

Unity 3D游戏编程自学#3——Unity 3D初步

原创 Zerg_Wang 于 2019-02-19 18:53:22 发布 617 收藏 4

编辑 版权

分类专栏： Game Programming



Game Programming 专栏收录该内容

0 订阅 7 篇文章

1.开始

在创建的项目文件夹中，各个子文件夹的作用：

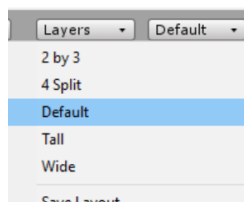
Assets：保存游戏所需资源。

Library：保存当前项目运行所需要的库。

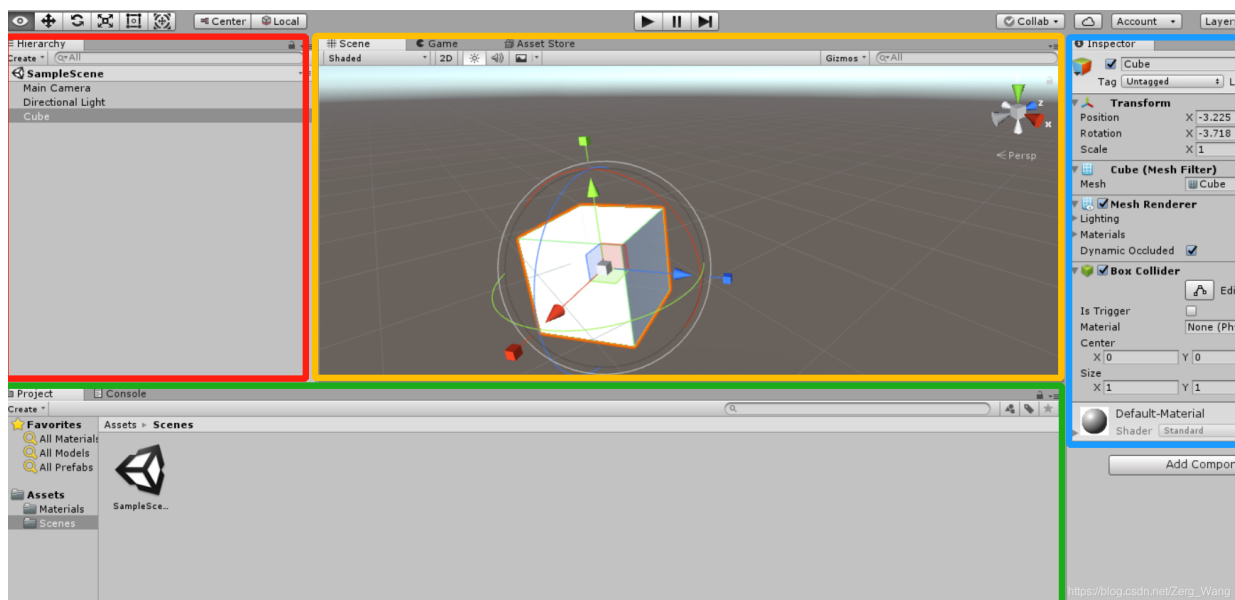
ProjectSettings：保存项目设置信息。

Temp：保存临时数据。

面板布局：设置为Default。



软件界面介绍：



左上是层级（Hierarchy）面板，用于显示当前场景有哪些资源，可在此为场景添加资源。

上面这个是场景（scene）面板，用于预览当前场景。

下面这个是项目（project）面板，用于管理项目资源。

右侧这个是属性（Inspector）面板，用于查看、编辑物体和项目资源的具体信息。

场景（scene）是Unity 3D中的一个基本概念，是一种资源，保存于Assets文件夹中。Ctrl+N可新建场景，Ctrl+S可保存当前场景。

每个新建场景时都会自带两个物体：摄像机和光源。

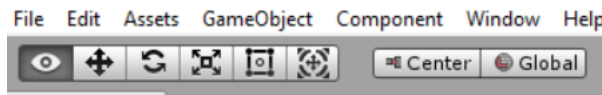
2.观察

在场景面板的右上角有一个三位坐标轴，其中红色的X轴代表右方，绿色的Y轴代表上方，蓝色的Z轴代表前方，可称其为世界坐标轴。点击相应坐标轴可切换到该轴的视角去观察。

坐标轴下方为显示模式：Persp或者Iso（点一下可切换）。Persp为透视模式，物体显示近大远小，Iso为正交模式，远近物体一样大。

每个3D物体有各自的坐标轴，在创建的时候其坐标轴与世界坐标轴一致，若物体旋转后可能就不一致了。

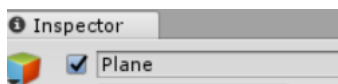
然而在修改物体的时候，可以切换物体的坐标轴为世界坐标轴：



右边的Global按钮意为当前选中物体的坐标轴是与世界一致的，若点一下这个按钮，变为Local，这个时候物体坐标轴就是自身的。

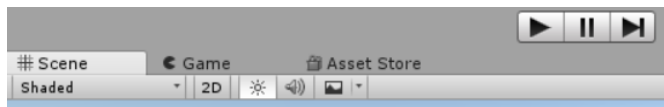
按F键可以聚焦于物体（也可双击其在Hierarchy面板中的文件名），方便观察。

所有在Hierarchy面板中的物体（包括摄像机和光源）都可以在Inspector面板上令其“隐形”：



将复选框上的勾去掉即可，此时该物体并没有消失，仅仅是隐形了。（在Hierarchy面板上其文件名也会变暗）

点击Scene面板右侧的Game面板，可以预览游戏运行时的情况，中上处的播放键可以预览游戏运行情况。（此时所有编辑、修改都无效）



摄像机：

决定游戏运行时我们所看场景的角度，Camera有个立体区域，称为视锥体，物体要在这个区域内才能在游戏中运行时被看到。

可在Camera的Inspector面板中的Clipping Planes中修改该区域的大小，在该面板的Projection中可以调整摄像机的拍摄模式。

（Perspective相当于坐标轴中的Persp，Orthographic相当与Iso）

点击Hierarchy面板中的Main Camera可以调整摄像机的位置，Scene面板右下角也会出现一个预览界面。

在游戏开发时我们可能会调整到一个合适的角度去编辑，若想摄像机应用于此位置，可点击摄像机，然后Ctrl+Shift+F。

在预览时我们可以发现远处的虚空是天空，可在Camera的Inspector面板中的Clear Flags中修改。（虚空为Skybox，纯色为Solid Color）

光源

可在Hierarchy面板右键创建。

默认的光源为方向光（Directional Light，为一束平行光），在其对应的Inspector面板可调节其属性：

Type：灯光类型，有Directional（方向光）、Spot（聚光灯）、Point（点光源）。

Color：光的颜色

Intensity：灯光强度

Shadow Type：阴影类型，有Soft Shadows（软阴影，较为柔和），Hard Shadows（硬阴影，会有锯齿）和No Shadows。

此外，点光源还有Range（光照范围）、聚光灯还有Spot Angle（范围角）等属性。

3.3D模型创建与修改

基本功能

在层级面板中右键可创建3D模型（3D Object），里面有：

Cube：立方体

Sphere：球

Capsule：胶囊体

Cylinder：圆柱

Plane：平地

然后在面板的左上角有以下几个键可以调整这些物体：



1. 手型按钮用于调整用户所看场景的视角，右键可进行旋转。实际上还有另一种视角调整方法：鼠标滚轮缩放，中键移动，Alt+左键旋转，Alt+右键缩放。

2. 四方的箭头按钮用于调整模型位置，可以沿线拉动，也可以沿着某一面拉动（效果是：平行于该面移动）。

3. 旋转按钮用于旋转模型，同样可以沿线旋转和沿面旋转。

4. 缩放按钮用于调整模型大小，可以沿X、Y、Z三轴等比例缩放。

5. 第2、3、4按钮功能的综合。

这些按键的快捷键从左到右分别是Q、W、E、R、T、Y。

复制物体：Ctrl+C 粘贴物体：Ctrl+V 前两个综合：Ctrl+D

材质球（Materials）与贴图（Textures）

用于编辑物体材质、显示效果等。一般会在Assets中建立Materials以及Textures的文件夹存储这些文件。

材质球文件的后缀名为.mat。一个材质球决定一系列的颜色、纹理，往往一个材质球应用于一类模型中。

材质球的使用：

点击材质球属性面板中的Main Maps下的Albedo（漫反射贴图）前的圆点，将对应的贴图应用即可。更高级会使用到Shader（着色器）来调整材质球的属性。

若要将某个材质球上的材质应用于某个物体上，直接将材质球拖到该物体（或者拖到Hierarchy面板上对应物体文件名处）即可。

可调节材质球中纹理的密度：Main Maps下Tiling的值，数值越高，纹理越密。（密到一定程度，就看不出纹理了）

材质球的属性一旦更改，可立即应用于所有采用该材质球的物体上。

空物体和父子关系

在层级面板中可以创建空物体（Create Empty），它相当于文件夹，里面可以放置其他的3D Project。

比如说我要创建一个椅子模型，需要多个Cube，为了方便统一管理，可将这些Cube全部放在一个命名为Chair的空物体中，这样可以统一管理、编辑。

正常物体也可被当作“文件夹”，此时它与别的物体就存在一个父子关系（充当“文件夹”的物体为父）。

编辑子物体，父物体不会与之一同改变，但改变父物体，其所有子物体会跟着变化。

预制体（Prefab）

事先做好的物体，可在游戏开发中大量重复利用。

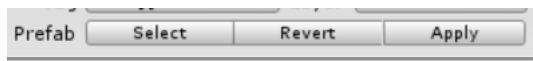
在Assets文件夹中创建Prefabs文件夹，预制体的后缀名是.prefab。

可将Hierarchy面板中做好的物体直接拖入Prefabs文件夹中，使其成为预制体。

将Prefabs中的预制体直接拖入Hierarchy面板从进行应用。

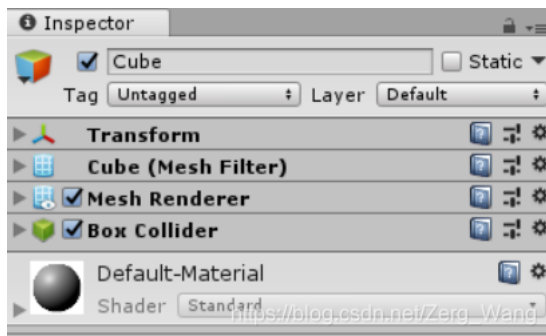
预制体在Hierarchy面板中文件名是蓝色的，非预制体是白色的，且预制体的Inspector面板中多了一个Prefab选项。

若对某个预制体进行了二次编辑，可以点击其右侧Inspector面板中Prefab的Apply，应用于所有预制体（包括Assets文件夹中的）。



组件 (Component)

每个物体都是由多个组件构成的，组件在Inspector面板中：



例如Cube由五个组件构成，有些组件是可选的（可以不用，例如Mesh Renderer，把其复选框中的勾去掉）。

组件控制物体的属性，例如Transform组件，有Position、Rotation、以及Scale，分别确定物体的位置、旋转角度、大小。

Position、Scale中的数值单位为米，Rotation的单位为度。（例如X后面为30，意为以X轴为轴旋转30°）。

小技巧：按住X、Y或Z，然后左右拖动，可以微调数值。

若想重设这些数值，可以点击组件最右边的“齿轮”，里面有Reset选项。

组件也是一种类，每当物体挂载了组件后，系统会自动为我们实例化对象。

4.脚本 (Script)

基本特性

控制游戏运行逻辑的一种特殊组件，Unity 3D支持C#和JavaScript，但C#为主流。

脚本也是一种资源，先在Assets中创建名为Scripts的文件夹，然后再文件夹中创建C# Script，脚本的后缀名为.cs。

双击相对应的脚本，会打开编辑器（本人电脑为Visual Studio 2017）对脚本进行编辑。

若想将脚本应用于某个物体，直接把脚本拖到Hierarchy面板上的该物体处即可。（也可拖到该物体的Inspector面板上）

脚本中有两个方法：Start和Update，Start中的命令在游戏开始时仅执行一次，而Update中的命令会重复执行（每帧一次，每秒60帧）

一个简单的语句：Debug.Log()：相当于Console.WriteLine()，可在括号里面放一段字符串，将这些内容输出到Unity 3D的控制台中。

在Project面板旁边是Console（控制台），可以看到游戏运行时的脚本输出信息，其中有三个按钮：

Clear：清除信息。

Collapse：折叠相同信息，仅更新信息条数。

Clear on Play：运行时清除上次运行产生的信息。

在Update方法之下，其实还有一种方法：FixedUpdate，与Update不同的是，Update是按帧刷新，而每一帧的渲染所用时间可能不同，FixedUpdate是按时间刷新，该方法常用于物体间物理事件的处理（如碰撞）。

在Unity 3D的Edit—Project Setting—Time中的Fixed Timestep可以调整FixedUpdate的时间间隔。

5.物体移动与碰撞

获取键鼠输入

Input.GetKey()：按下某键后持续返回True，直到你抬起。

Input.GetKeyDown()：按下某键的瞬间返回True。

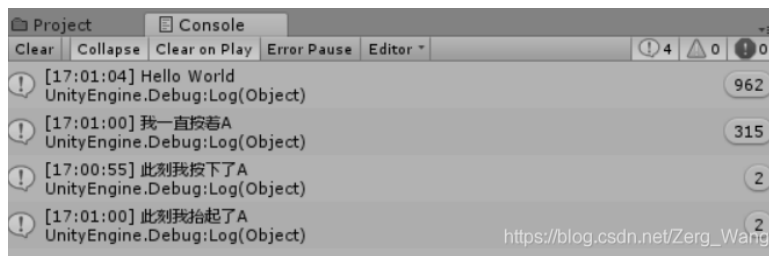
Input.GetKeyUp()：抬起某键的瞬间返回True。

括号里面可以填入一个枚举类型KeyCode，例如，KeyCode.A就指键盘上的A。

```
void Update () {
    Debug.Log("Hello World");
    if (Input.GetKey(KeyCode.A))
    {
        Debug.Log("我一直按着A");
    }
    if (Input.GetKeyDown(KeyCode.A))
    {
        Debug.Log("此刻我按下了A");
    }
    if (Input.GetKeyUp(KeyCode.A))
    {
        Debug.Log("此刻我抬起了A");
    }
}
```

https://blog.csdn.net/Zerg_Wang

对应的控制台输出：



https://blog.csdn.net/Zerg_Wang

鼠标的输入与之相似：

Input.GetMouseButton(); //按下某键后，持续返回 true

Input.GetMouseButtonDown(); //按下某键的一瞬间，返回 true

Input.GetMouseButtonUp(); //抬起某键的一瞬间，返回 true

https://blog.csdn.net/Zerg_Wang

括号内直接填数值即可，0代表左键，1代表右键，2代表中键。

控制物体移动

物体中的所有组件都是类，当组件挂载在物体上后，运行时游戏引擎会自动实例化对象。

我们所编写的脚本其实也是一种类，但这个类相当于这些物体的大脑，可以控制器件。

命令：gameObject.GetComponent<>()，用于引用某物体的组件，尖括号中填写组件名称，返回该组件。

如果我们想控制物体移动，需要用到Transform组件，则首先需要在Start方法中调用到Transform：

```
public class Move : MonoBehaviour {
    private Transform cubeTransform;
    // Use this for initialization
    void Start () {
        cubeTransform = gameObject.GetComponent<Transform>();
    }
}
```

https://blog.csdn.net/Zerg_Wang

在Transform这个类中有一个用于移动物体的无返回值的方法：Transform.Translate(Vector3,Space)

Vector3表示3位向量，用于指示移动方向（Vector3.forward、Vector3.back、Vector3.left、Vector3.right）

Space为一个枚举类型，用于确定物体是相对自己的坐标轴移动（Space.Self）还是世界坐标轴移动（Space.World）

则在Update中：

```
void Update () {
    if (Input.GetKey(KeyCode.A))
    {
        cubeTransform.Translate(Vector3.left, Space.Self);
    }
}
```

https://blog.csdn.net/Zerg_Wang

然后依次把S、D、W的功能完善，就可以了。

注意：若感觉物体移动过快，可以在Vector3后面乘以0.1f减速

刚体

使物体具有重力，并能与其他刚体间发生物理反应。

选中相应物体，在Component中找到Physics选项，添加Rigidbody组件，该组件有以下几个参数：

Mass：物体质量，默认为1（kg）。

Drag：空气阻力，默认为0。

Angular Drag：角阻力，物体受到扭曲力时的空气阻力（当物体旋转下落时受到的横向的空气阻力）

Use Gravity：是否使用重力。

刚体移动：

首先在脚本中要引用到Rigidbody组件：

```
public class RigidMove : MonoBehaviour {  
  
    private Rigidbody cubeRigidbody;  
  
    void Start () {  
        cubeRigidbody = gameObject.GetComponent<Rigidbody>();  
    }  
}
```

https://blog.csdn.net/Zerg_Wang

刚体移动的方法：Rigidbody.MovePosition(Vector3)，该方法默认是相对世界坐标轴移动的。

其中Vector3参数的格式为：当前位置+移动方向

例如：

```
1 | if (Input.GetKey(KeyCode.A))  
2 | {  
3 |     cubeRigidbody.MovePosition(cubeTransform.position + Vector3.left*0.1f);  
4 | }
```

所以在Start方法中还要引用Transform组件。

同样，若移动过快，可在Vector3中乘以0.1

Rigidbody.MovePosition和Transform.Translate功能看似一样，实际上有所不同，后者一般仅仅用于物体的空间上移动，但前者加上了物理效果，其效果更像“走路”。

碰撞体

若要使两个物体之间可以发生物理碰撞，则这两个物体都要有碰撞体组件（Collider）。该组件可在Component的Physics中可添加（其实在创建3D Object时会自动创建）

物体仅有刚体组件而没有碰撞体组件是没有意义的，这样的物体会穿透任何东西，一路往下掉。

一般会将场景中无法穿透的障碍物设为无刚体的碰撞体。

碰撞体组件有以下属性：

Center：碰撞体的中心（相对于物体的坐标，一般在物体的中心，这个也是物体的重心）

Size：碰撞体的大小，如果这个Size超过模型显示的地方，模型以外的地方也会产生碰撞反应（相当于该物体产生了一个无形的力场）

这里也说明一下，我们看到的物体（颜色，纹理），其实是Mesh Renderer组件的作用，该组件只是将物体让用户“可视”，物体其他的属性还需别的组件实现。

碰撞体有多种：Box Collider、Sphere Collider……这些碰撞体仅形状不同而已，不同的3D Object对应不同的碰撞体（实际上你可以往Cube上套一个Sphere Collider），这里比较特殊的是Mesh Collider（网格碰撞体），该碰撞体非常精细，一般用于复杂物体（例如崎岖不平的山地）。

刚体的受力

方法：Rigidbody.AddForce(Vector3,ForceMode)，其中Vector3和之前的一样确定方向（该方法是相对于世界坐标系移动的），ForceMode为枚举类型，有以下四种：

Acceleration：加速度

Force：普通的力，最常用，常用于模拟真实物理情况。

Impulse：冲击力，用于瞬间的发力。

VelocityChange：速度的变化。

例如，我通过鼠标左键发射一枚子弹：

```

1 | if (Input.GetMouseButtonDown(0))
2 | {
3 |     bulletRigidbody.AddForce(Vector3.forward*1000,ForceMode.Force);
4 | }

```

除了AddForce，还有相对于自身坐标系移动的AddRelativeForce这个方法，其他参数两者都是一样的。

使用AddRelativeForce移动球体时，球体不会一直加速出去，而是会原地来回移动，因为球体滚动后，其自身坐标系也发生改变，下一帧力的作用方向也就改变了，从而产生这种现象。

碰撞检测

方法：OnCollisionEnter、OnCollisionExit、OnCollisionStay（这三个方法与Start和Update同级）

分别在：碰撞开始的瞬间、碰撞结束的瞬间、碰撞时启用（与Input.GetKey类似）。

这三个方法均包含一个参数：Collision的类，用于返回与当前物体相撞的物体的信息。

代码示例：

```

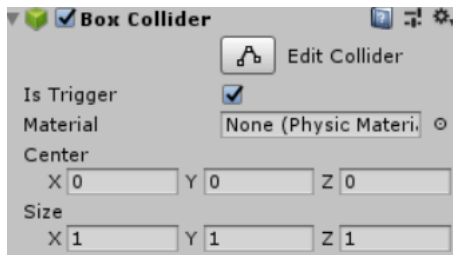
1 | void OnCollisionEnter(Collision coll)
2 | {
3 |     Debug.Log(coll.gameObject.name);
4 | }

```

相撞瞬间被撞物体的名字会在Console中显示，其他两个方法类似。

触发检测

在物体的碰撞体组件中可以将该物体的碰撞体调整为触发器：勾选上Is Trigger即可：



这里可以调整触发范围（比如说，我想把该物体设置为自动感应门，其触发范围会在该物体之前）

若物体的碰撞体调为触发器，则其他物体可以直接穿过它，所有有关Collision的方法、函数也不会触发。

触发检测：OnTriggerEnter、OnTriggerExit、OnTriggerStay（功能与上述类似）

这些方法的参数仍然是一个Collision类，与碰撞检测一样。

旋转

方法：Transform.Rotate(Vector3,float)

其中，Vector3指定沿哪个轴转动，float表示旋转角度。例如，按下E键开门：

```

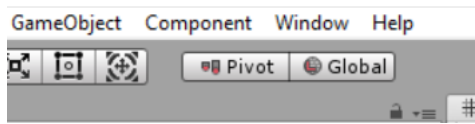
1 | if (Input.GetKeyDown(KeyCode.E))
2 | {
3 |     doorTransform.Rotate(Vector3.up, 90);
4 | }

```

Vector3.up表示沿Y轴正方向转动（forward、down、left、right等分别对应其他轴，均可使用）

物体自身坐标轴的原点是其中心点，但门这样的物体是不会绕自身中心点旋转的，因此可通过父物体的方式将门的中心点移动到一侧。

多个物体共同中心点的设置：



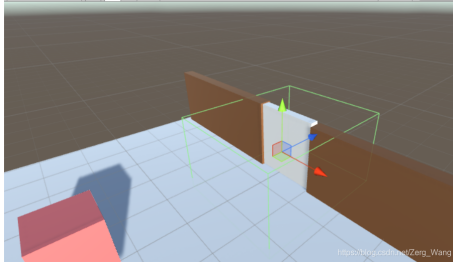
Global左侧的按钮有两个状态：Pivot和Center。

在Pivot状态下选中多个物体，共同中心点是最后点选的物体的中心点，若是Center状态，则共同中心点位于各个物体中心点的中心。

自动旋转门

结合触发器使用。首先要添加一个空物体作为触发器。（不能把门设置为触发器，防止其他物体穿越）

为该物体添加Box Collision，调整好触发范围后：



然会用脚本在DoorTrigger中调用到使Door旋转的方法：

首先要查找到提供旋转功能的脚本：

每一个脚本也是一个类，可以被其他脚本调用。

对于触发器：

```
1 private DoorRotate doorR;//调用使门旋转的脚本
2 void Start()
3 {
4     doorR = GameObject.Find("DoorF").GetComponent<DoorRotate>();
5 }
6 //注意区分: GameObject是一个静态方法，与之前用的gameObject不一样，该方法用于得到被旋转的对象（即门）
7 void OnTriggerEnter(Collider coll)
8 {
9     if (coll.gameObject.name == "Person") //要使人来了才自动开门，其他物体一律不开
10    {
11        Debug.Log("In");
12        doorR.OpenDoor();
13    }
14 }
15
16 void OnTriggerExit(Collider coll)
17 {
18     if (coll.gameObject.name == "Person")
19     {
20         Debug.Log("Out");
21         doorR.CloseDoor();
22     }
23 }
```

对于门，把OpenDoor和CloseDoor完善即可：

```
1 public void OpenDoor()
2 {
3     doorTransform.Rotate(Vector3.up, 90);
4 }
5 public void CloseDoor()
6 {
7     doorTransform.Rotate(Vector3.down, 90);//这里参数也可以是: Vector3.up, -90
8 }
```


6.模型渲染

决定物体显示效果的组件： Mesh Filter和Mesh Renderer

Mesh Filter（网格过滤器）用于决定物体形状（方块、球.....）

Mesh Renderer（网格渲染器）决定物体的阴影效果、材质球等，其中：

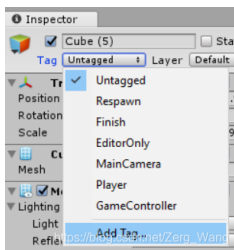
Cast Shadows：是否显示该物体的阴影

Receive Shadows：是否显示其他物体投射到该物体上的阴影

7.标签（tag）

为方便管理模型，可为模型添加标签：在Inspector面板中操作。

Unity 3D自带多个标签，但也可自定义。



可用以下静态方法批量得到同标签的物体：GameObject.FindGameObjectsWithTag(String)

string为标签名，该方法返回GameObject类的数组。

注意：另外一个方法GameObject.FindGameObjectWithTag的Object后面没有s，是对单个物体使用的。

```
public class TagTest : MonoBehaviour {  
  
    private GameObject[] objectArray;  
    // Use this for initialization  
    void Start () {  
        objectArray = GameObject.FindGameObjectsWithTag("cube");  
    }  
}
```

https://blog.csdn.net/Zerg_Wang

批量更改游戏物体属性：（例如，所有物体在按W键后向上移动2米）

```
void Update () {  
    if (Input.GetKeyDown(KeyCode.W))  
    {  
        for (int i=0;i<objectArray.Length;i++)  
        {  
            objectArray[i].GetComponent<Transform>().Translate(Vector3.up*2, Space.Self);  
        }  
    }  
}
```

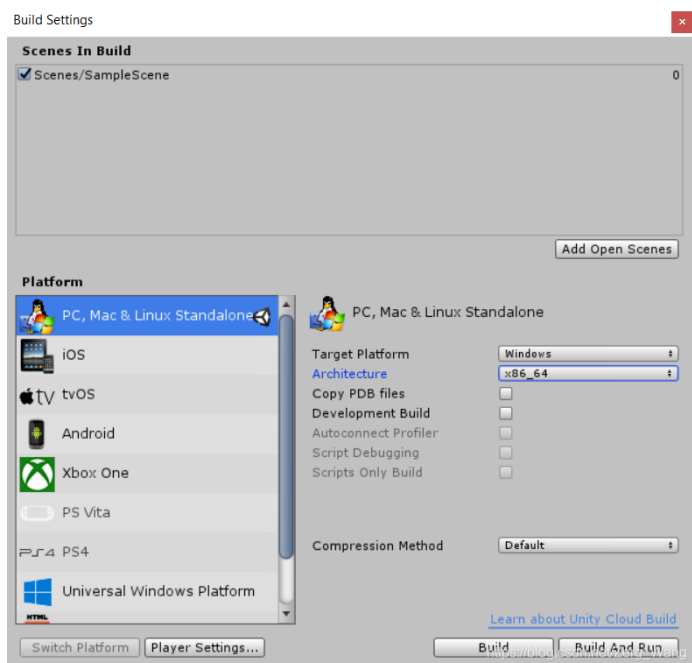
https://blog.csdn.net/Zerg_Wang

其中，中间那句话是之前属性定义和GetComponent的合并写法。

该脚本可以应用于被移动物体本身，也可以用于其他物体。

8.游戏打包发布

File—Build Settings，然后选择发布平台以及要发布的场景以及发布者信息（Player Settings）：



之后打包出来的文件有：

名称	日期/时间	类型
Mono	12/5/2018 12:01 AM	File folder
MoveDemo_Data	12/5/2018 12:01 AM	File folder
MoveDemo.exe	6/15/2018 9:58 PM	Application
UnityPlayer.dll	6/15/2018 10:06 PM	Application extension
UnityCrashHandler64.exe	6/15/2018 10:06 PM	Application

其中最下面那个exe是可删除的，运行游戏点击MoveDemo.exe，Data文件夹存放游戏数据。

本文部分内容来自撻码网（<http://www.mkcode.net>）Unity 3D课程，经本人学习、整理得来，若有错漏，欢迎指正！