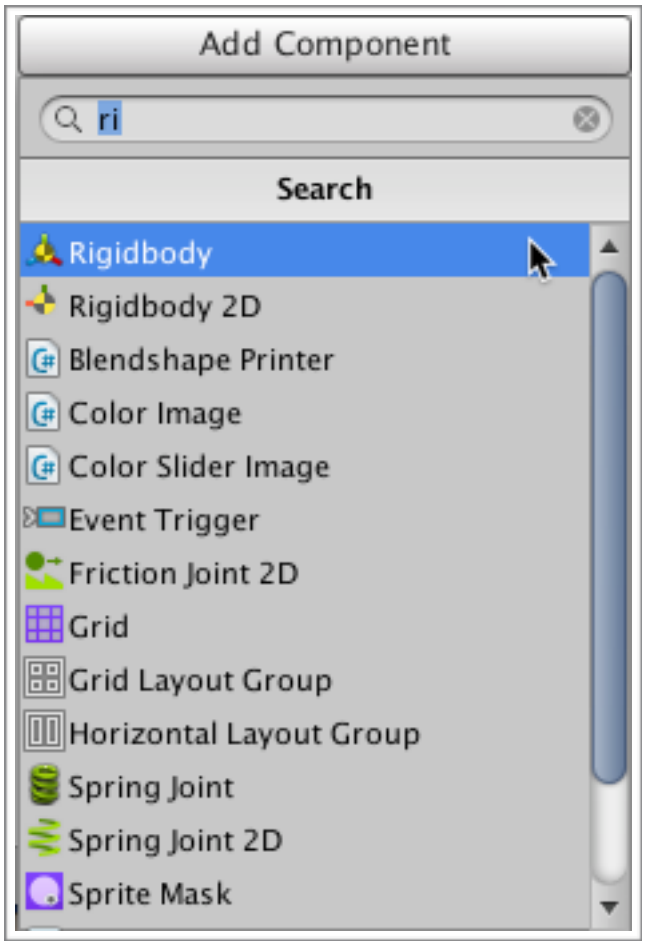
欢迎回到我们的学习。

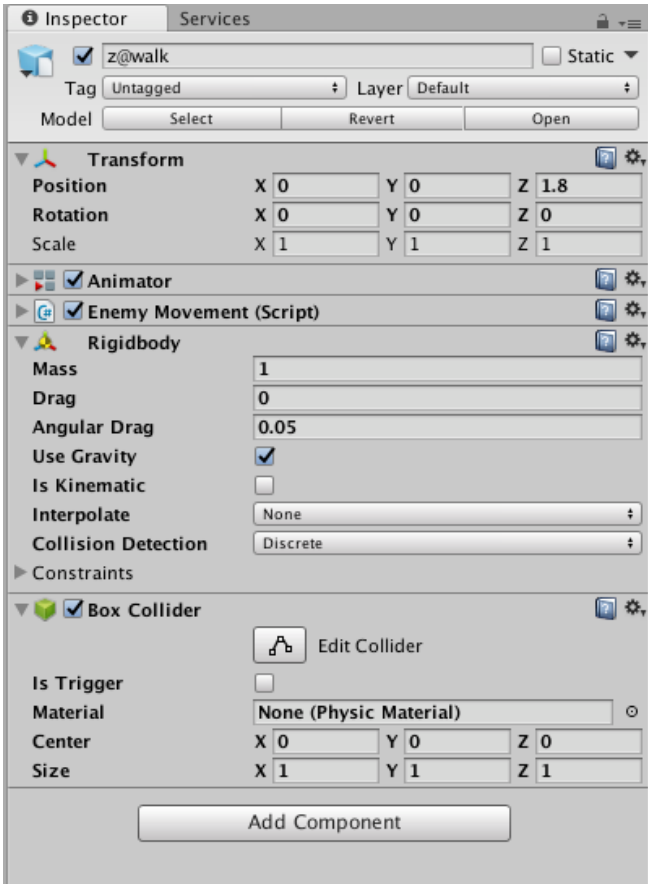
在这一课的内容中，我们将给僵尸敌人添加碰撞检测机制。

当敌人碰到Main Camera（也就是玩家）时，敌人将发起攻击。而为了实现这一点，我们需要添加碰撞检测。

在Unity编辑器的Hierarchy视图中选择z@walk游戏对象，然后在Inspector视图中点击Add Component，搜索Rigidbody,并将其添加到当前游戏对象上。

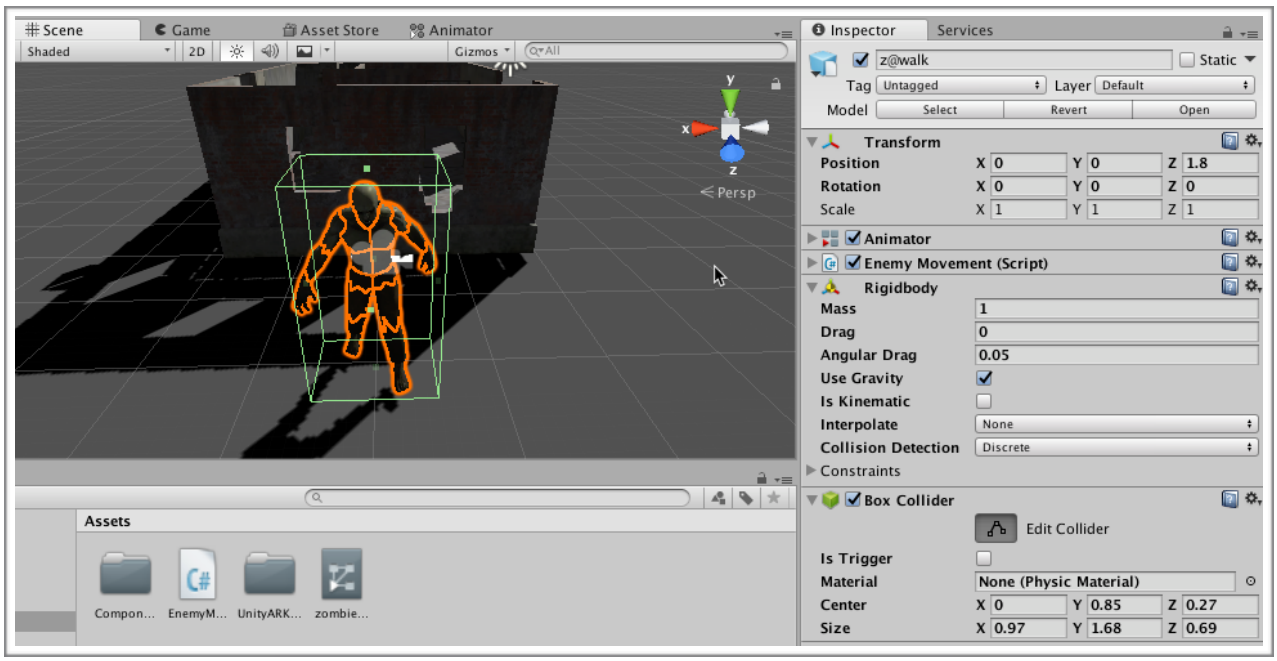


此时还需要再添加一个Box Collider组件，添加完成后的Inspector视图如下图：



在添加了Box Collider之后，可以看到僵尸敌人的周围出现一个绿色的盒子，接下来我们要设置Box Collider的大小。

点击Box Collider组件下面的Edit Collider，然后直接调整相关的大小，到觉得合适位置。

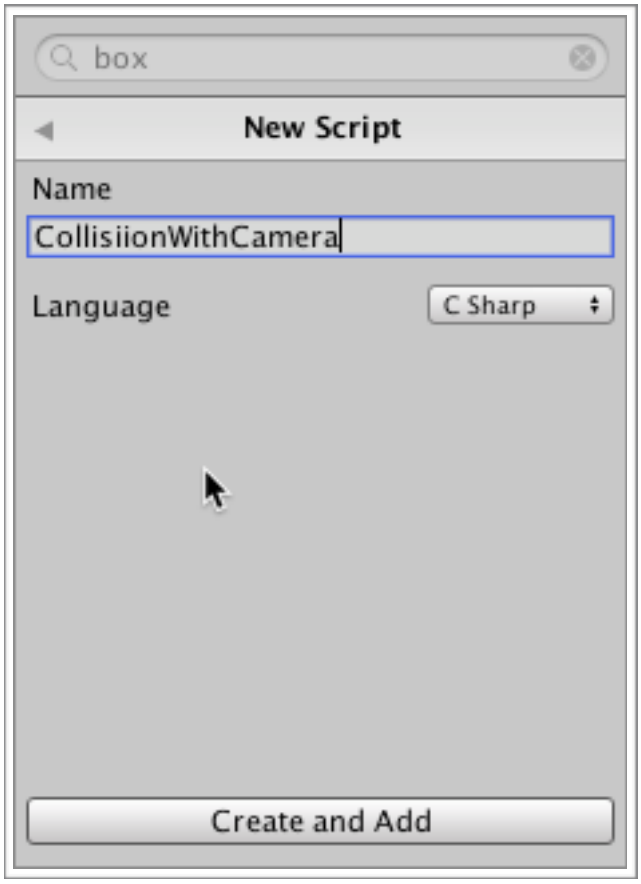


现在已经给敌人添加好了Box Collider，还需要给代表玩家的Main Camera添加碰撞机制。

在Hierarchy视图中选择Main Camera，然后点击Inspector视图中的Add Component，并添加一个Box Collider。

接下来我们还需要给敌人添加一个新的互动脚本，用来处理碰撞检测的相关事件。

在Hierarchy视图中选择z@walk游戏对象，然后点击Inspector视图中的Add Component，并添加一个新的脚本文件，将其命名为CollisionWithCamera。



我们将添加三个方法来实现碰撞检测。双击在MonoDevelop中打开该文件，并更改其代码如下：

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CollisionWithCamera : MonoBehaviour {
```

```
    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
    //碰撞开始

    void OnCollisionEnter (Collision col)
    {
        //判断碰撞体中是否有主摄像机
        if (col.gameObject.tag == "MainCamera") {

            Debug.Log ("enter");
        }
    }

    //碰撞结束
    void OnCollisionExit(Collision col){

        //判断碰撞体中是否有主摄像机
        if (col.gameObject.tag == "MainCamera") {

            Debug.Log ("exit");
        }

    }

    void Attack(){

    }
}
```

这里我们添加了三个方法，分别是OnCollisionEnter，OnCollisionExit，以及Attack方法，其作用如下：

（1）OnCollisionEnter

该方法用来获取碰撞开始事件，并作出相应的反应。

在这段代码中我们获取了碰撞体的tag标志，如果该标志和MainCamera一致，则表明碰撞体的另一方是主摄像机。

如果是，则输出结果”enter”

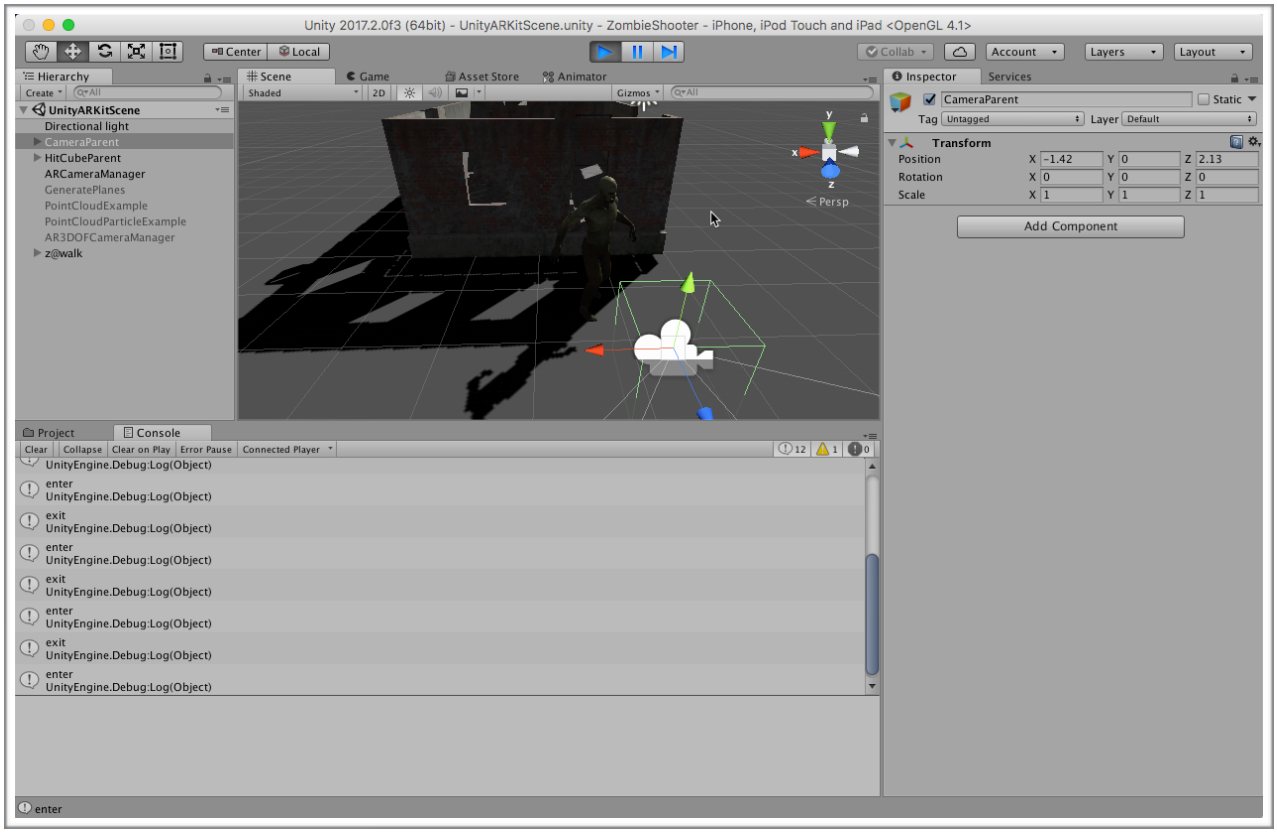
（2）OnCollisionExit
该方法用来获取碰撞结束事件，并作出相应的反应。
在这段代码中我们获取了碰撞体的tag标志，如果该标志和MainCamera一致，则表明碰撞体的另一方是主摄像机。
如果是，则输出结果“exit”。

（3）Attack
该方法将在后面添加敌人的攻击行为。

注意：
以上的OnCollisionEnter和OnCollisionExit方法的首字母必须是大写，否则无法生效。

回到Unity的主编辑器，点击Play按钮来预览效果。

现在敌人将朝着主摄像机的方向行走，拖动主摄像机，当其和敌人碰在一起时，在Unity编辑器下方的Console视图中将输出enter，当碰撞结束时，将输出exit。



接下来让我们定义几个变量。
然后更改OnCollisionEnter和OnCollisionExit方法中的输出结果代码。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CollisionWithCamera : MonoBehaviour {

	//1.敌人是否在场
	public bool zombieIsThere;
	//2.计时器
	float timer;
	//3.两次攻击之间的间隔
	int timeBetweenAttack;

	// Use this for initialization
	void Start () {

		//4.定义初始数值
		timeBetweenAttack = 2;
	}

	// Update is called once per frame
	void Update () {

		//5.获取系统时间
		timer += Time.deltaTime;
//		print (timer);

		//6.判断敌人是否在场，而且攻击间隔大于2秒
		if (zombieIsThere && timer >= timeBetweenAttack) {

			//7.开始攻击动作
			Attack ();
		}
	}
	//碰撞开始

	void OnCollisionEnter (Collision col)
	{
		//判断碰撞体中是否有主摄像机
```

```
            if (col.gameObject.tag == "MainCamera") {

//              Debug.Log ("enter");
                //8.确认敌人在现场
                zombieIsThere = true;
            }
        }

        //碰撞结束
        void OnCollisionExit(Collision col){

            //判断碰撞体中是否有主摄像机
            if (col.gameObject.tag == "MainCamera") {

//              Debug.Log ("exit");
                //9.设置敌人不在现场
                zombieIsThere = false;
            }

        }

        //攻击指令
        void Attack(){

            //10.恢复计时器为0
            timer = 0;
            //11.输出结果
            Debug.Log ("attack");
        }
}
```

下面按照数字编号来解释一下相关的代码：

（1）`public bool zombieIsThere;`

这里我们定义了一个public bool类型的变量，public表示其它类也可以访问这个变量，而bool则是变量的具体类型。

所谓的bool类型是一种逻辑判断类型，它只有两个数值，true或者false，代表“是”与“否”。

（2）`float timer;`

这里定义了一个float类型的变量，用来保存系统时间。

float是一种数据类型，用来保存浮点类的数据

(3)`int timeBetweenAttack;`

这里定义了一个int类型的变量，用来保存两次攻击之间的时间间隔。

（4）在Start方法中，我们定义了timeBetweenAttack的初始值。
Start()是Unity提供的一种事件函数，在场景启动的时候就会自动执行其中的代码。

（5）我们使用Time.deltaTime来获取系统时间

（6）这里使用了逻辑判断，&&是逻辑与的意思，表示符号左右的两个条件必须同时满足，才能执行后面的动作。

（7）调用攻击动作的方法。

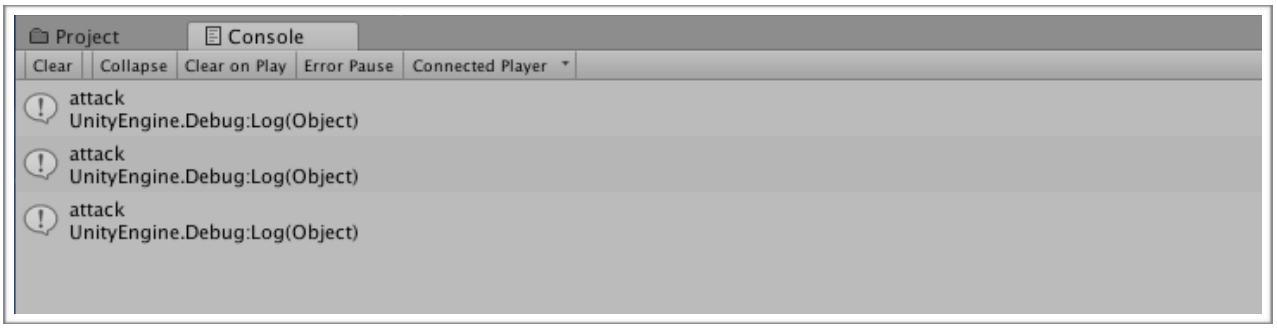
（8）当碰撞开始时，把zombieIsThere这个bool类型的变量数值设置为true

（9）当碰撞结束时，把zombieIsThere这个bool类型的变量数值设置为false

（10）当攻击开始时，恢复计时器为0.

（11）输出结果，表示攻击开始。

接下来回到Unity的主编辑器，点击Play来预览游戏效果。
当敌人开始攻击时，可以看到Console视图中出现了attack提示。



这样，我们已经验证了游戏的基本逻辑。
接下来，我们让敌人在攻击时播放攻击的动画，而不仅仅在console中输出一个提示。

更改Attack方法的代码如下：

```
    //攻击指令
    void Attack(){
```

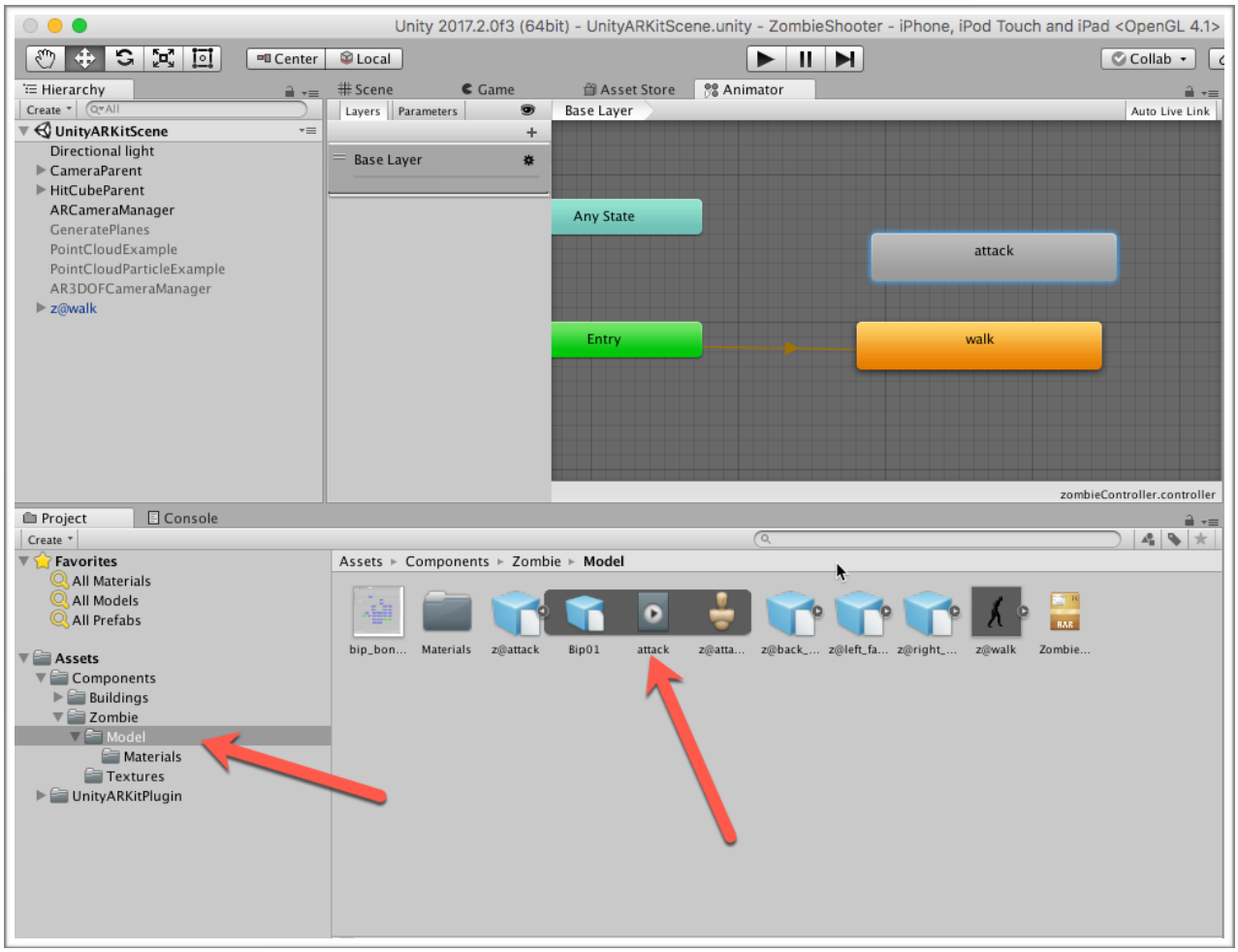
```
            //10.恢复计时器为0
            timer = 0;
            //11.输出结果
//          Debug.Log ("attack");

            //12.播放攻击动画
            GetComponent<Animator>().Play("attack");
        }
```

在编号12的代码中，我们使用GetComponent<Animator>函数获取了当前游戏对象的Animator组件，然后让其播放attack动画。

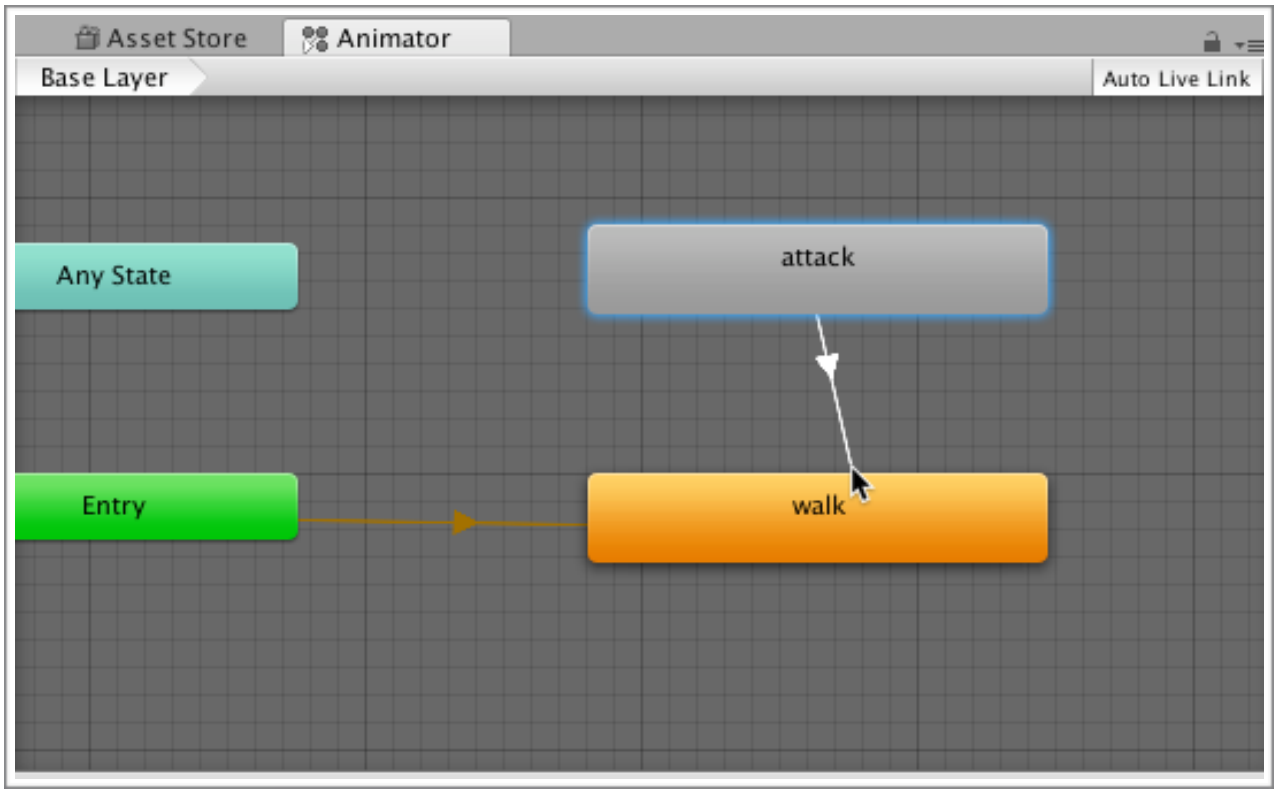
问题来了，我们在设计敌人的Animator动画控制器时，并没有添加attack动画。

因此让我们回到Unity编辑器，在Project视图中找到zombieController,双击打开Animator视图。

从Project视图中找到Assets-Components-Zombie-Model中的z@attack预设体，展开右三角，会看到里面有一个attack动画片段，将其拖动到Animator视图中。



右键单击attack动画片段，选择Make Transition，然后拖一条线到walk动画片段上。



因为我们希望当攻击动画播放结束时，僵尸敌人会回到walk这个动作状态上来。

在Unity编辑器中点击Play按钮，预览游戏效果，会看到如我们所设想的那样，当敌人碰到主摄像机并满足条件时，会开始播放攻击动画。而当我们移走主摄像机时，敌人又回到正常的行走动画。

好了，今天的学习到此结束。