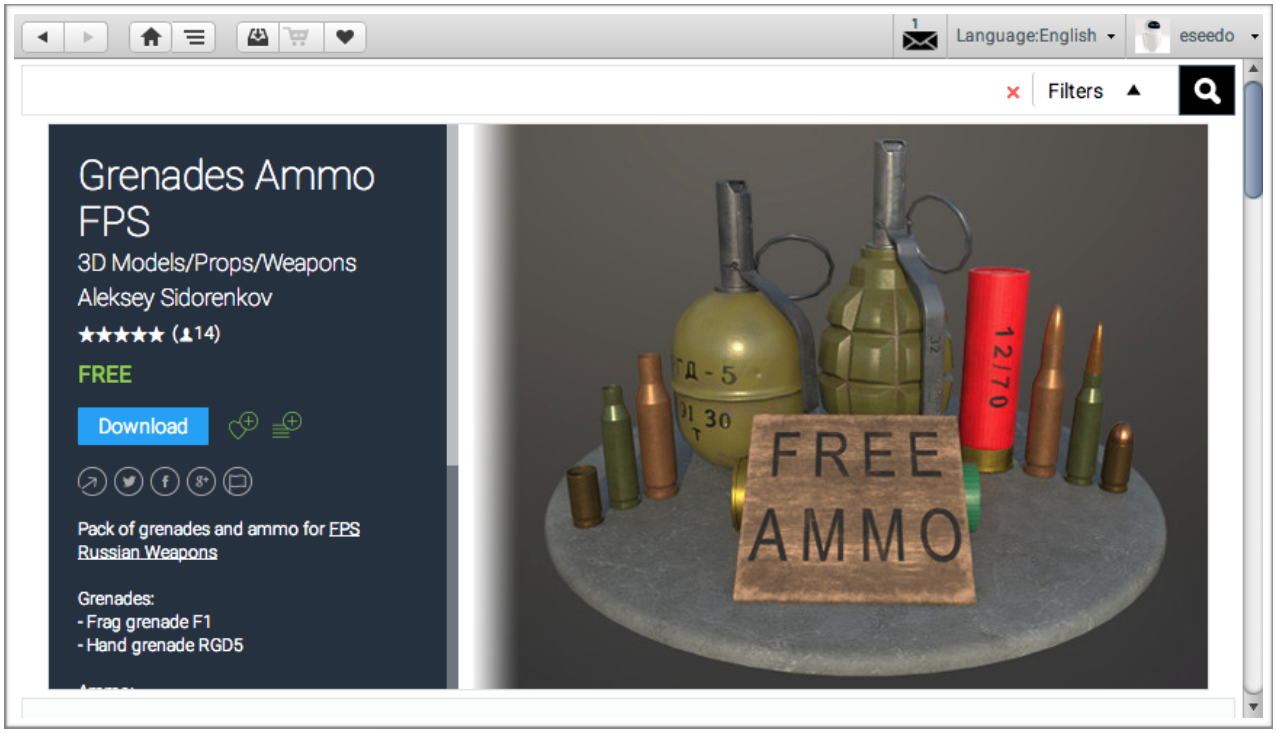
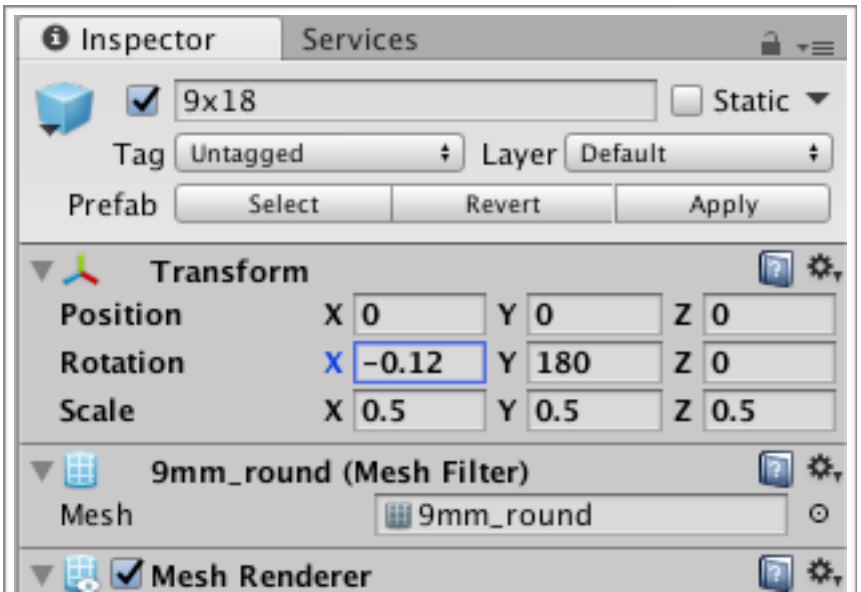
为了让游戏的效果更加接近真实，我们希望当手机射击的时候可以弹出弹壳。
在本课的内容中，我们将主要实现这一功能。

打开Unity，切换到Asset Store视图，在搜索栏中输入shell,选择FREE ONLY，然后下载并导入下图中的资源。



下载完成后，把Grenades_Ammo_FPS文件夹拖动到Project视图的Assets/Arts中。
从Grenades_AMMO_FPS的Prefabs文件夹中找到9x18这个预设体，在Inspector视图中，右键单击Transform，选择reset。调整其Scale比例，并设置Rotation值。



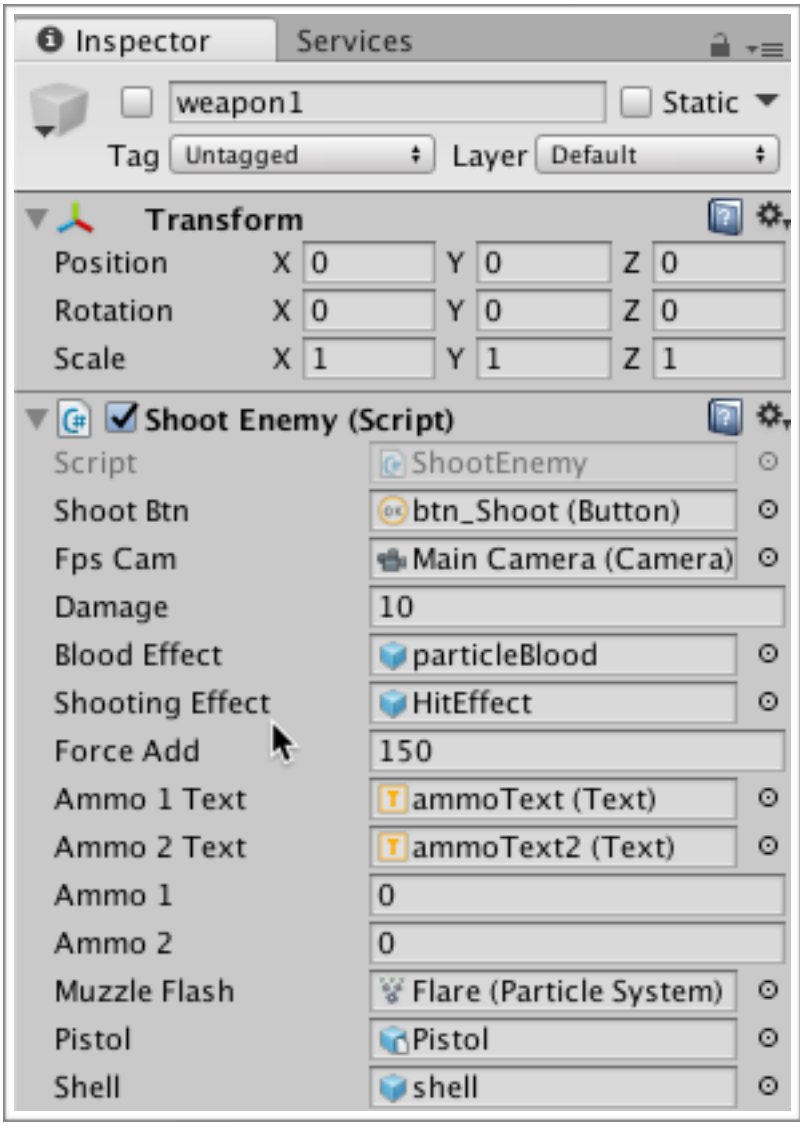
将其更命名为shell,然后拖动到Project视图的Assets/_Prefabs文件夹中。

然后从Project视图中找到并打开ShootEnemy.cs。
在Start方法之前添加一行代码：

```
    //创建到弹壳的引用
    public GameObject shell;
```

这里我们创建了到弹壳对象的引用。

回到Unity编辑器，在Hierarchy视图中选中CameraParent-Main Camera-weapon1对象，然后在Inspector视图中将Shoot Enemy组件中的Shell属性设置为刚才创建的shell 预设体，如图所示。

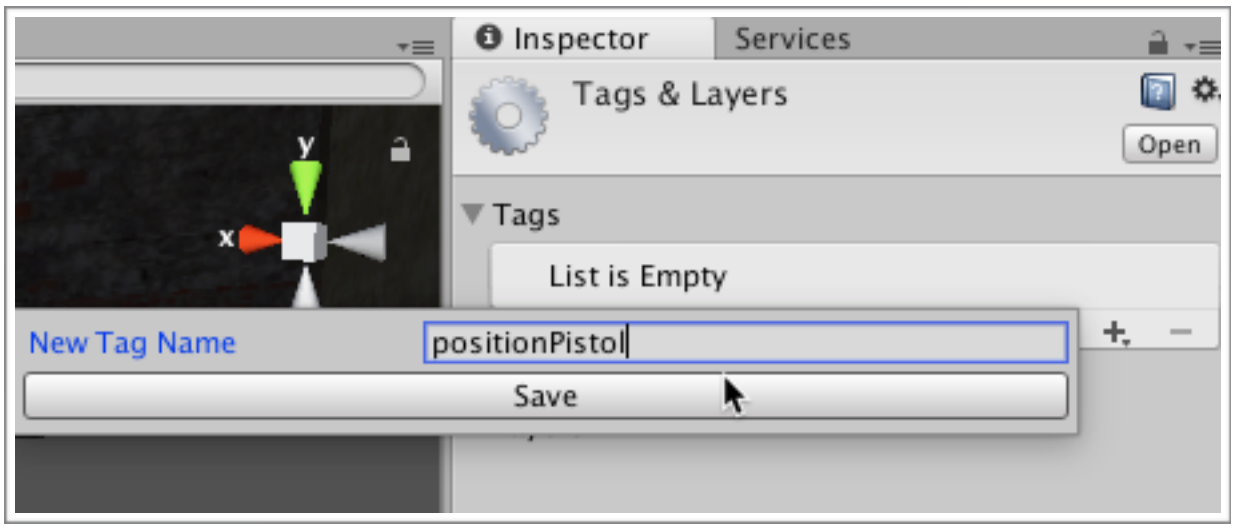


然后在OnShoot方法中，紧接着播放开火动画的那行代码添加以下代码：

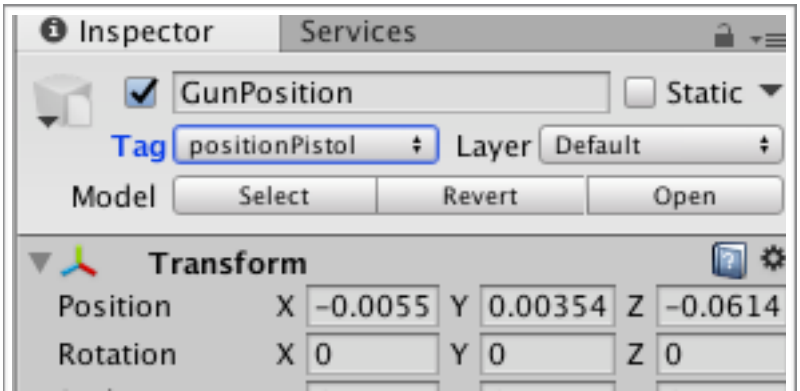
```
                //loading shell

                Vector3 position =
GameObject.FindGameObjectWithTag
("positionPistol").transform.position;
                Quaternion rotation = Quaternion.Euler (0, 0, 0);

                Instantiate(shell,position,rotation);
```

这里我们获取了Tag标记为positionPistol的对象的位置，然后将rotation设置为0，并使用Instantiate方法生成弹壳对象。

接下来我们要设置这个Tag。在Hierarchy视图中选中CameraParent-Main Camera-weapon1-Pistol-All-GunAndRightArm-GunPosition，然后在Inspector视图中添加一个Tag。



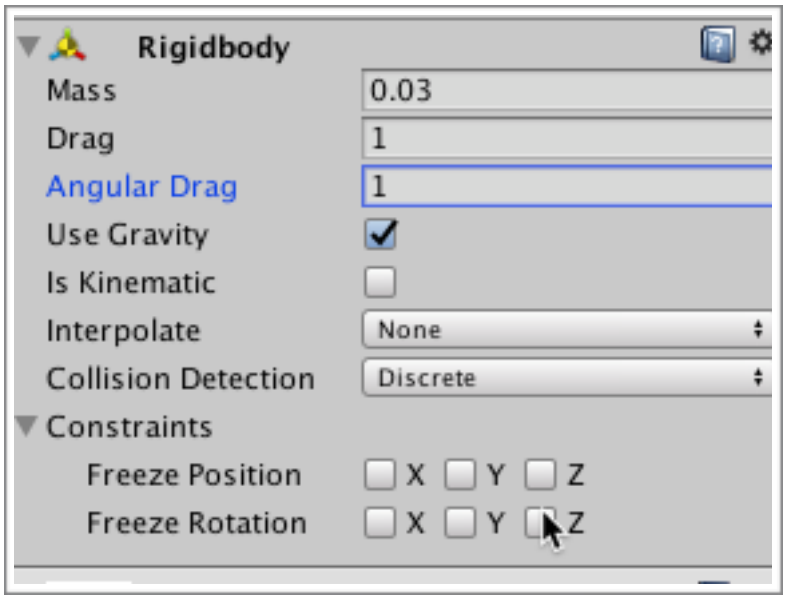
创建完成后，将GunPostion的Tag设置为positionPistol。



好了，接下来点击Unity编辑器工具栏上的Play按钮预览游戏效果。
可以看到在Hierarchy视图中子弹壳已经生成了，但是在Game视图中还无法看到子弹壳的实体。这是因为它的位置在手枪的扳机处，而且保持不变。
为此，我们需要给弹壳添加物理机制。

从Project视图中的Assets/_Prefabs文件夹中找到shell这个预设体，然后在Inspector视图中做一些调整。

首先给它添加一个Rigidbody组件，然后将Mass设置为0.03，Drag设置为1，Angular Drag设置为1.

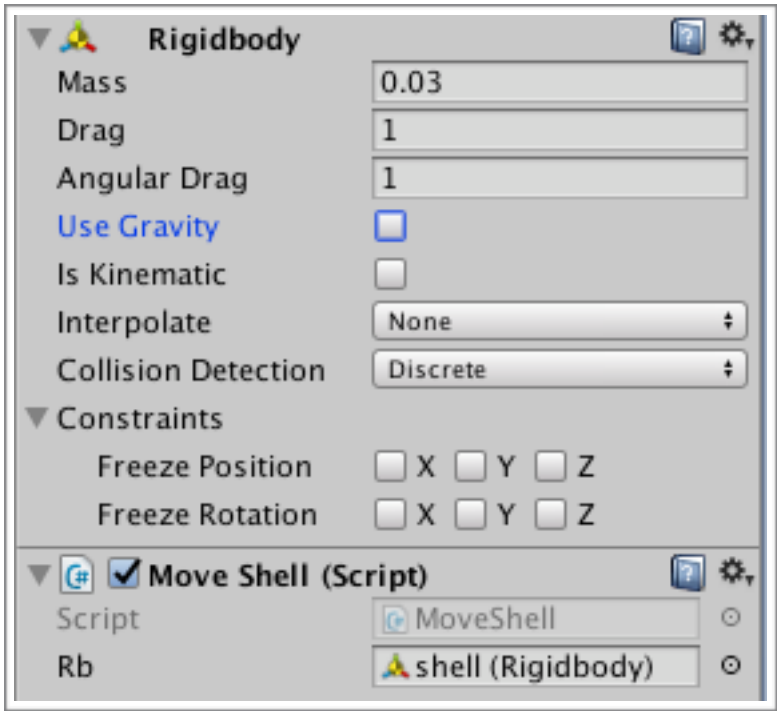


紧接着添加一个新的脚本组件，名为MoveShell.cs，在MonoDevelop中打开并编辑。

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

public class MoveShell : MonoBehaviour {

	//创建到Rigidbody的引用
	public Rigidbody rb;

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

```
            rb = GetComponent<Rigidbody> ();
        }

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

            //施加一个向右的力
            rb.AddForce (transform.right * 0.05f);
            //施加一个向上的力
            rb.AddForce (transform.up * 0.05f);
        }
}
```

回到Unity编辑器，在Inspector视图中将Move Shell脚本组件中的Rb属性设置为Rigidbody，同时禁用Use Gravity，如图。



点击Unity编辑器上的Play按钮预览游戏，弹壳出来了，但是弹出的方向始终在一个方向，显然不符合常识。我们希望子弹有个随机的掉落方向，接下来将实现这一点。继续编辑MoveShell.cs脚本如下：

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

public class MoveShell : MonoBehaviour {

	//创建到Rigidbody的引用
	public Rigidbody rb;

	// Use this for initialization
	void Start () {
		//获取Rigidbody
		rb = GetComponent<Rigidbody> ();

		//1.添加随机旋转的协程
		StartCoroutine("Rotate");

		//2.添加恢复重力的协程
		StartCoroutine("RecoverGravity");
	}

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

		//施加一个向右的力
		rb.AddForce (transform.right * 0.05f);
		//施加一个向上的力
		rb.AddForce (transform.up * 0.05f);
	}

	IEnumerator Rotate(){
		while (true) {

			//3.等待0.1秒
			yield return new WaitForSeconds (0.1f);
			//4.随机旋转
			transform.eulerAngles += new Vector3
(Random.Range (-360f, 360f), Random.Range (-360f,
360f),Random.Range(-360f,360f));

		}

	}
```

```
    IEnumerator RecoverGravity(){

        //5.等待0.2秒
        yield return new WaitForSeconds (0.2f);
        //6.恢复重力的影响
        rb.useGravity = true;
    }
}
```

按照注释行的数字编号简单解释一下：
1.添加了一个随机旋转的协程
2.添加了恢复重力作用的协程
3.等待0.1秒
4.在0.1秒后开始随机旋转
5.等待0.2秒
6.在0.2秒后开始恢复重力的作用

回到Unity编辑器，点击Play,可以预览下游戏效果。

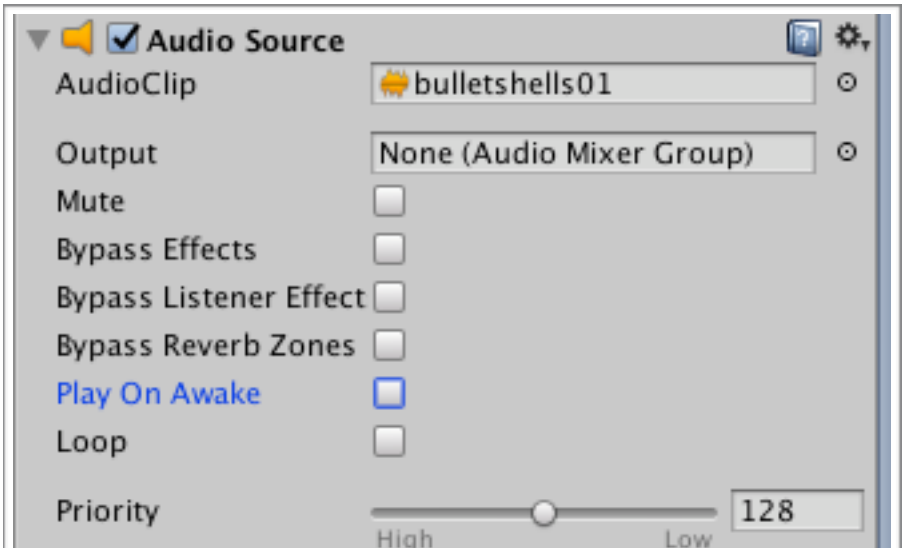
接下来还有一个事情需要完成，那就是销毁弹壳，不然游戏场景中的弹壳会无穷无尽了。

继续回到刚才的MoveShell.cs脚本，在Start方法的最后添加一行代码：

```
        //3.在2秒后销毁弹壳
        Destroy(gameObject,2.0f);
```

最后给弹壳也添加一个音效。

在Project视图中选中shell预设体，然后点击Inspector视图中的Add Component，添加一个Audio Source组件。禁用Play On Awake，将AudioClip属性设置为bulletshells01,如图所示。



最后回到MoveShell.cs，在RecoverGravity方法的最后添加以下代码：

```
            //等待0.2秒
            yield return new WaitForSeconds(0.2f);

            //播放弹壳的音效
            AudioSource shell = GetComponent<AudioSource>();
            shell.Play ();
```

代码的作用很直白，首先要再等待0.2秒，然后播放弹壳的音效。

好了，本课的内容就到此结束了，我们下一课再见~