**Unity3D教程：脚本初级知识（三）**

Posted on 2013年01月29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 266 次

**成员变量 & 全局变量**

定义在任何函数之外的变量是一个成员变量。在Unity中这个变量可以通过检视面板来访问，任何保存在成员变量中的值也可以自动随工程保存。

var memberVariable = 0.0;

上面的变量将在检视面板中显示为名为"Member Variable"的数值属性。如果你设置变量的类型为一个组件类型（例如Transform, Rigidbody, Collider，任何脚本名称，等等）然后你可以在检视面板中通过拖动一个游戏物体来设置它们。

var enemy : Transform;  
function Update()  
{  
if ( Vector3.Distance( enemy.position, transform.position ) < 10 );  
print("I sense the enemy is near!");  
}  
}

你也可以创建私有成员变量。私有成员变量可以用来存储那些在该脚本之外不可见的状态。私有成员变量不会被保存到磁盘并且在检视面板中不能编辑。当它被设置为调试模式时，它们在检视面板中可见。这允许你就像一个实时更新的调试器一样使用私有变量。

private var lastCollider : Collider;  
function OnCollisionEnter( collisionInfo : Collision ) {  
lastCollider = collisionInfo.other;  
}

**全局变量**

你也可以使用static关键字创建全局变量,这创造了一个全局变量，名为someGlobal。

static var someGlobal = 5;// 你可以在脚本内部像普通变量一样访问它  
print(someGlobal);// 'TheScriptName.js'中的一个静态变量  
someGlobal = 1;

为了从另一个脚本访问它，你需要使用这个脚本的名称加上一个点和全局变量名。

print(TheScriptName.someGlobal);  
TheScriptName.someGlobal = 10;

**实例化**

实例化，复制一个物体。包含所有附加的脚本和整个层次。它以你期望的方式保持引用。到外部物体引用的克隆层次将保持完好，在克隆层次上到物体的引用映射到克隆物体。

实例化是难以置信的快和非常有用的。因为最大化地使用它是必要的。例如， 这里是一个小的脚本，当附加到一个带有碰撞器的刚体上时将销毁它自己并实例化一个爆炸物体。

var explosion : Transform;  
// 当碰撞发生时销毁我们自己  
// 并生成给一个爆炸预设  
function OnCollisionEnter (){  
Destroy (gameObject);  
var theClonedExplosion : Transform;  
theClonedExplosion = Instantiate(explosion, transform.position, transform.rotation);  
}

实例化通常与预设一起使用。

**Coroutines & Yield**

在编写游戏代码的时候，常常需要处理一系列事件。这可能导致像下面的代码。

private var state = 0;  
function Update()  
{  
if (state == 0) {  
state = 1;// 做步骤0  
return;  
}  
if (state == 1) {  
state = 2;// 做步骤1  
return;  
}  
// …  
}

更方便的是使用yield语句。yield语句是一个特殊类型的返回，这个确保在下次调用时该函数继续从该yield语句之后执行。

while(\*\*\*e) {  
yield; //等待一帧  
// 做步骤2  
yield; //等待一帧  
// 做步骤1  
// ...  
}

你也可以传递特定值给yield语句来延迟Update函数的执行，直到一个特定的事件发生。

// 做一些事情  
yield WaitForSeconds(5.0); //等待5秒  
//做更多事情…

可以叠加和连接coroutines。这个例子执行Do，在调用之后立即继续。

Do ();  
print ("This is printed immediately");  
function Do ()  
{  
print("Do now");  
yield WaitForSeconds (2);  
print("Do 2 seconds later");  
}

这个例子将执行Do并等待直到它完成，才继续执行自己。

yield StartCoroutine("Do");//链接coroutine  
print("Also after 2 seconds");  
print ("This is after the Do coroutine has finished\*\*\*cution");  
function Do ()  
{  
print("Do now");  
yield WaitForSeconds (2);  
print("Do 2 seconds later");  
}

任何事件处理句柄都可以是一个coroutine ,注意你不能在Update或FixedUpdate内使用yield，但是你可以使用StartCoroutine来开始一个函数。参考YieldIns\*\*\*ction, WaitForSeconds, WaitForFixedUpdate, Coroutine and MonoBehaviour.StartCoroutine获取更多使用yield的信息。