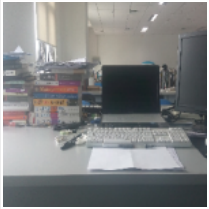


个人资料



yongtao_liu

+ 加关注 私信

访问：29836次
积分：1174
等级：BLOG 4
排名：千里之外
原创：84篇 转载：32篇
译文：1篇 评论：0条

文章搜索

文章分类

- unity&cocos2dx (31)
- C++ (12)
- windows (15)
- linux (14)
- node.js/pomelo/v8 (7)
- go (0)
- lua学习笔记 (14)
- C#加强及设计模式 (13)
- 数据结构与算法 (6)
- vs使用技巧 (3)
- 网络 (2)
- shader (2)



小企业
促销月

购任意笔记本
+9元优惠购包鼠！



文章存档

- 2016年07月 (1)
- 2016年06月 (1)
- 2016年05月 (1)

程序员3月书讯 CSDN日报20170406 ——《代码很烂，所以离职。》 Python数据分析与机器学习 博客搬家，有礼相送

u3d中脚本的知识点

标签：游戏 unity3d

2015-05-23 21:04 513人阅读 评论(0) 收藏 举报

分类： unity&cocos2dx (30)

版权声明：本文为博主原创文章，未经博主允许不得转载。

快速回复

我要收藏

返回顶部

一 创建和使用脚本

1 概述

GameObject的行为都是被附加到其上面的组件控制，脚本本质上也是一个组件。

在unity中创建一个脚本，脚本通过实现一个派生自“MonoBehaviour”的类来与unity的内部工作机制建立联系。可以将新创建的组件类型的类作为一个蓝图，该类作为一个新类型的组件被附加到游戏对象。每次将一个脚本组件附加到游戏对象，都会创建一个该蓝图定义的对象实例。创建的脚本文件的文件名必须与里面的类名相同，这样才能将其附加到游戏对象上。

Update函数处理游戏对象的帧更新相关的操作。可能包括移动，触发动作以及对用户输入的反馈，基本上在游戏过程期间需要处理的任何事情都可以在这里面处理。Start函数在游戏开始前被unity调用（例如，在Update被第一次调用之前），因而是一个进行初始化操作的理想位置。为什么不把初始化操作放在类的构造函数中，这是因为对象的构造是由编辑器处理的，在游戏开始的时候并不如想象中那样会发生。如果为一个脚本组件定义构造器，它会和unity的正常操作发生干扰从而导致一些问题。脚本被创建之后是处于不激活状态的，只有将它的一个实例附加到一个游戏对象之后代码才会被激活。同时，一个游戏对象的一个类型的组件只能有一个，也就是可以附加到游戏对象的脚本组件不能超过一个。

2变量

就是指类的成员变量，只不过在unity这里将成员变量设为公有的时候，将其附加到游戏对象后，可以在游戏对象的监视面板中的脚本组件那栏里面看到该公有变量，也就是说可以在编辑器里面直接对该公有变量进行赋值，同时在debug状态下也可以在面板中看到它的值。

unity在编辑器中显示类的成员变量的时候，在类名中遇到（不是第一个字母的）大写字母的时候，会在该字母前加入一个空格，例如在类里面定义个公有成员变量名为“TestVar”，在编辑器中显示的变量名为“Test Var”，中间加了一个空格，但是这只是一个unity显示变量的方式，真正在代码中访问这个变量仍然应该使用“TestVar”这个名字。

3 事件函数

unity中的脚本并不像传统意义上的程序，即在一个循环中连续的运行直到完成任务后再退出，而是unity通过调用定义在脚本内的某个函数，间断的将控制权交给一个脚本。一旦一个函数就是所知的事件函数，因为它们由unity调用用于响应发生在游戏过程

<1> 常规更新事件(Regular Update Events)

一个游戏更像一个在线生成动画帧的动画。游戏编程中的一个关键观念就是行为和进行改变。Update函数是unity中放置这类代码的主要地方，它

update事件函数

```
void Update() {  
    float distance = speed * Time.deltaTime *  
    Input.GetAxis( "Horizontal" );
```



2015年12月 (1)

2015年10月 (6)

展开

推荐文章

- * Android安全防护之旅---带你把Apk混淆成中文语言代码
- * TensorFlow文本摘要生成 - 基于注意力的序列到序列模型
- * 创建后台任务的两种代码模式
- * 一个屌丝程序猿的人生（六十）
- * WKWebView与js交互之完美解决方案
- * 年轻人，“砖砖瓦瓦”不应该成为你的梦想！

```
transform.Translate(Vector3.right * speed);
}
```

物体引擎也以和一个帧渲染类似的方式每隔一个离散时间段进行更新。另一个叫FixedUpdate的函数在每次物理更新之前调用。由于物理更新和帧更新并不使用相同的频率更新，如果将上面的代码放到FixedUpdate函数而不是Update函数的话，可以得到更加精确的结果（即FixedUpdate会以一个比Update更加稳定的帧率运行）。

```
void FixedUpdate() {
    Vector3 force = transform.forward * driveForce *
    Input.GetAxis( "Vertical" );
    rigidbody.AddForce(force);
}
```

同时在场景中所有对象的Update和FixedUpdate调用之后，以及所有的动画计算之后仍然可能需要进行一些额外的改变。一个例子就是在一个摄像头需要监视一个目标对象的时候，对摄像机朝向的调整必须在目标对象已经移动之后。另一个例子是在在需要用脚本代码来覆盖动画效果的场合（比如，使角色的头看向场景中的目标对象）。

LateUpdate函数可以用于这些场合。

<2> 初始化事件

在游戏中的任何更新函数之前调用初始化代码是经常用到的，初始化函数Start在物体的第一帧更新之前被调用。Awake函数在场景被加载的时候，场景中的每个对象上的该函数都会被调用，所有的Awake函数都会运行在第一个Start函数（有好多对象的话就有好多Start函数）调用之前被调用。这就意味着Start函数中使用的数据是在Awake阶段里面的初始化后的数据。

<3> GUI事件

因而它需要放到OnGUI函数中，周期性的被调用unity有一个用于渲染GUI控件的系统，这些GUI控件处于场景之上并且响应鼠标事件。这个代码处理起来和常规的帧更新有些不大一样。

```
void OnGUI() {
    GUI.Label(labelRect, "Game Over" );
}
```

同时当发生鼠标单击场景中的游戏对象的时候可以检测这些消息。这个可以用于调整武器或者显示当前在鼠标下面的角色信息。一系列的OnMouseXXX事件函数可以用于和用户的鼠标行为交互。

<4> 物理事件

物理引擎会通过调用物体上的事件函数来报告发生的与另一物体的碰撞事件。OnCollisionEnter、OnCollisionStay和OnCollisionExit函数分别在刚接触，持续和离开(broken)的时候调用。当碰撞器被配置成触发器的时候发生碰撞后对应的OnCollisionEnter、OnCollisionStay和OnCollisionExit会被调用。假如在物理更新期间有多于一个接触被检测到这些函数可能会被多次按顺序调用（即调用完一轮再来一轮）。

二 控制游戏对象

在unity中，可以在监视器面板中修改物体的组件属性，但是更多的时候，需要使用脚本来进行这些操作。

1访问组件

常见的一个情形是需要使用脚本访问附加到相同游戏对象上的另一个组件（当前脚本就是一个组件，其他的组件也就是另一个组件了）。一个组件实质上是一个类的实例，因而首先需要做的是获取想要操作的组件实例的引用。这个通过GetComponent函数来实现。典型的，可能会想要将一个组件赋值给一个变量，如下代码所示

```
void Start () { Rigidbody rb = GetComponent();}
```

一旦获取了组件实例的引用，就可以对它的属性进行想要的操作，同时也可以调用它之上的一些功能函数。如果想要访问另一个的脚本文件，也可以使用GetComponent，只需使用脚本的类名作为该函数的组件类型参数（因为脚本本来就也是一个组件）。如果想要去获取一个并没有添加到当前游戏对象的组件，GetComponent函数会返回null，如果试图去改变一个null对象上的任何值，将会发生null引用错误

了一些内置的变量来访问它们，例如可以使用下面的代码

```
void Start () {
    transform.position = Vector3.zero;
}
```

而不用使用GetComponent去获得Transform组件，所有的内置组件变量都有。

2 访问其他对象

虽然游戏对象有的时候都是各自处理，使用代码进行跟踪其他物体是常



小企业
促销月

购任意笔记本
+9元优惠购包鼠！



英特尔 广告



道玩家的位置，unity提供了一系列不同的方法来获取其他对象，各适合不同的场合。

<1>将对象链接到变量

直接的办法是将一个游戏对象添加到脚本的公有成员变量上，直接在编辑器中将需要访问的游戏对象拖到对应脚本组件的那个公有成员变量上，unity会自动根据变量的类型将添加的游戏对象中相同的组件类型映射到该变量。

例如将一个游戏对象拖给一个Transform的成员变量，就会自动的将游戏对象的Transform组件和该变量映射起来。

直接将对象和变量链接起来在处理需要有永久链接的对象的时候是最有用的方法。同时也可以使用一个数组变量和几个相同类型的对象链接起来，但是这种链接必须在unity编辑器中完成，而不能在运行时进行。通常使用下面的两种方法来在运行时定位对象。

<2>查找子物体

有的时候，一个游戏场景中可能会用到很多同一类型的对象，例如敌人、路点（waypoints）和障碍物。这些对象在游戏中需要由一个特定的脚本来监视和响应。这个时候使用变量来链接这些对象太过麻烦不好操作。对于这种情况，通常更好的方法是将一系列的对象添加到一个父对象下面，这些子对象可以通过使用父对象的Transform组件来获得。

```
public class WaypointManager : MonoBehaviour {
    public Transform waypoints;

    void Start() {
        waypoints = new Transform[transform.childCount];
        int i = 0;

        for (Transform t in transform) {
            waypoints[i++] = t;
        }
    }
}
```

同时也可以使用Transform.Find来查找某个具体的子对象。使用Transform来进行对象查找操作是因为每一个游戏对象都有Transform组件。

<3>通过名称或标签访问对象

只要有一些信息，在层级场景中的任何位置定位到该游戏对象是可能的。单个对象可以通过GameObject.Find函数进行查找。如下：

```
GameObject player;
void Start() {
    player = GameObject.Find( "MainHeroCharacter" );
}
```

某个对象或者一系列的对象也可以分别通过GameObject.FindWithTag和GameObject.FindObjectsWidthTag函数进行定位。

<4> 查找特定类型的对象

static Object FindObjectOfType(Type type) 返回指定类型对象中的第一个活动的加载对象

需要注意的是这个函数很慢（可能是由于要在整个场景中进行遍历），不推荐每一帧都使用这个函数，在大多数情况下可以使用单件模式

例如

```
Camera cam = FindObjectOfType(typeof(Camera)) as Camera;
```

由于该函数返回的类型是Object，所以需要使用时进行一下强制转换。

```
static Object[] FindObjectsOfType(Type type);
```

返回指定类型的加载活动对象的列表，速度也慢

```
HingeJoint[] hinges = FindObjectsOfType(typeof(HingeJoint)) as HingeJoint[];
```

三 创建和销毁对象

在运行时创建和销毁对象是常有的事。在unity中，可以使用Instantiate来创建新的游戏对象。

```
public GameObject enemy;
void Start() {
    for (int i = 0; i < 5; i++) {
        Instantiate(enemy);
    }
}
```

快速回复

我要收藏

返回顶部



```
}  
}
```

值得注意的是用于进行拷贝的对象并不一定需要放置在场景中。更普遍的做法是将一个预设(Prefab)拖到脚本的对应公有成员变量上，实例化的时候直接对这个成员变量进行实例化即可。

同时也有一个Destroy函数在帧更新函数完成后或设定的一个延时时间后销毁一个对象。

```
void OnCollisionEnter(otherObj: Collision) {  
    if (otherObj == "Missile" ) {  
        Destroy(gameObject,.5f);  
    }  
}
```

注意到Destroy函数可以销毁单独的组件而不对游戏对象本身产生影响，一个通常范的错误是Destroy(this);这句代码仅仅影响脚本组件，而不会销毁该脚本所附加在的对象。

四协程(Coroutines)

当调用一个函数的时候，它会在返回前运行到结束位置。

几乎所有的函数都由unity在帧更新的时候调用一次，然后到下一帧后再调用一次，那这个时候在函数中写一些循环体类的代码想要达到产生动画的效果可能会失败。如下

```
void Fade() {  
    for (float f = 1f; f >= 0; f -= 0.1f) {  
        Color c = renderer.material.color;  
        c.alpha = f;  
        renderer.material.color = c;  
    }  
}
```

这是想要产生一个透明渐变的效果，但是由于Fade函数会在一个frame

update时间片中全部执行完，达不到渐变效果。要达到这个效果，可以在帧更新函数中进行处理即可。但是，对于这类情形更方便的做法是使用coroutine。

一个coroutine就像一个可以暂停执行并将控制权返回给unity的函数，但是在下一帧的时候又可以在它停止的位置继续执行。在C#中，这样声明一个coroutine：

```
IEnumerator Fade() {  
    for (float f = 1f; f <= 0; f -= 0.1f) {  
        Color c = renderer.material.color;  
        c.alpha = f;  
        renderer.material.color = c;  
        yield return;  
    }  
}
```

实质上它是一个返回类型为IEnumerator的函数，同时在函数体中增加了yield return这句代码。yield return这行就是会在执行的时候暂停、在下一帧的时候恢复执行的位置。要启动coroutine，需要使用StartCoroutine函数。

```
void Update() {  
    if (Input.GetKeyDown( "f" )) {  
        StartCoroutine( "Fade" );  
    }  
}
```

默认的情况下，一个coroutine在它暂停后的下一帧恢复，但是也可以使用WaitForSeconds来引入一个延时。

```
IEnumerator Fade() {  
    for (float f = 1f; f <= 0; f -= 0.1f) {  
        Color c = renderer.material.color;  
        c.alpha = f;  
        renderer.material.color = c;  
        yield return new WaitForSeconds(.1f);  
    }  
}
```

这个可以用于产生一个随时间变化的效果，同时也是一个用于进行优化的执行，最常用的做法是将它们包含在Update函数中。但是Update函数

[快速回复](#)[我要收藏](#)[返回顶部](#)

要这么频繁的被调用的时候，可以把它放在一个coroutine中按一定时间进行调用，而不是每一帧都调用。一个这样的例子就是用于在游戏中如果有敌人接近的话就提示玩家，代码如下

```
function ProximityCheck() {
    for (int i = 0; i < enemies.Length; i++) {
        if (Vector3.Distance(transform.position, enemies[i].transform.position) < dangerDistance) {
            return true;
        }
    }
    return false;
}

IEnumerator DoCheck() {
    for(;;) {
        ProximityCheck;
        yield return new WaitForSeconds(.1f);
    }
}
```

当有很多敌人的时候，使用coroutine0.1秒执行一次靠近检查，可以减少大量的计算量。

   快速回复

 我要收藏



 返回顶部

顶

0

踩

1

-  上一篇
- [unityy注意问题](#)
-  下一篇
- [cocos2dx 基础知识体系](#)

我的同类文章

unity&cocos2dx (30)

Cocos2D-X设计模式:委托模... 2015-09-02 阅读 307

Cocos2D-X设计模式：观察... 2015-09-01 阅读 192

Cocos2D-X设计模式：防御... 2015-08-28 阅读 149

Cocos2D-X设计模式：管理... 2015-08-28 阅读 166

Cocos2D-X设计模式:单例模... 2015-08-28 阅读 355

cocos3.0+ shader 2015-08-24 阅读 246

Cocos2D-X设计模式:中介者.. 2015-09-01 阅读 188

Cocos2D-X设计模式: 组合... 2015-09-01 阅读 141

Cocos2D-X设计模式:外观模.. 2015-08-28 阅读 222

Cocos2D-X 设计模式：二段.. 2015-08-28 阅读 294

cocos3.x 优化提升渲染速度 2015-08-24 阅读 249

更多文章

小企业
促销月

购任意笔记本
+9元优惠购包鼠！

英特尔 广告

猜你在找

- unity5游戏开发Mecanim动画系统项目实战

C# For Unity系列之基础篇

C# For Unity系列之入门篇

unity3D一游戏/AR/VR在线就业班 C#入门（二）

Unity5游戏开发Mecanim高级特性项目实战
- cura 3d打印底座防止翘边

3D数学基础图形知识点

away 3d 小知识点

unity3D知识点

u3d fbx动画模型生成Ani

关闭



灌浆料

11/16/2010

