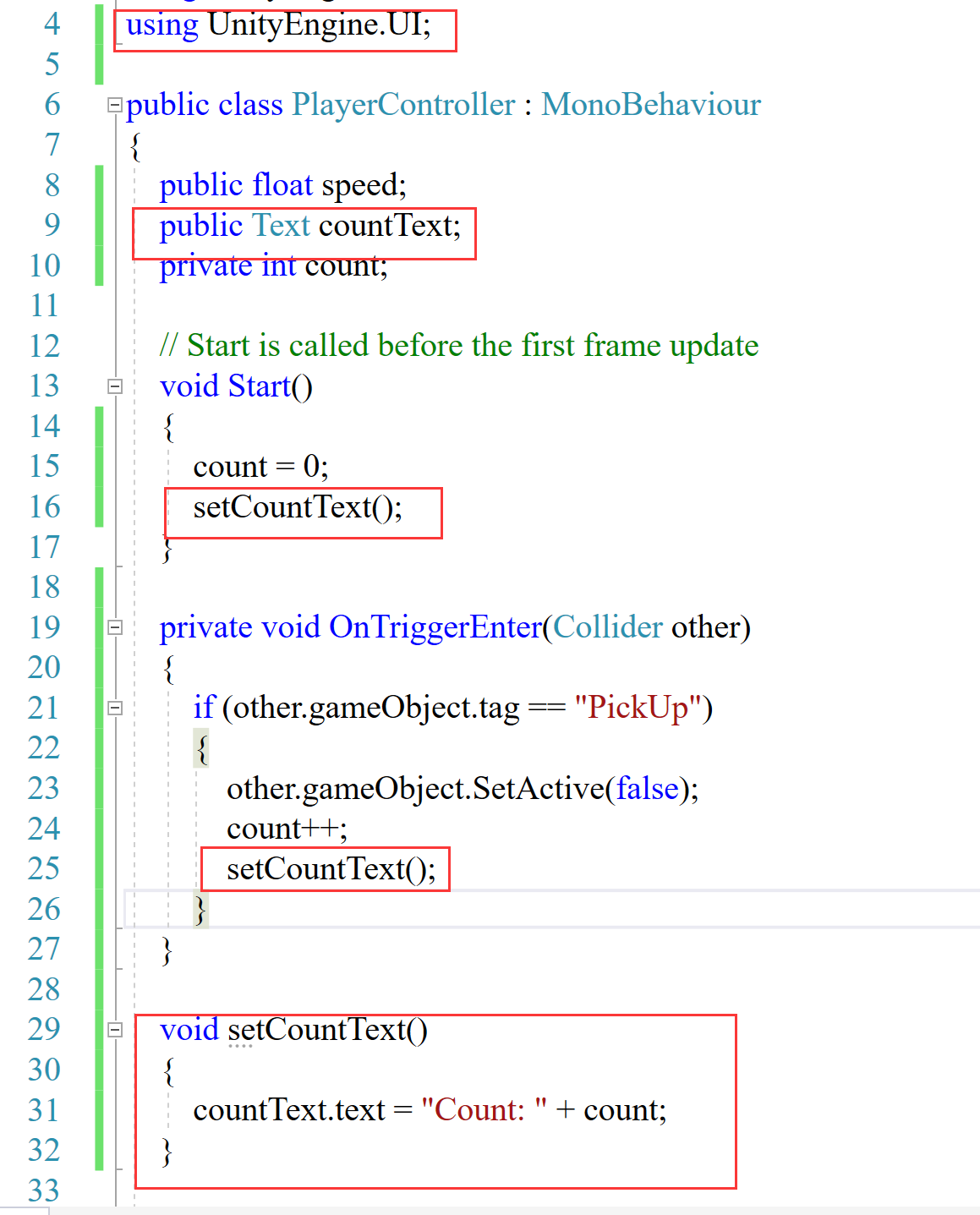


1. *在PlayerController中加入分数*

这里加入了新的分数变量count，它在Start中初始化为0，在拾起方块时（OnTriggerEnter中）数量加1。

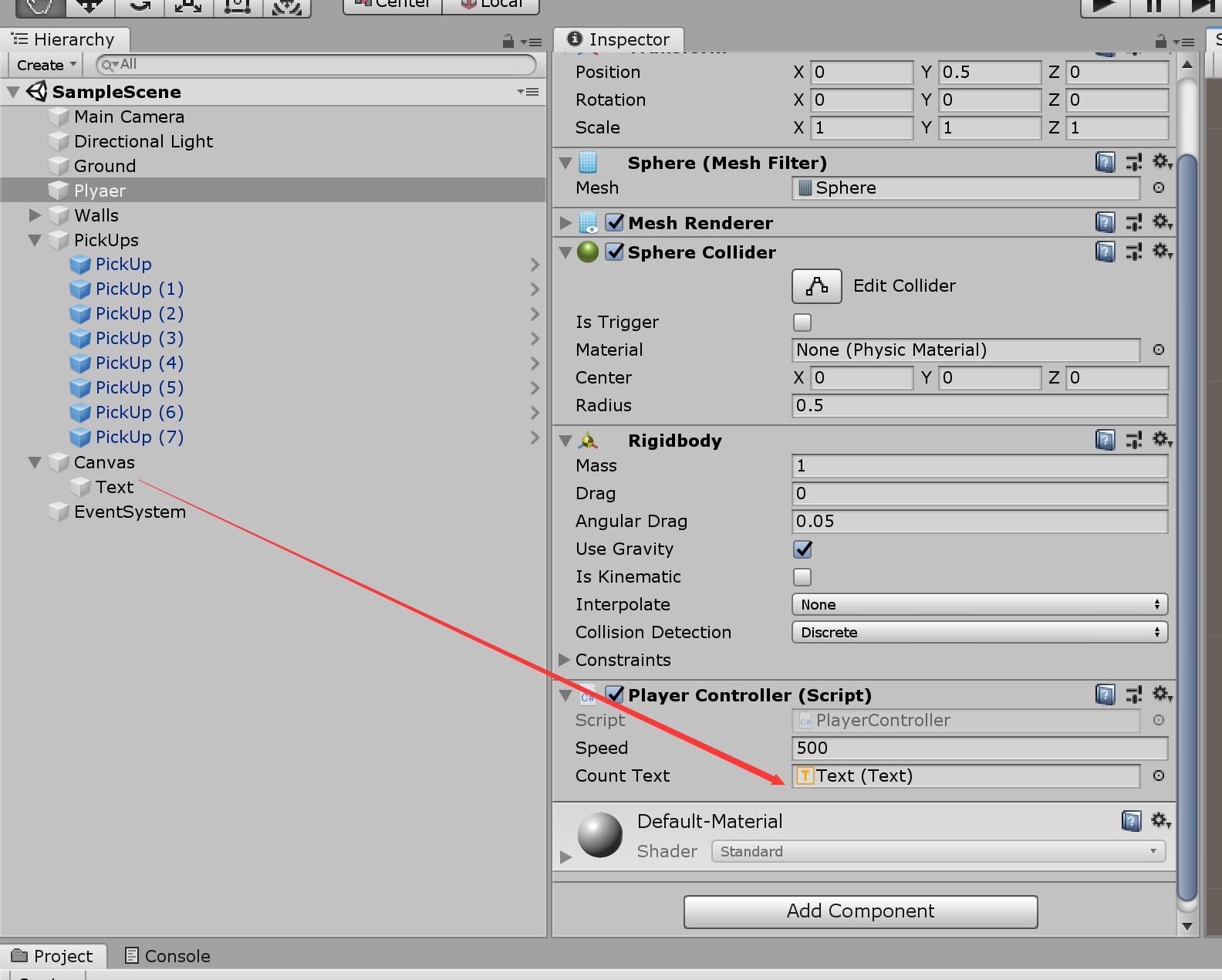
然后，我们需要一个UI来展示分数。通过GameObject->UI->Text，可以来创建文本显示的UI，调整Text至屏幕的左上角。

有了UI，我们将再次修改PlayerController脚本，如图25所示：



1. *在PlayerController中显示需要分数*

在代码最前面，我们需要引入UnityEngine.UI，只有这样后面代码才会识别Text类型的变量，然后我们在定义countText变量来保存UI Text。接着我们在Start方法中调用setCountText方法，来更新countText的内容，然后我们在每次球体碰到方块的时候，也需要更新countText的内容。最后是我们定义的setCountText方法，然后我们保存代码，回到Unity编辑器里面，仿照之前拖入小球到CameraController的操作，将Text拖入到PlayerController里面的Count Text属性。



1. *将Text UI拖入PlayerController的Count Text中*

做完这个，运行游戏，尝试拾取几个物体，我们就可以看到UI的数值发生了变化。

### 5 发布游戏

游戏到这里我们就告一段落了，接来下我们可以尝试将游戏发布到网页，让你的朋友试玩你的游戏，Unity发布游戏非常简单

第1步：首先选择File->Build Settings

第2步：我们选择PC，Mac & Linux Standalone平台，然后点击Switch Platform

第3步：我们点击Add Current按钮，将当前游戏场景加入进去

第4步：点击Build按钮，然后选择保存路径，就可以完成发布了

最后我们找到保存路径下生成的文件，点击后缀为exe的文件，就可以运行游戏了

## 实验评分

实验得分是根据最终提交的exe/unitypackage中包含内容来评价的。

1. 创建Unity项目（5分）
2. 添加平面、球体和方块（10分）
3. 建立控制小球移动的脚本，小球受键盘控制移动。（15分）
4. 建立控制摄像头的脚本，摄像头随小球移动而移动。（15分）
5. 建立控制目标方块旋转的脚本，方块能够旋转。（15分）
6. 创建目标方块的prefab并复制7份。（10分）
7. 编写拾取方块的脚本部分，目标方块与小球碰撞后消失。（15分）
8. 编写积分表部分脚本，UI中的分数能够随着消失的目标方块个数增加。（15分）