**Unity3D教程：Javascript的使用介绍（一）**

Posted on 2013年06月07日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 102 次

在Unity中脚本是必不可少的。因为他将定义你游戏的各种行为和规则。

**目标**

在Unity中，脚本是用来界定用户在游戏中的行为或规则。Unity推荐使用的编程语言是JavaScript，同时也支持C#或Boo。

**命名规范**

变量 - 首写为小写字母。变量用来存储游戏状态中的任何信息。

函数 - 首写为大写字母。函数是一个代码块，在需要的时候可以被重复调用。

类 - 首写为大写字母。可以被认为是函数的库。

当阅读范例时注意首写字母，将有助于你更好的理解对象之间的关系。

**用户输入**

例子：在场景中实现一个简单的移动。

1)设置场景：

创建一个用来移动的平面。GameObject->CreateOther->Plane。并且在Inspector面板中设置Position为“0，0，0”。如果当前页面没有Inspector面板，选择Window->Layouts->2by 3。建议熟悉各种布局以便开发需要。

创建一个Cube。GameObject->CreateOther->Cube。在Inspector面板中设置Position为“0，1，0”。

我们都知道现实世界里物体成像靠的是光反射，那么我们这里也是需要光线的。选择GameObject->CreateOther->PointLight。在Inspector面板中设置坐标为“0，5，0”。Unity3D教程手册

保存当前场景。快捷键为Ctrl+s。

2)新建脚本：

我们打算移动用户的视线，这需要通过控制主相机的位置来实现。我们就要写一个脚本，然后把脚本和相机结合起来。

创建一个空脚本。Assets->Create->JavaScript并命名为“Move”。重命名快捷键为F2。

双击打开脚本Move。默认包含Update()函数。将我们的代码加入这个函数，他将在每一帧执行一次。

我们需要用transform来改变相机的位置，用到Translate这个函数，他有x，y，z三个参数。我们加入以下代码:

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | function Update(){ |
| 2 | transform.Translate(Input.GetAxis("Horizontal"),0,Input.GetAxis("Vertical")); |
| 3 | } |

Input.GetAxis()函数返回一个从-1到1之间的值，如横轴上左半轴为-1到0，右半轴为0到1。如果需要，可以通过Edit->ProjectSettings->Input中重定义按键映射。

3)连接脚本

脚本写完了，如何让他起作用呢？我们需要把脚本赋予物体才行。

点击希望应用脚本的物体对象。这里对我们而言就是相机。从Hierarchy面板中选中它。

在菜单中选择Components->Scripts->Move，这样我们便从Inspector面板中看到相机中添加了Move这个组件。(我们也可以用鼠标把脚本拖拽到物体对象上)Unity3D教程手册

点击运行，我们即可前后左右按键来控制相机移动了。

注意一个关于DeltaTime的问题。放在Update()函数中的代码是按照帧来执行的。

如果我们需要物体的移动按照时间秒来执行，我们需要将Input.GetAxis()函数的返回值乘以Time.deltaTime:

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | **var** speed = 5; |
| 2 | function Update(){ |
| 3 | **var** x = Input.GetAxis("Horizontal")\*Time.deltaTime\*speed; |
| 4 | **var** y = Input.GetAxis("Vertical")\*Time.deltaTime\*speed; |
| 5 | transform.Translate(x, y, 0); |
| 6 | } |

这样设置之后按下WASD便可以实现上下左右的视角移动了。

这里的speed为显式变量，可以在Inspector面板中看到。我们可以在使用中随时调整它的值，很方便。

**连接变量**

Unity允许在界面上拖拽(drag and drop)的方式赋脚本。快捷方便。这里，我们将涉及连接变量的概念。

在场景中添加一个聚光灯。GameObject->CreateOther->SpotLight。Position为“0，5，0”，Rotation为“90，0，0”。

创建一个JavaScript脚本，命名为“SpotLight”。

我们想让聚光灯照向主相机。我们可以使用transform.LookAt()这个函数。

如何使用这个函数呢？我们需要创建一个显式变量。在SporLight脚本中写入如下代码:

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | **var** target : Transform; |
| 2 | function Update(){ |
| 3 | transform.LookAt(target); |
| 4 | } |

把脚本赋予聚光灯对象。要么在菜单栏Component中添加，要么鼠标拖放。之后，“target”变量就出现在Inspector面板里了。

将Hierarchy面板中的主相机对象拖放到Target变量上。如果我们想让聚光灯照向其他物体，我们也可以将其他物体拖放上去，只要是Transform类型的即可。

运行。你可以看到聚光灯一直照向主相机。

**Unity3D教程：Javascript的使用介绍（二）**

Posted on 2013年06月07日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 86 次

**访问组件**

一个游戏对象可能有多个脚本或其他组件。它将不可避免的要访问其他组件的函数或变量。Unity中通过GetComponent()函数来实现这个目的。我们现在要实现按下空格键后让聚光灯照向Cube。

我们考虑一下这个步骤:1.监视空格键被按下。2.空格键按下后聚光灯照向Cube。

由于SporLight脚本中有target变量，我们只需要为这个变量设定新的值就可以了。

创建一个脚本，命名为Switch，添加如下代码:

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | **var**&nbsp;switchToTarget&nbsp;:&nbsp;Transform;&nbsp;&nbsp; |
| 2 | function&nbsp;Update(){&nbsp;&nbsp; |
| 3 | &nbsp;&nbsp;&nbsp;&nbsp;if(Input.GetButtonDown("Jump"))&nbsp;&nbsp; |
| 4 | &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;GetComponent(SpotLight).target&nbsp;=&nbsp;switchToTarget;&nbsp;&nbsp; |
| 5 | } |
| 6 |  |

注意GetComponent()的参数，它将返回一个参数给Switch脚本，我们就可以用它来访问“target”变量。

这里的GetComponent获取的不是点光源，而是那个名为SpotLight的JS文件。

添加Switch脚本到聚光灯对象，并将Cube拖放到Inspector面板中的switchToTarge变量上。

运行。按下空格后，聚光灯将照向Cube。

前面我们提到，可以通过写代码的方式指定变量(而不是通过Unity界面)，如何做到呢?上面按下空格时告诉聚光灯聚焦到Cube上，我们是在SpotLight脚本中做一个显式的变量，将Cube拖放上去。而在代码中主要有2种方法去实现：使用对象的名称(name)与使用对象的标签(tag)

1.对象名称

可以从Hierarchy面板上看到对象的名称。用GameObject.Find()函数来使这些名称作为参数。因此，我们可以这样写:

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | function Update(){ |
| 2 | **if**(Input.GetButtonDown("Jump")) |
| 3 | { |
| 4 | **var** newTarget = GameObject.Find("Cube").transform; |
| 5 | GetComponent(SpotLight).target = newTarget; |
| 6 | } |
| 7 | } |

注意，以上没有出现显式的变量。

2.对象标签

对象标签是一个字符串，用来识别一个组件。在Inspector面板中点击Tag按钮查看内建的标签。我们也可以创建自己的标签。GameObject.FindWithTag()函数能通过具体的标签寻找组件并返回一个字符串作为参数。

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | function Update(){ |
| 2 | **if**(Input.GetButtonDown("Jump")) |
| 3 | { |
| 4 | **var** newTarget = GameObject.FindWithTag("fang").transform; |
| 5 | GetComponent(SpotLight).target = newTarget; |
| 6 | } |
| 7 | } |

**实例**

我们要在运行时创建(create)对象就要用到Instantiate函数。让我们来实现如何在每次按下开火按钮(鼠标左键或Ctrl键)后通过实例化一个新的对象。需要思考以下几点:

1.哪个物体做我们的实例?

2.在哪儿实例化?

关于第一个问题，最好的办法就是显式变量。这样我们就可以随时通过拖放来改变我们的实例对象。至于在哪儿实例化，我们只要实现当按下开火键时，在用户的当前位置创建一个对象即可。实例化函数有3个参数:1.我们要建立的对象。2.对象的位置坐标。3.对象的旋转位置。完整的代码如下:

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | **var** newObject : Transform; |
| 2 | function Update(){ |
| 3 | **if**(Input.GetButtonDown("Fire1")){ |
| 4 | Instantiate(newObject, transform.position, transform.rotation); |
| 5 | } |
| 6 | } |

要记住，transform、position、transform、rotation是附加这个脚本的物体的位置。我们这里假设为主相机。一般情况下，将要被实例化的对象设置为预设(prefab)，我们现将Cube设置为预设。

首先，让我们创建一个预置。Assets->Create->Prefab。命名为Cube。从Hierarchy面板中选取Cube拖放到Project面板的Cube上。创建一个脚本并命名为Create，并把上面的代码加进去。把这个脚本赋予相机，并将Cube预设赋予脚本变量。运行。移动相机并按下开火键，你能看到新的Cube出现。

**调试**

调试是发现和修正你的代码中人为错误的技巧，Unity中提供了Debug类，我们现在看看Debug.Log()函数。Log()函数允许用户发送信息到Unity的控制台。这样做的原因可能包括:

1.在运行时要验证某部分代码是否达成。

2.报告变量的状态。

我们现在使用Log()函数，当用户点击开火按钮时，发送一个消息到Unity控制台。

打开创建的脚本并在“Instantiate”代码内“if”处添加如下代码:

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 1 | Debug.Log("Cube created"); |

运行。点击开火键，我们能看到Unity界面下方显式“Cube created”。

另外一个有用的功能是用于调试私有变量。这使得当Debug模式被选中时Inspector面板中的变量，但它不能被编辑。为了证明这一点，我们将显式一个私有变量作为Cube实例的计数器。我们在脚本中加入两行内容1.添加私有变量cubeCount。2.当一个cube实例创建后增加这个变量。完整代码如下:

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | **var** newObject : Transform; |
| 02 | **private** **var** cubeCount = 0; |
| 03 | function Update(){ |
| 04 | **if**(Input.GetButtonDown("Fire1")) |
| 05 | { |
| 06 | Instantiate(newObject, transform.position, transform.rotation); |
| 07 | Debug.Log("Cube created"); |
| 08 | cubeCount++; |
| 09 | Debug.Log(cubeCount); |
| 10 | } |
| 11 | } |

**常见脚本类型**

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | *//在这个函数体中的代码每隔固定的间隔执行。它通常在Rigibody中有力的作用的时候被用到。如:* |
| 02 |  |
| 03 | *// Apply a upwards force to the rigibody every frame* |
| 04 |  |
| 05 | function FixedUpdate(){ |
| 06 |  |
| 07 | rigidbody.AddForce(Vector3.up); |
| 08 |  |
| 09 | } |
| 10 |  |
| 11 | *//这里的代码将被用作初始化。* |
| 12 |  |
| 13 | Awake(){ |
| 14 |  |
| 15 | } |
| 16 |  |
| 17 | *//这个函数在Update()之前，但在Awake()之后执行。* |
| 18 |  |
| 19 | *//Start()函数和Awake()函数的不同点在于Start()函数仅在脚本启用时候执行(Inspector面板中的复选框被选中)。* |
| 20 |  |
| 21 | Start(){ |
| 22 |  |
| 23 | } |
| 24 |  |
| 25 | *//当游戏对象的碰撞脚本与另一个游戏对象碰撞时执行这个函数内的代码。* |
| 26 |  |
| 27 | OnCollisionEnter(){ |
| 28 |  |
| 29 | } |
| 30 |  |
| 31 | *//当鼠标在一个载有GUI元素(GUIElement)或碰撞器(Collider)的游戏对象里按下时执行该函数体内的代码。* |
| 32 |  |
| 33 | *// Loads the level named “SomeLevel” as a response* |
| 34 |  |
| 35 | *// to the user clicking on the object* |
| 36 |  |
| 37 | function OnMouseDown(){ |
| 38 |  |
| 39 | Application.LoadLevel(“SomeLevel”); |
| 40 |  |
| 41 | } |
| 42 |  |
| 43 | *//当鼠标悬停在一个GUI元素或碰撞器的对象上时，执行该函数体内的代码。如:* |
| 44 |  |
| 45 | *// Fades the red component of the material to zero* |
| 46 |  |
| 47 | *// while the mouse is over the mesh* |
| 48 |  |
| 49 | function OnMouseOver(){ |
| 50 |  |
| 51 | renderer.material.color.r -= 0.1 \* Time.deltaTime; |
| 52 |  |
| 53 | } |