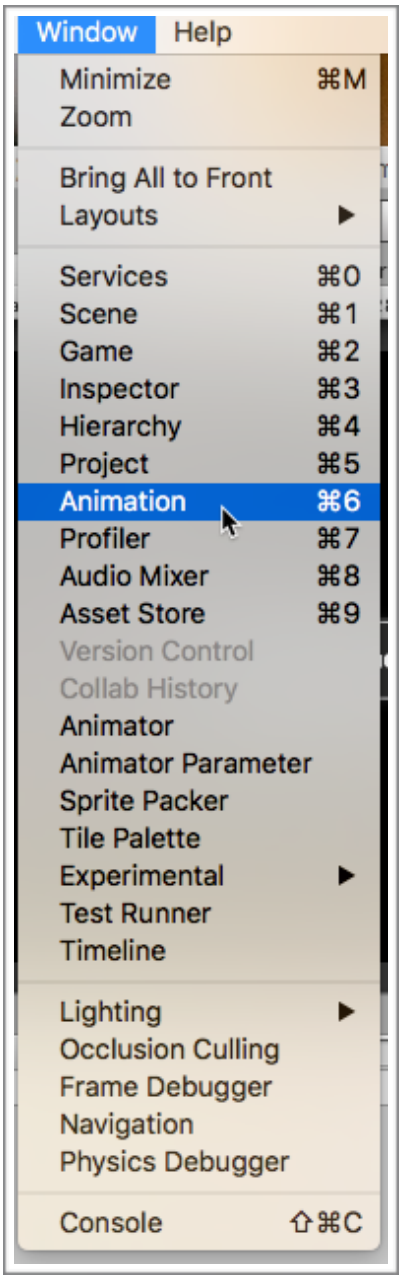
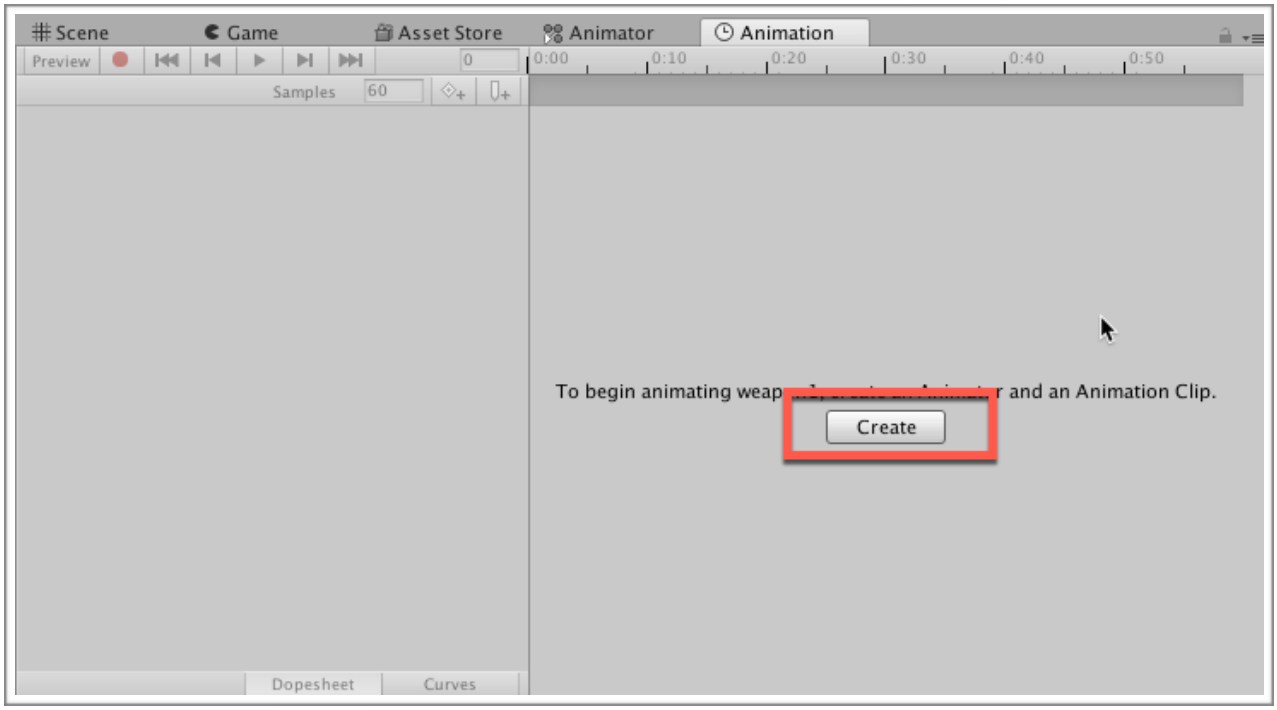
在这一课的内容中，我们将给武器添加idle、开火和装弹时的动画，让画面显得更为真实。
首先我们要手动创建一个idle动画。在Hierarchy视图中选中Pistol对象，对于idle动画，我们只需要更改Transform中的Position Y即可。

在Unity编辑器的菜单栏中点击Window-Animation,打开动画编辑器。

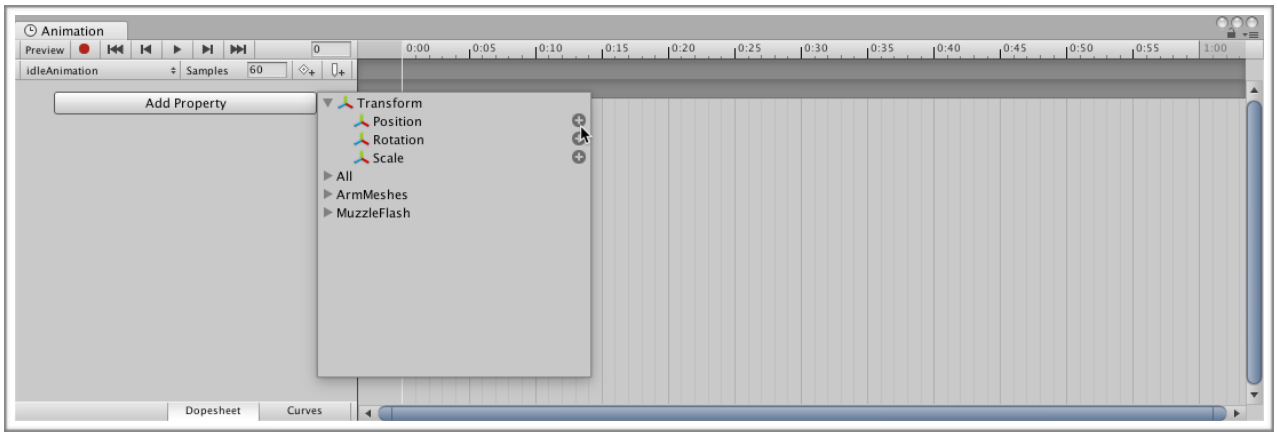


点击Create以创建一个新的Animation，并将其命名为idleAnimation。

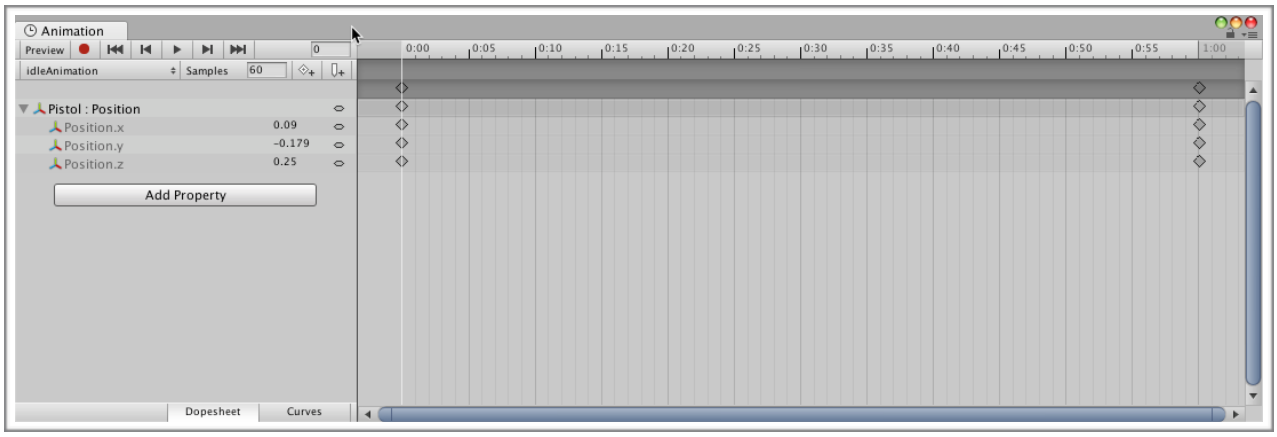


需要注意的是，在点击Create之前，需要在Hierarchy视图中选中Pistol对象。

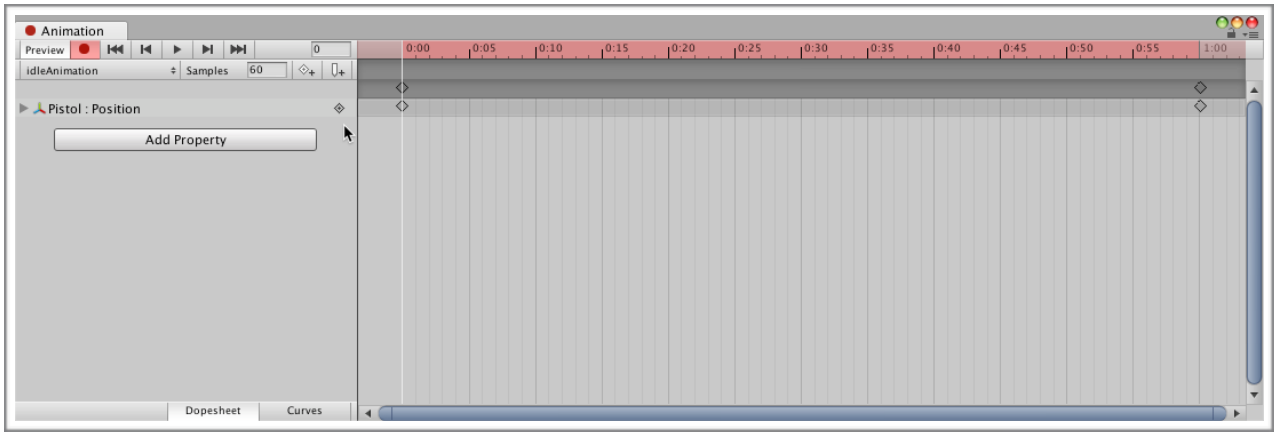
接下来我们只需要设置几个关键帧即可。
首先点击Animation面板左侧的Add Property按钮，并选择Position，如图所示。



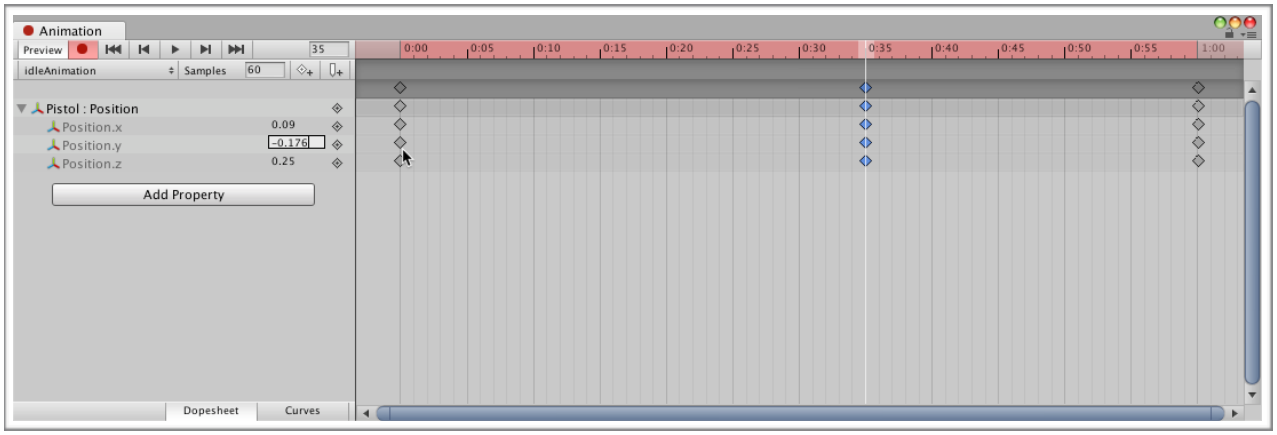
添加完成后，展开Pistol:Position,如下图所示：



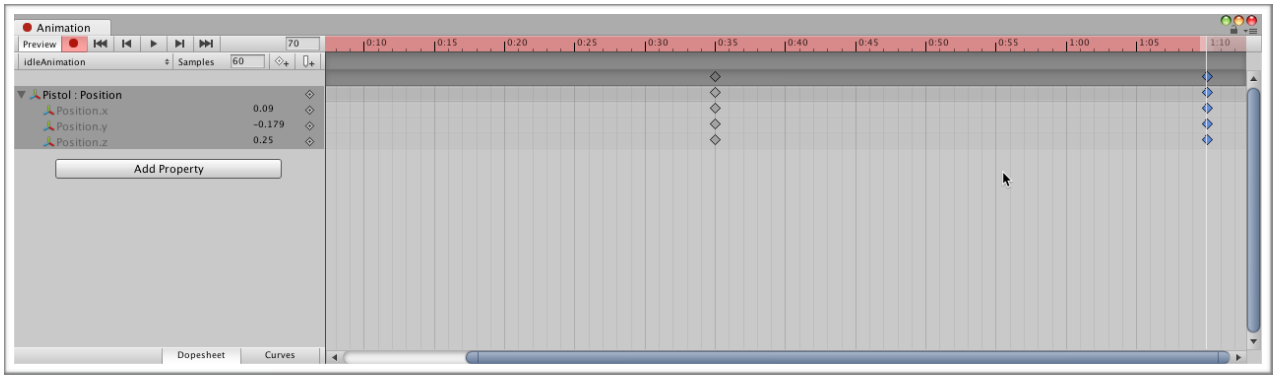
然后点击Animation面板左上的红色录制按钮开始录制动画。



在面板中间的时间轴的大概0.35秒处点击，然后点击左侧的Add Keyframe按钮添加一个关键帧，然后将Position.y的数值更改为-0.176,如图所示。



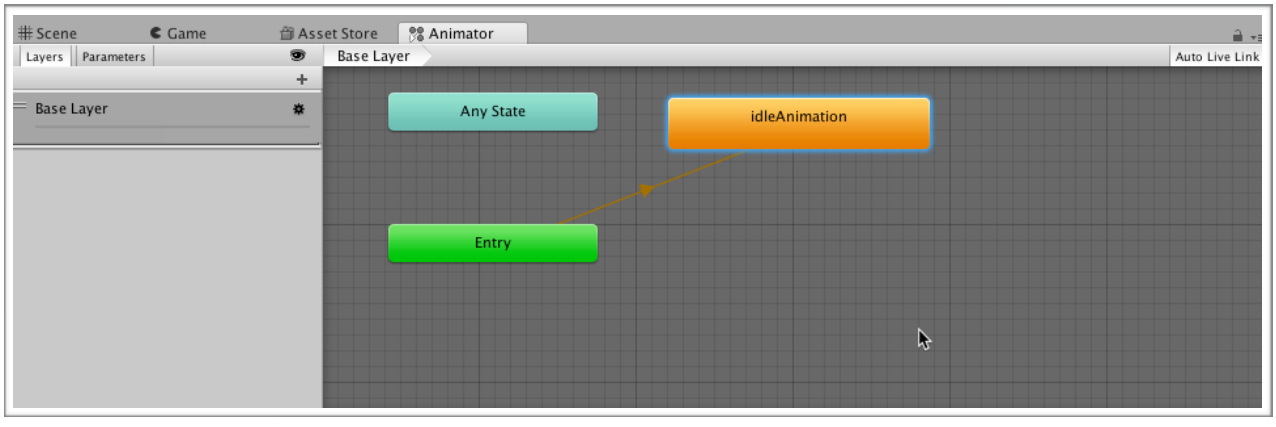
最后将时间轴上另外一个关键帧拖动到1.10秒左右的位置，然后将Position.y的数值更改为-0.179,如图所示。



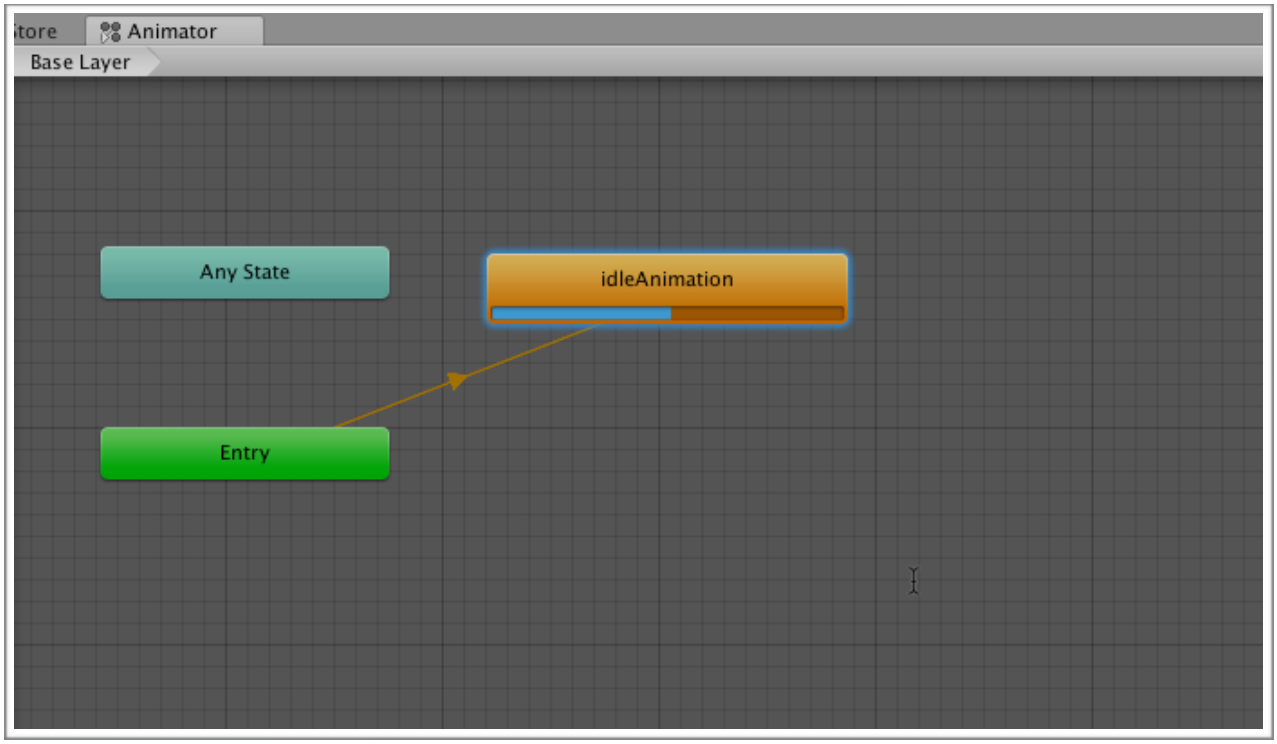
然后点击Animation面板工具栏上的播放按钮预览动画效果。如果觉得还不错，那么可以点击红色录制按钮停止录制。

好了，这样我们的idleAnimation动画就录制完毕了。

接下来在Inspector视图中打开Pistol的Animator Controller，此时在Animator视图中只有一个idleAnimation状态，如图。



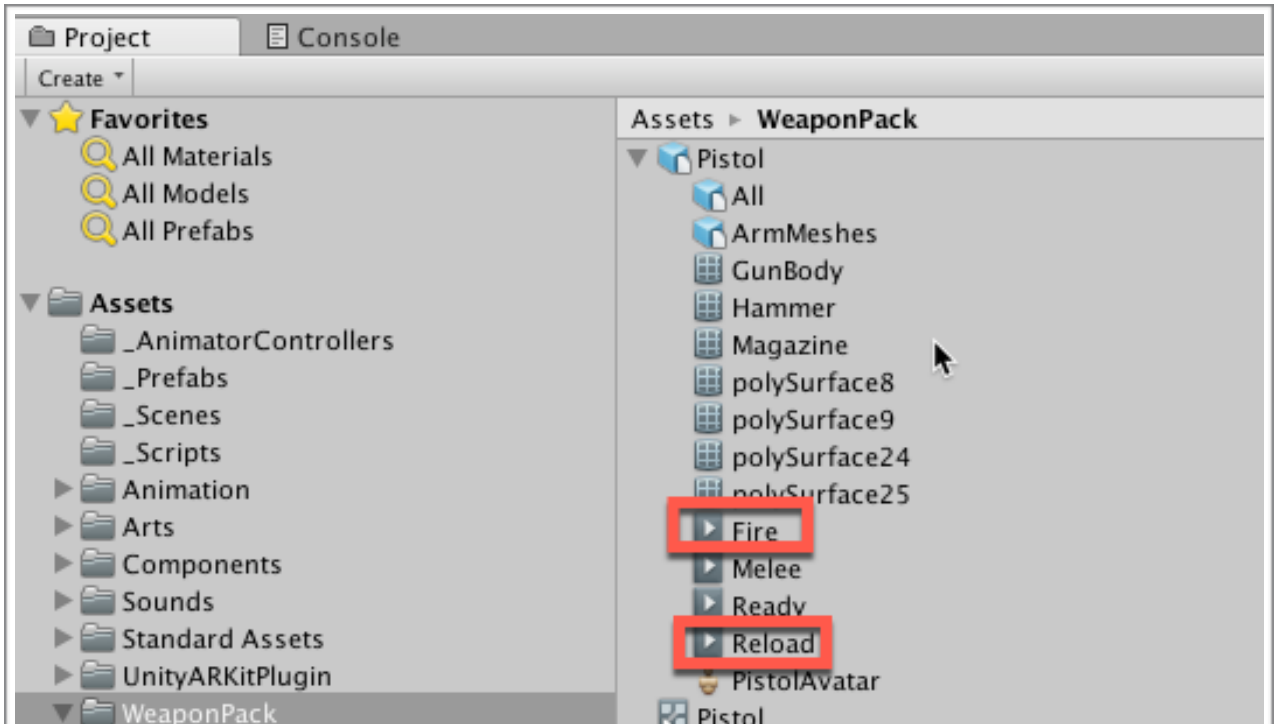
此时点击工具栏上的Play按钮，可以看到动画在播放中。



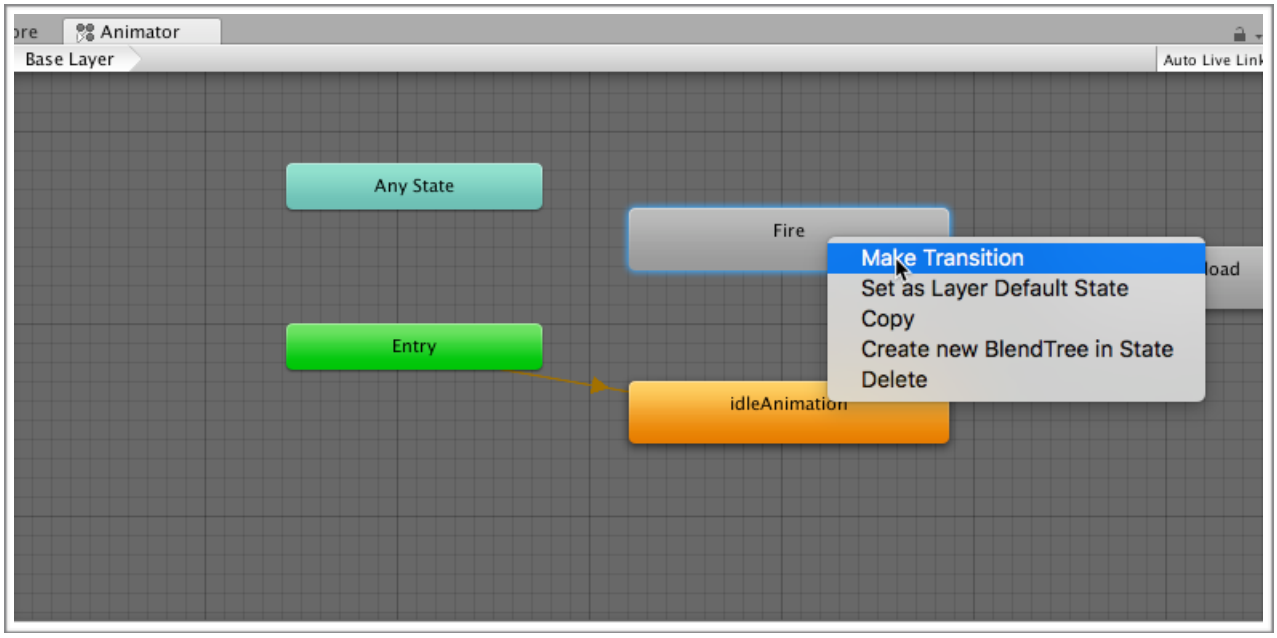
但是如果触碰开火按钮，会发现仍然播放的是idle动画。

因此接下来我们将添加开火的动画状态和重新装弹的动画状态。

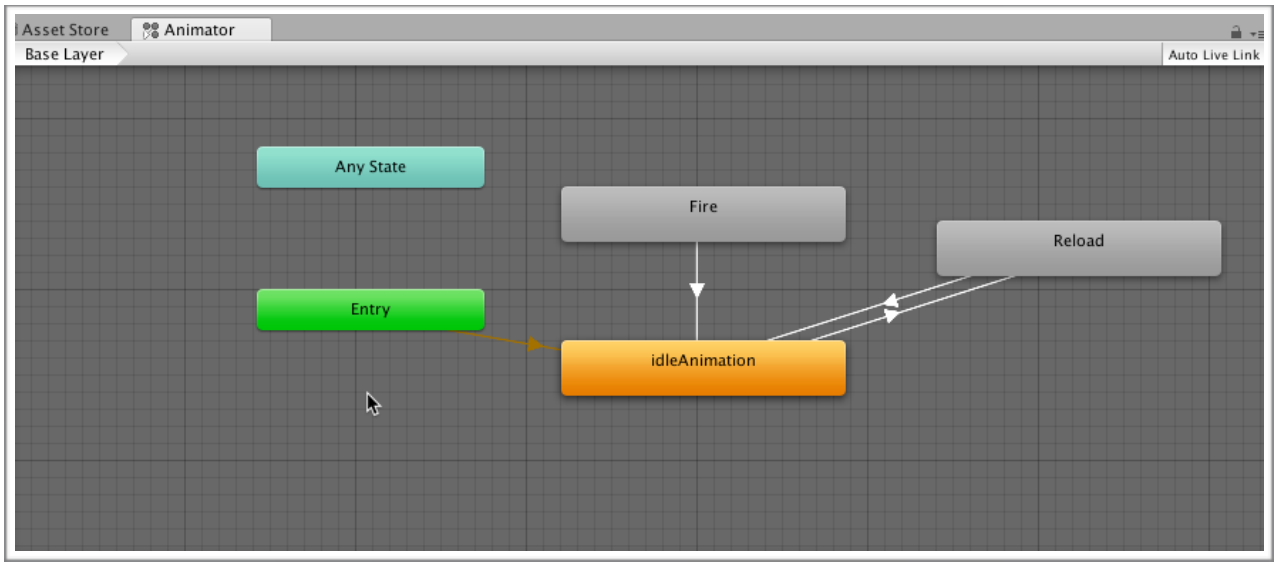
在Unity编辑器的Project视图中找到WeaponPack中的Pistol预设体并将其展开，发现里面已经提供了Fire动画和Reload动画，如图所示。



将Fire动画和Reload动画拖动到Animator视图中。
右键单击Fire动画，选择Make Transition，并拖出一条线到idleAnimation.



接下来选中idleAnimation,使用同样的方式拖一条线到Reload动画，不过此时我们还要从Reload动画托一条线到idleAnimation。
完成后的连线如下图所示。



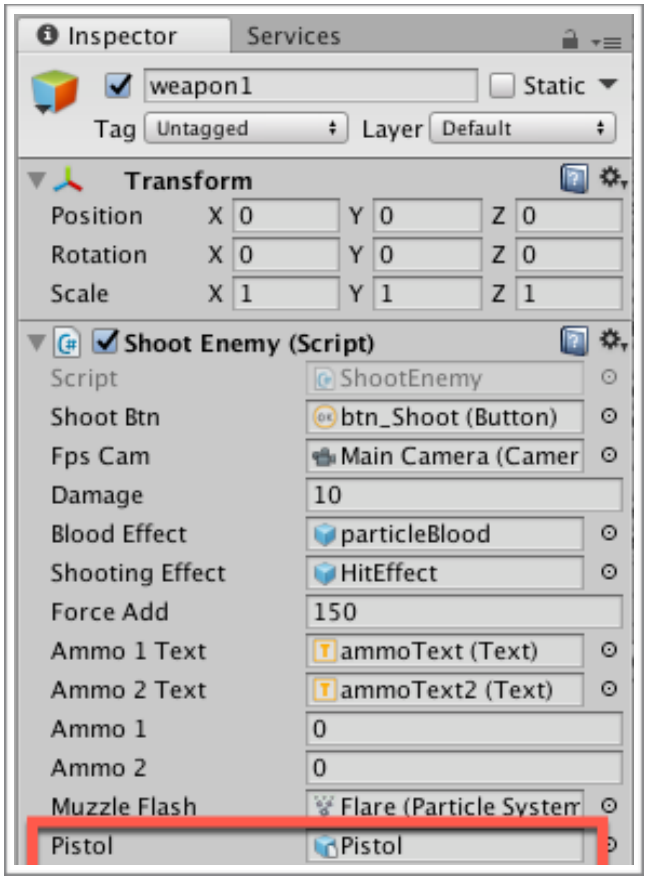
好了，Animator的设置基本上完成了，待会儿我们会回来添加一些其它的设置。

接下来进入代码时间，在Project视图中的Assets-_Scripts中找到并打开ShootEnemy.cs脚本文件。

在Start方法的前面添加以下代码：

```
//创建到手机对象的引用
public GameObject pistol;
```

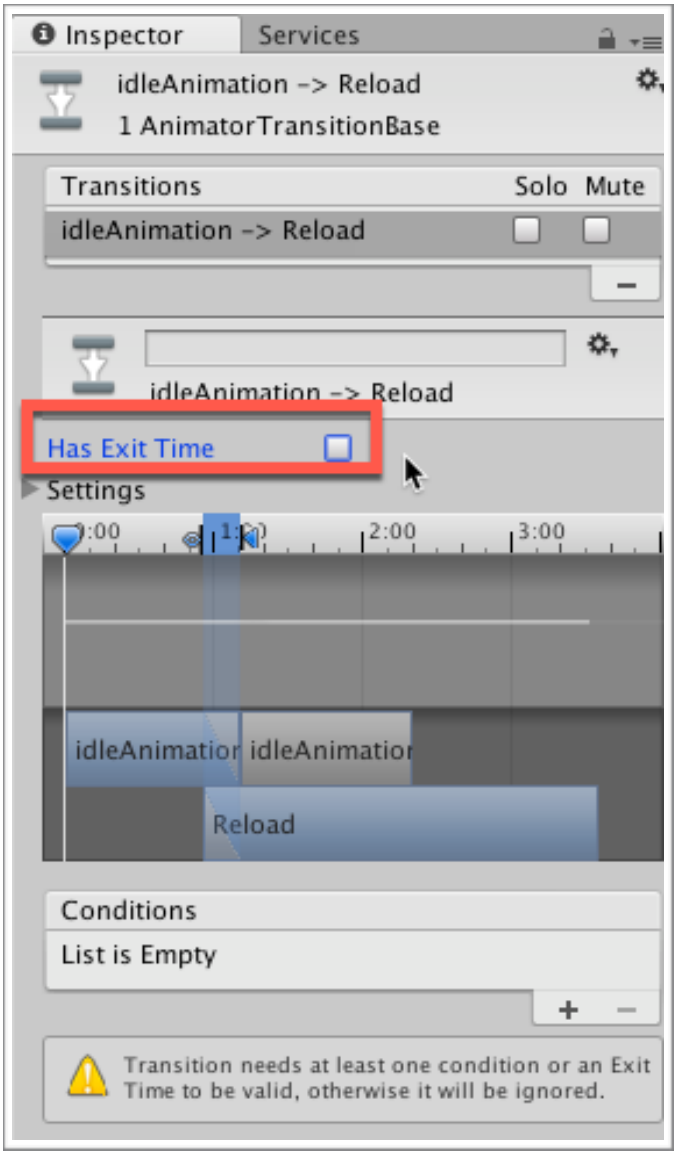
然后回到Unity编辑器，选中CameraParent-Main Camera- weapon1对象，然后在Inspector视图中设置Shoot Enemy组件的Pistol属性为Pistol对象。



然后回到ShootEnemy.cs脚本，在OnShoot方法的最后，紧接着muzzleFlash.Play();添加一行代码如下：

```
        //播放开火动画
                pistol.GetComponent<Animator>().Play("Fire");
```

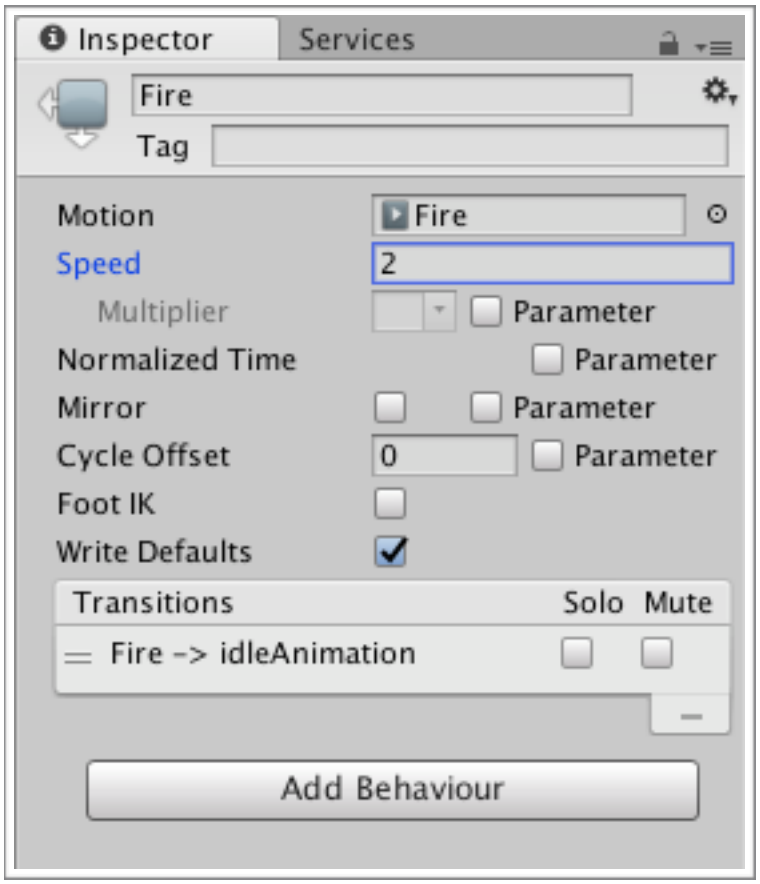
接下来回到Animator视图，选中从idleAnimation到Reload动画状态之间的连线，然后取消勾选Has Exit Time，如图所示。



然后点击Unity编辑器工具栏上的Play按钮，来预览下游戏效果：

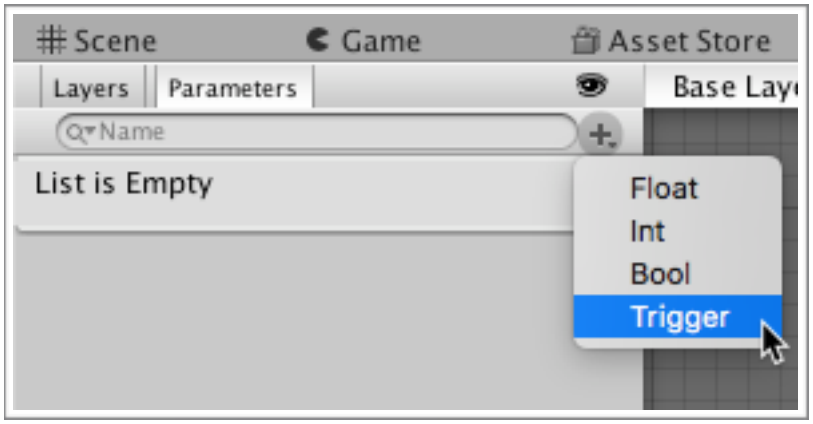
开火动画起作用了，但是似乎稍微有点慢。

回到Animator视图，选中Fire动画，将Inspector视图中的Speed更改为2.

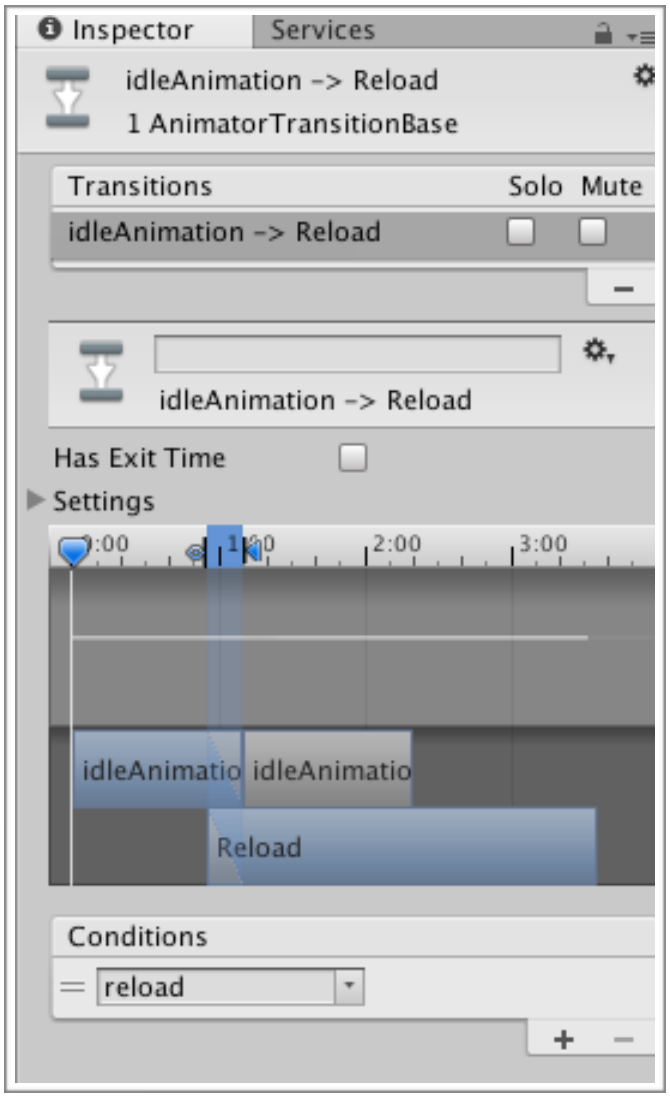


再次预览，发现效果好一些了。

接下来让我们添加重新装弹的Reload动画。
首先回到Animator视图，在左侧切换到Parameters选项卡，点击+号，选择Trigger类型，将其命名为reload。



然后选中从idleAnimation到Reload的连线，在Inspector视图中的Conditions处点击+号，从而创建一个新的触发条件reload,如图所示。



Trigger类型的参数触发条件意味着，仅当reload条件满足时，才会播放相关的动画。

接下来回到ShootEnemy.cs，并更改其中的代码如下：

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//import namespace
using UnityEngine.UI;
```

```
public class ShootEnemy : MonoBehaviour {

    //创建到Button对象的引用
    public Button shootBtn;
    //创建到主摄像机的引用
    public Camera fpsCam;

    //设置敌人每次受到伤害的数值
    public float damage = 10f;

    //敌人受伤的粒子特效
    public GameObject bloodEffect;

    //攻击的粒子特效

    public GameObject shootingEffect;

    //添加的攻击力度
    public int forceAdd = 300;

    //定义两个音源对象
    AudioSource shootSound;
    AudioSource reloadSound;


    //创建到弹药UI元素的引用
    public Text ammo1Text;
    public Text ammo2Text;
    public int ammo1;
    public int ammo2;

    //判断弹药是否已空
    private bool ammoIsEmpty;

    //创建到开火粒子系统的引用
    public ParticleSystem muzzleFlash;


    //创建到手机对象的引用
    public GameObject pistol;

    //1.重新装弹检查判断
    private bool reloadCheck;
```

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


        //Debug.Log ("Activated!");
        //添加按钮的响应事件
        shootBtn.onClick.AddListener (OnShoot);

        //获取音源组件
        AudioSource[] audios = GetComponents<AudioSource>();

        //设置音源
        shootSound = audios [0];
        reloadSound = audios [1];

        //设置弹药数量的初始值
        ammo1 = 20;
        ammo2 = 100;

        //2.设置重新装弹检查的默认值为true

        reloadCheck = true;
    }

    //3.Coroutine方法
    IEnumerator WaitForReload(){

        yield return new WaitForSeconds (3f);
        reloadCheck = true;


    }

    public void OnShoot(){

        //4.仅在ammoIsEmpty和reloadCheck同时为真时才可执行逻辑判断中
的操作

        if (!ammoIsEmpty && reloadCheck) {

            if (ammo1 == 1) {
```

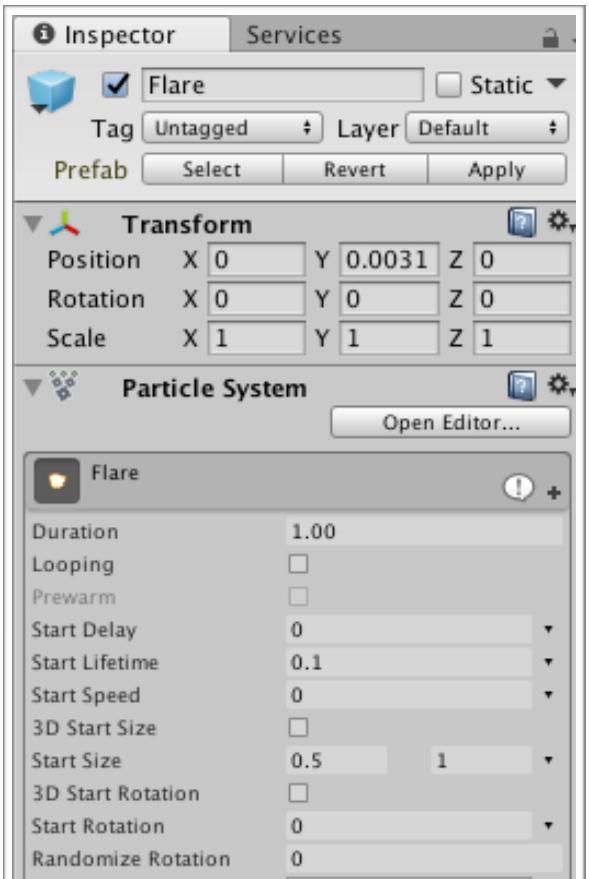
```
                ammo1 = 21;

                //5.触发重新装弹动画
                pistol.GetComponent<Animator> ().SetTrigger
("reload");
                reloadCheck = false;

                //6.等待3秒
                StartCoroutine (WaitForReload ());

                //7.播放重新装弹的音效
                reloadSound.Play ();
            }



            //弹药数量减少
            ammo1 -= 1;
            string ammo1String = (ammo1).ToString ();
            ammo1Text.text = ammo1String;

            ammo2 -= 1;
            string ammo2String = (ammo2).ToString ();
            ammo2Text.text = "/"+ ammo2String;


            //如果弹药总数量为0，则设置ammoIsEmpty为true
            if (ammo2 == 0) {

                ammoIsEmpty = true;
                ammo1 = 0;
                string ammoTempString = (ammo1).ToString ();
                ammo1Text.text = ammoTempString;

            }


            //播放音效
            shootSound.Play();

            Debug.Log ("shooting!");

            //定义一个RaycastHit类型变量，用于保存检测信息
            RaycastHit hit;
```

```
                //判断是否检测到命中敌人
                if (Physics.Raycast (fpsCam.transform.position,
fpsCam.transform.forward, out hit)) {


                    //获取所受攻击的敌人
                    Enemy target =
hit.transform.GetComponent<Enemy>();

                    //destroy enemy

                    if (target != null) {

                        //instantiate blood effect

                        target.TakeDamage (damage);
                        //创建敌人受伤的粒子特效

                        GameObject bloodBurst = Instantiate
(bloodEffect, hit.point, Quaternion.LookRotation (hit.normal));

                        //0.2秒后销毁粒子特效

                        Destroy (bloodBurst, 0.2f);
                    }  else {
                        //load shooting effect

                        //如果没有击中敌人，则创建攻击时的粒子特效

                        GameObject shootingGo =  Instantiate
(shootingEffect, hit.point, Quaternion.LookRotation
(hit.normal));

                        //0.2秒后销毁粒子特效
                        Destroy (shootingGo,0.2f);

                    }


                    //攻击敌人时添加一个额外的冲击力

                    if (hit.rigidbody != null) {
```

```
                        hit.rigidbody.AddForce (-hit.normal *
forceAdd);

                    }


                    //输出所命中的对象名称
                    Debug.Log (hit.transform.name);
                }
                //播放开火的粒子特效
                muzzleFlash.Play ();
                //播放开火动画
                pistol.GetComponent<Animator>().Play("Fire");
            }

        }

}
```

所添加的新代码都已经添加了注释，这里就不再一一赘述了。

在最后测试之前，我们还有一处小小的修改，在Hierarchy视图中找到CameraParent-Main Camera-weapon1-MuzzleFlash-Flare，然后在Inspector视图中取消勾选Play On Awake.



全部完成后在编辑器上点击工具栏上的Play按钮，预览下游戏效果。

好了，本课的内容到此结束，我们下一课再见。