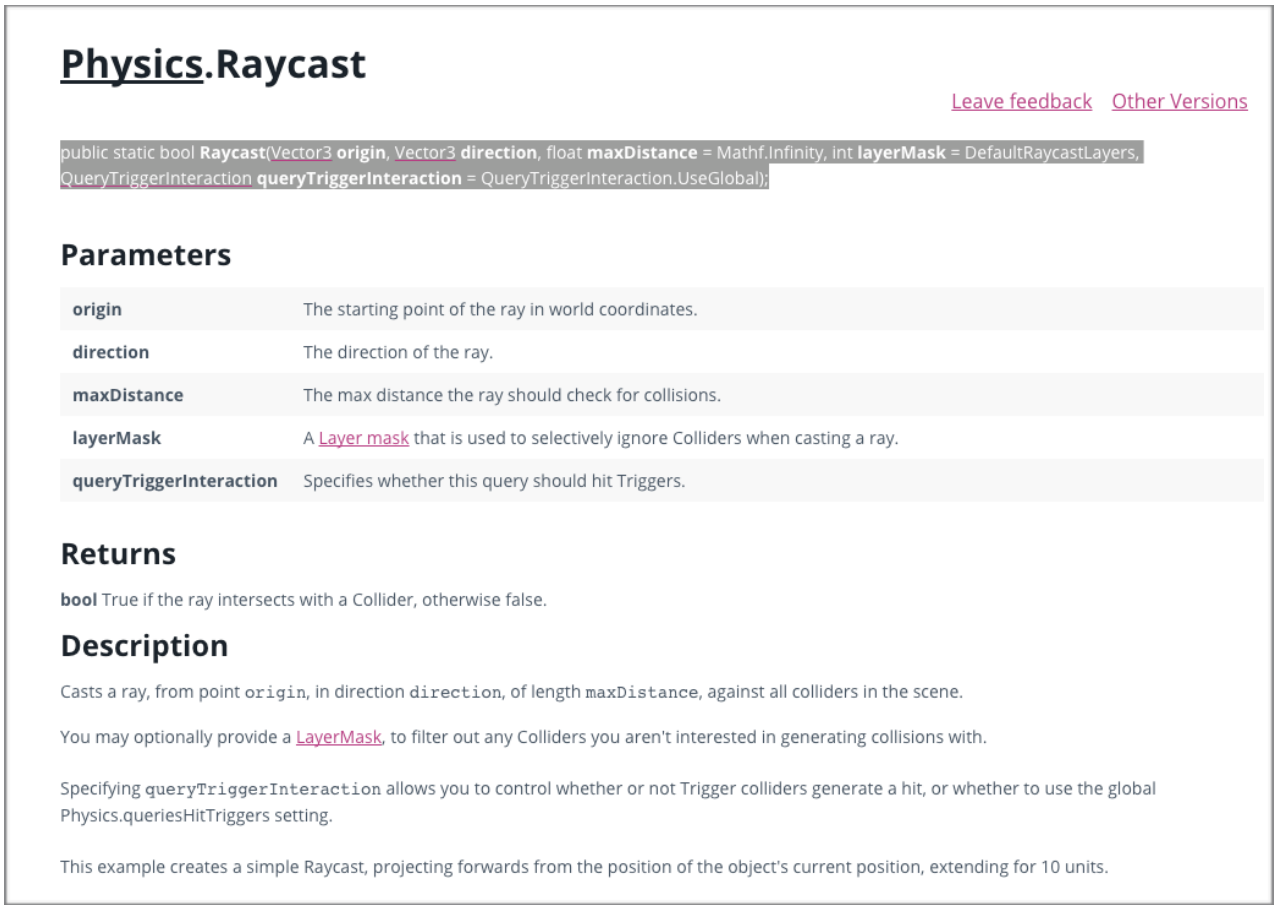
欢迎回到我们的学习。

到目前为止，敌人的各种功能可以说是比较齐备了，但是作为玩家，却只能被动挨打，不能主动攻击。

因此，在本课的内容中，我们将实现对敌人的反击。为此，我们需要学习Unity中Physics.Raycast的概念。

首先在官方API文档中查看与之相关的定义：
https://docs.unity3d.com/ScriptReference/Physics.Raycast.html



Physics.Raycast

Leave feedback Other Versions

public static bool **Raycast**(Vector3 **origin**, Vector3 **direction**, float **maxDistance** = Mathf.Infinity, int **layerMask** = DefaultRaycastLayers, QueryTriggerInteraction **queryTriggerInteraction** = QueryTriggerInteraction.UseGlobal);

Parameters

origin	The starting point of the ray in world coordinates.
direction	The direction of the ray.
maxDistance	The max distance the ray should check for collisions.
layerMask	A Layer mask that is used to selectively ignore Colliders when casting a ray.
queryTriggerInteraction	Specifies whether this query should hit Triggers.

Returns

bool True if the ray intersects with a Collider, otherwise false.

Description

Casts a ray, from point `origin`, in direction `direction`, of length `maxDistance`, against all colliders in the scene.

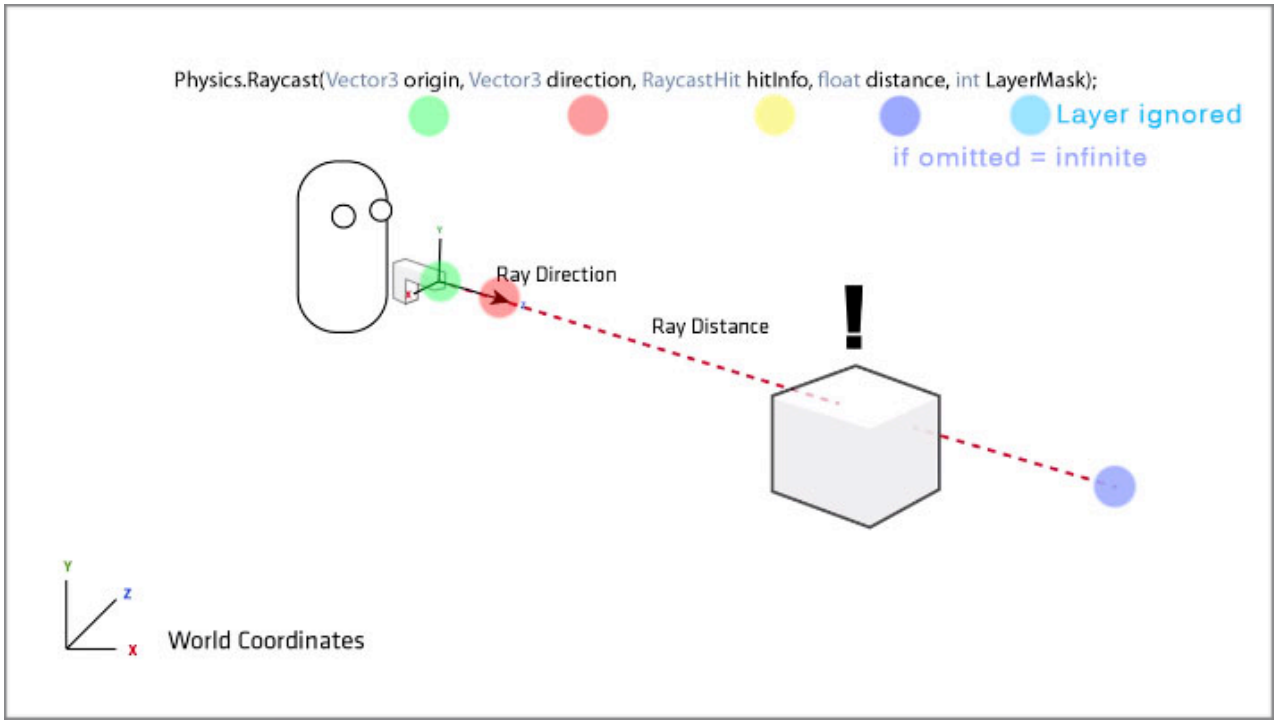
You may optionally provide a LayerMask, to filter out any Colliders you aren't interested in generating collisions with.

Specifying `queryTriggerInteraction` allows you to control whether or not Trigger colliders generate a hit, or whether to use the global Physics.queriesHitTriggers setting.

This example creates a simple Raycast, projecting forwards from the position of the object's current position, extending for 10 units.

可以看到，在Unity中，Physics.Raycast是从origin位置沿着direction的方向，发出一条长度为maxDistance的射线，而目标则是检测场景中的所有碰撞体。
其中layerMask参数用来选择性的过滤某些碰撞体。
queryTriggerInteraction参数用来指定该查询是否应命中trigger。

具体可以参考下图。



在了解了相关原理之后，接下来就是具体来实现了。

打开Unity编辑器，在Project视图中，找到Assets-_Scripts文件夹，然后右键单击，创建一个新的脚本文件，将其命名为ShootEnemy，在MonoDevelop中打开。

更改其中的代码如下：

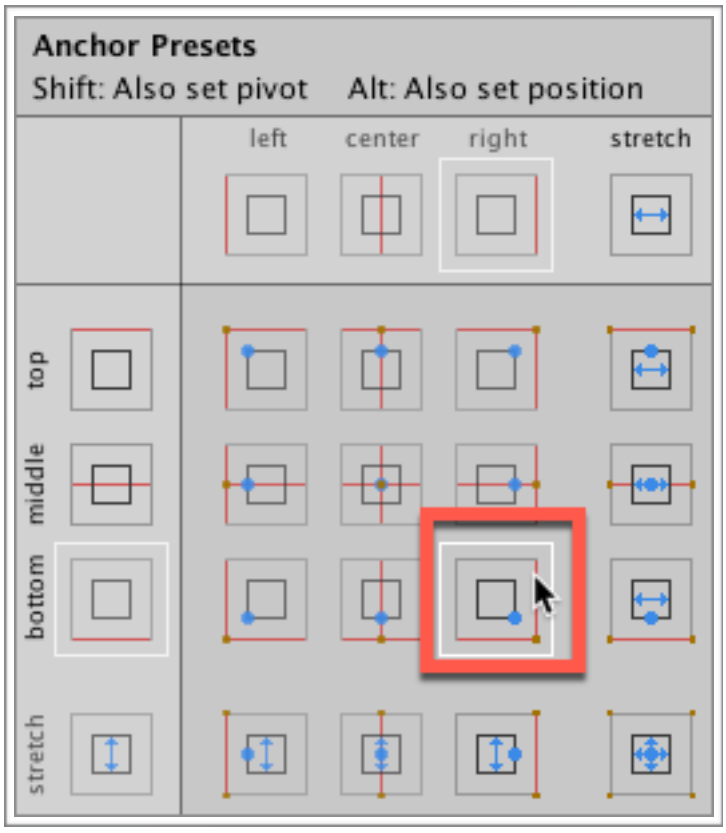
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//1.import namespace
using UnityEngine.UI;

public class ShootEnemy : MonoBehaviour {

    //2.创建到Button对象的引用
    public Button shootBtn;
    //3.创建到主摄像机的引用
    public Camera fpsCam;

    // Use this for initialization
    void Start () {

    }
```

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

        }
}
```

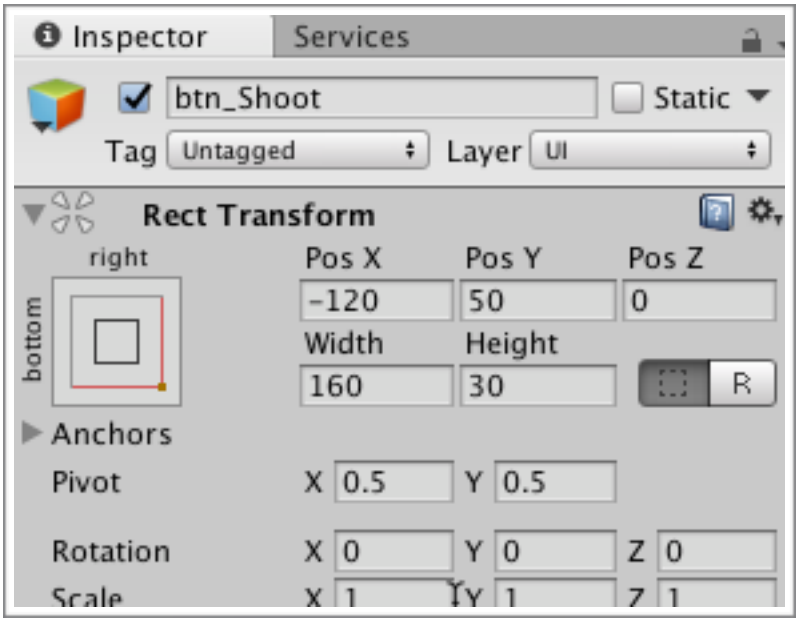
以上代码中只有注释行1，2，3相关的代码是我们添加的，大家直接看注释就明白每行代码的作用了，这里就不再赘述。

接下来回到Unity编辑器，在Hierarchy视图中找到Canvas对象，然后右键单击，选择UI-Button，创建一个新的按钮，并将其命名为btn_Shoot。删除按钮所对应的文本，然后切换到Game视图。

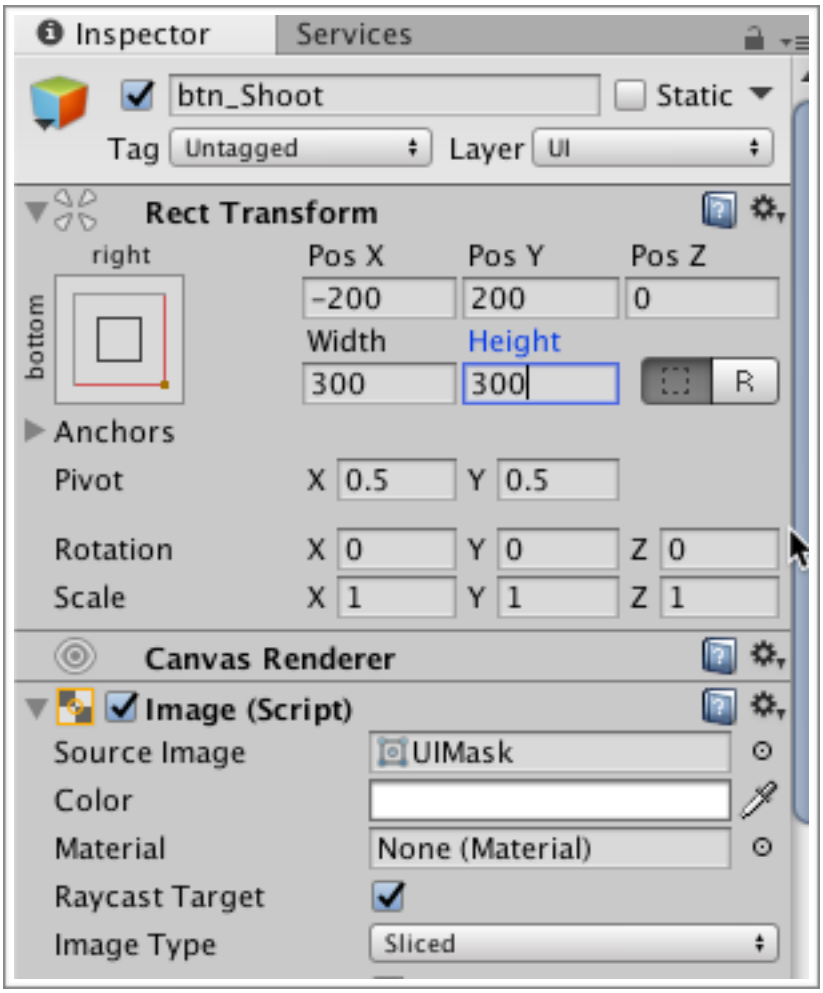
保持选中btn_Shoot对象，在Rect Transform中设置锚点类型为bottom right。



然后设置Pos X和Pos Y，使得按钮显示在合适的位置。



然后更改按钮的Image组件的Source Image属性为UIMask，更改Rect Transform中的Width 和Height 为300，300



设置好了按钮之后，在Hierarchy视图中选择CameraParent对象下面的MainCamera，然后右键单击，创建一个空的游戏对象，将其命名为weapon1。在Inspector视图中点击Add Component，然后选择ShootEnemy。

在Shoot Enemy组件的属性中，将Shoot Btn设置为btn_Shoot按钮，将Fps Cam设置为Main Camera。

接下来打开ShootEnemy.cs，并添加对事件的响应代码，更改后的代码如下：

```
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;
```

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

		//1.添加按钮的响应事件
		shootBtn.onClick.AddListener (OnShoot);
	}

	void OnShoot(){

		//2.定义一个RaycastHit类型变量，用于保存检测信息
		RaycastHit hit;

		//3.判断是否检测到命中敌人
		if (Physics.Raycast (fpsCam.transform.position,
fpsCam.transform.forward, out hit)) {

		//destroy enemy

		//instantiate blood effect

		//load shooting effect

		//4.输出所命中的对象名称
		Debug.Log (hit.transform.name);
		}

	}

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

	}
}
```

按照注释行编号简单解释一下：

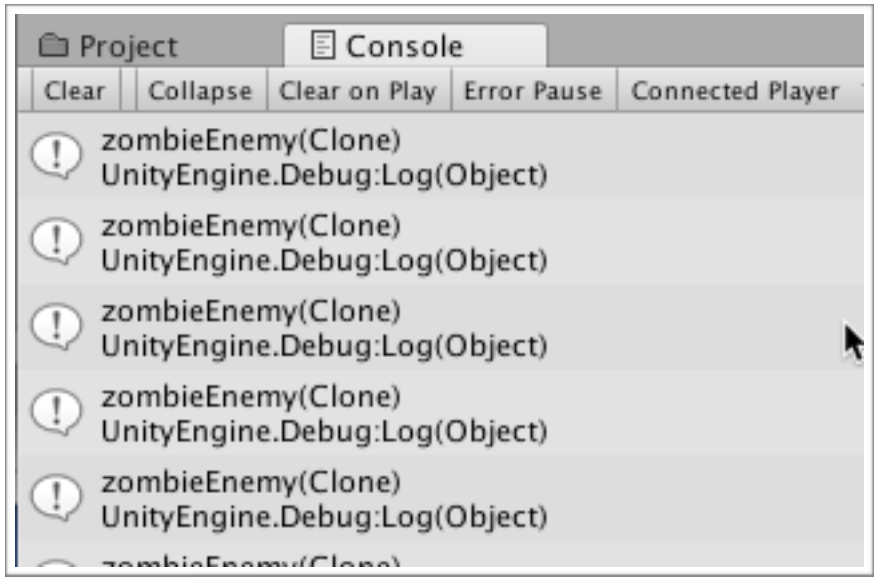
1.这里添加射击按钮的响应方法，OnShoot

2.这里定义了一个RaycastHit类型的变量，用于保存检测信息

3.使用Physics.Raycast方法来判断是否检测到命中敌人

4.在实现具体的效果之前，先在Console中输出所命中的对象名称。

回到Unity编辑器，点击Play按钮预览游戏效果。在Game视图中点击Start Game，移动主摄像机的位置，然后点击屏幕右下角的按钮，可以在Console面板中看到所命中的对象名称。



接下来让我们完善命中敌人后的具体效果。

首先在Project视图中找到Assets-_Prefabs文件夹，选择zombieEnemy这个预设体，然后在Inspector面板中点击Add Component，添加一个新的脚本，命名为Enemy。

在MonoDevelop中将其打开，并更改代码如下：

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

public class Enemy : MonoBehaviour {

    //1.设置敌人的生命值
    public float health = 30f;

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

```
    }

    //敌人受到伤害后的处理
    public void TakeDamage(float damage){

        //2.敌人生命值减少特定的数值
        health -= damage;

        //输出敌人生命值
        print (health);


        //3.当敌人生命值变为0的时候，就死亡
        if (health <= 0) {

        //4.Enemy Die

            Die ();
        }

    }

    //敌人死亡
    void Die(){

        //5.在1秒钟后销毁敌人对象
        Destroy (gameObject, 1f);
    }


}
```

这里还是按照注释行的数字编号来解释下相关代码。
1.设置敌人的生命值变量
之所以设置为public类型，是因为我们将在Enemy.cs之外的代码中访问该变量

2.让敌人的生命值减少特定的数值

3.当敌人生命值减少为0时，就进入死亡状态

4.调用敌人死亡的方法

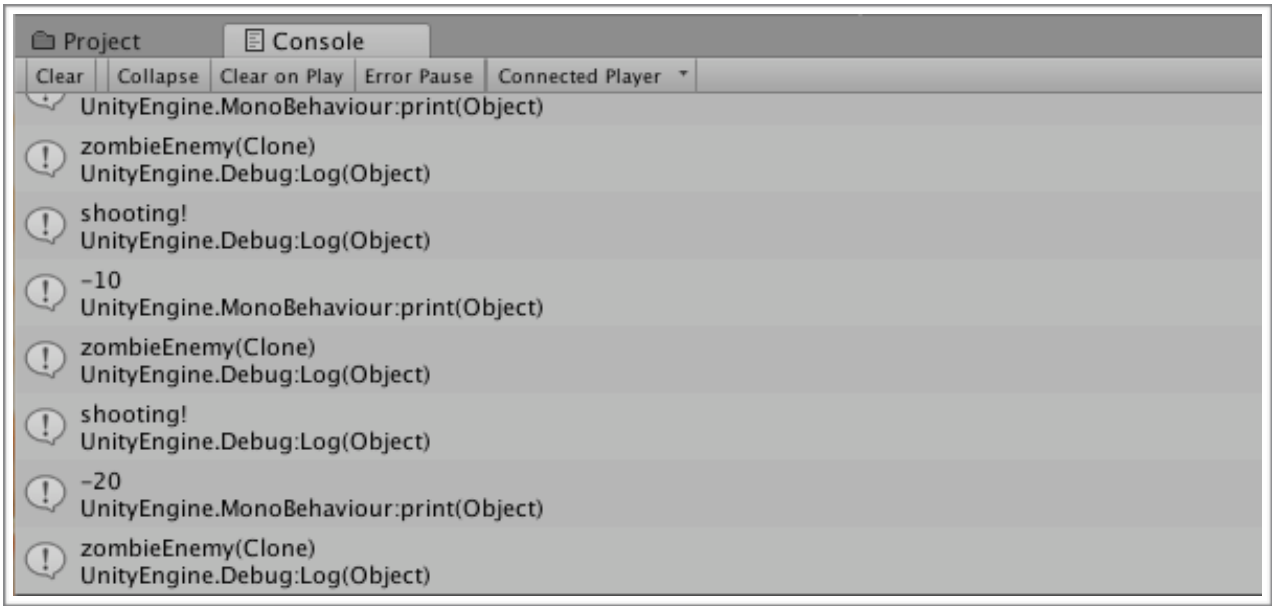
5.使用Destroy方法，在1秒钟后销毁当前敌人对象。

接下来切换到ShootEnemy.cs，更改后的代码如下：

```
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;

	//1.设置敌人每次受到伤害的数值
	public float damage = 10f;

	// Use this for initialization
	void Start () {

		//1.添加按钮的响应事件
		shootBtn.onClick.AddListener (OnShoot);
	}

	void OnShoot(){

		//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)) {



		//2.获取所受攻击的敌人
			Enemy target =
hit.transform.GetComponent<Enemy>();

		//3.destroy enemy
```

```
                if (target != null) {

                    target.TakeDamage (damage);

                }

            //instantiate blood effect

           //load shooting effect

            //输出所命中的对象名称
            Debug.Log (hit.transform.name);
            }

    }

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

    }
}
```

按照注释行的数字编号来简单解释一下：
1.这里设置了敌人每次受到伤害的数值
2.使用hit信息获取所受攻击的敌人对象
3.如果敌人对象不为空，那么调用Enemy.cs中的TakeDamage方法，让其生命值减少。
回到Unity编辑器，点击工具栏上的Play按钮预览游戏效果。
可以看到每次敌人受到攻击时，都会在Console中显示敌人的当前生命值。

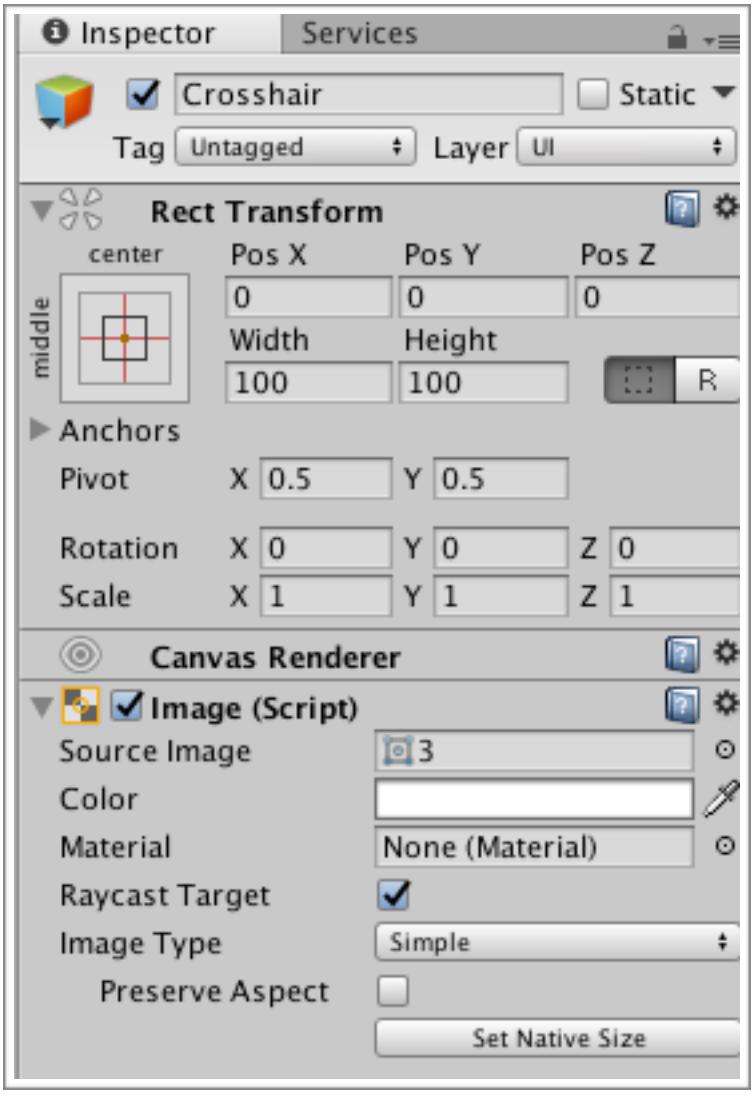


为了让我们方便攻击敌人，这里我们再添加一个准星。

在Unity编辑器中切换到Asset Store，搜索crosshair，然后找到下面这个插件。

下载并导入该插件，并将其归入Components文件夹。

在Hierarchy视图中选择Canvas，右键单击，选择UI-Image，添加一个新的Image控件，将其更名为Crosshair。然后在Inspector视图中将Image组件的Source Image属性更改为3这个纹理图片。



好了，此时在Game视图中可以看到多了一个准星。