Cha4-Unity3d 和 C#的双剑合璧 12

在上一课的内容中，我们成功的实现了开宝箱功能~

而在实际的游戏开发中，我们经常会遇到两类情况。其中一种是当玩家使用现代热武器（枪械、飞毛腿导弹/激光炮之类的）攻击敌人时，如何判断子弹/导弹/激光如何准确的灭中了敌军~而另一种情况则是，玩家或者 NPC 角色需要判断在自己的视野范围内是否存在不太令自己喜欢的角色。

那么应该怎么办呢？

用物理碰撞吗？显然不适合。那么 Trigger 呢？也不适合。这个时候，我们就要用到传说中的仙术“开天眼”了~

开个玩笑，这个时候，我们需要使用 Unity3d 提供的一种名为 Raycasting 的利器，从特定位置向特定方向发射一条射线，然后检测目标物是否处于射线所在的路线上。

好了，让我们开始这一课的学习吧~

个人微信号：iseedo

微信公众号：vrlife

12 Unity 和 C#的双剑合璧-尝尝本公主的镭射激光

在学习具体的代码之前，首先让我们到官方文档上去插查找一下所需要学习的内容。

首先在浏览器中输入以下网址：https://docs.unity3d.com/Manual/index.html

然后搜索 ray 或者 raycast 就好。

具体的内容就不多说了，大家可以自己仔细读一读官方的文档。

我们将分三步完成本课的内容：

1.发射一条射线

2.让射线返回信息

3.让射线返回我们需要的信息~

首先是第一个任务，发射一条射线（镭射激光）。

好了，废话少说，让我们直接上代码吧~

在 Unity 中打开项目，然后双击 PlayerScript.cs 在 Visual Studio 中将其打开。

在 Start()方法之前添加以下代码：

```
//创建变量用来保存 RaycastHit 类型的变量
private RaycastHit hit;

//创建变量用来保存 Ray 类型的变量
private Ray ray;

//创建变量用来保存 raycast 的长度
public float rayDistance = 5f;
```

以上三行代码其实没有什么好解释的，就是定义了三个不同类型的变量，分别用来保存 RaycastHit，Ray 的实例变量，以及 raycast 的长度。

其中 RaycastHit 表示 raycast(镭射激光）所命中的目标，Ray 代表镭射激光自身，而 rayDistance 则保存了镭射激光的长度。

接下来更改 Update 方法的内容如下：

```
//18.发射一条镭射激光
ray = new Ray(transform.position, transform.forward);

//19.实际绘制镭射激光
```

Debug.DrawRay(ray.origin, ray.direction * rayDistance , Color.red);

先回答一个问题~

为什么这些代码要放到 Update 方法中？很简单，因为玩家操控角色的位置和状态随时都在发生变化，因此我们需要每帧获取最新的信息，所以就需要在 Update（或者 xxxUpdate)方法中放置这些代码。

接下来按照注释行的编号来简单解释一下：

18. 这里我们首先创建了一个新的 Ray 结构体。

问题来了，什么是结构体？

简单来说，结构体就是一堆属性的组合。比如个人的体检报告就可以定义成一个结构体，里面包含了多个变量及其对应的属性值。

结构体看起来跟类有点像，但是它没有可执行的方法。比如个人的体检报告不会执行什么任务，只是一堆信息的组合而已~

Ray 这个结构体的定义其实非常简单，

public Ray(Vector3 origin, Vector3 direction);

它包含了两个属性变量，分别是起点（也就是一个 Vector3 类型的 origin 变量），方向（同样是一个 Vector3 类型的 direction 变量）。

在我们的这一行代码中，

ray = new Ray(transform.position, transform.forward);

我们将射线的起点设置为角色所在的位置，将方向设置为角色所面朝的默认方向~

接下来看下一行代码。

19. Debug.DrawRay(ray.origin, ray.direction * rayDistance , Color.red);

在之前的内容中，我们接触过 Debug.Log 这个系统方法，它的作用是在 Console 视图中输出文本信息，跟 print 类似。

需要注意的是，Debug.Log 所输出的信息玩家是看不到的哦~

类似的，Debug.DrawRay 方法的作用就是在 Scene 视图中绘制一条射线，但是在 Game 视图中是

看不到的，玩家角色同样也看不到这条射线。

Debug.DrawRay 这个方法的定义如下：

```
public static void DrawRay(Vector3 start, Vector3 dir, Color color =
Color.white, float duration = 0.0f, bool depthTest = true);
```

可以看到，完整的 Debug.DrawRay 方法可以有五个参数，分别是起点，方向和长度，颜色，持续时间和深度测试。

这里我们只用到了三个参数，也就是起点，方向和颜色。

其中起点设置成了 ray 射线的起点，方向长度设置成了射线的方向乘以射线长度，颜色设置为标准的红色。

好了，现在脚本中的代码已经搞定。

返回 Unity 编辑器，点击工具栏上的预览。

咦，为毛啥都没有？

Don't panic~不要恐慌，切换到 Scene 视图，就会看到期待中的红色射线如约而至了。

还是没有？！

哈哈，你会发现 Lady Fairy 这个游戏对象的脚本组件中有 Ray Distance 的默认属性值是 0，这里需要修改为 5.

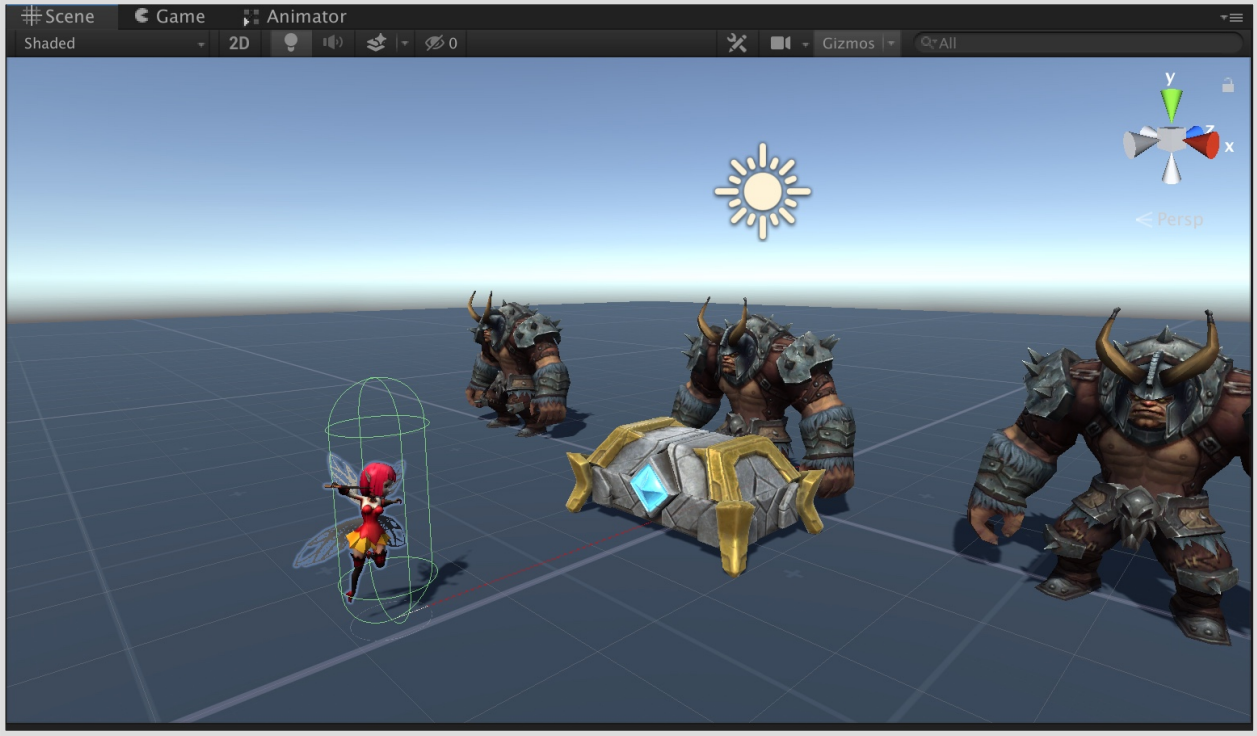
再次尝试，一次 OK 了~

当然，当前的射线是从角色的底部发射出去的，这是因为默认情况下游戏对象的 origin 点就是这个位置。此外，仔细观察你还会发现，这条射线是从 Z 轴发射出去的，方向就是精灵公主的朝向~

现在让我们稍微优化一下，让射线从精灵公主的身体中部发射。

回到 PlayerScript.cs，更改 Update 方法中刚才注释编号为 18 的代码：

```
//18.发射一条镭射激光
ray = new Ray(transform.position + new Vector3(0f, playerCollider.center.y,0f), transform.forward);
```



这里，这里我们让射线的初始位置的 y 方向抬高到了角色的碰撞体的中心位置高度。

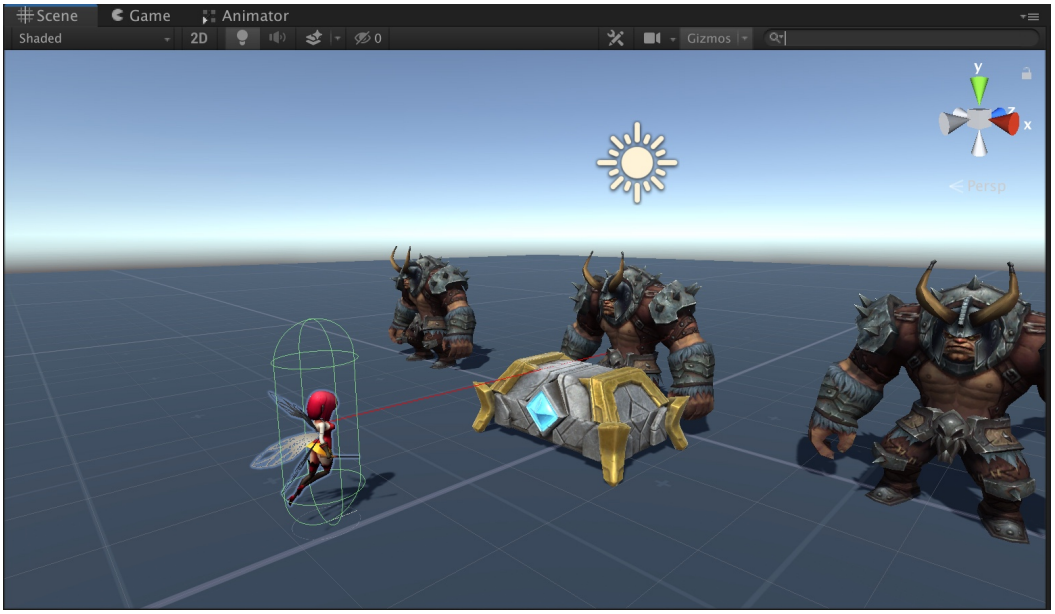
回到 Unity，点击播放预览，查看效果~

可以看到，现在射线的位置已经优化到从角色的身体中部发射了。

OK，镭射激光已经发射成功，接下来看看它能给我们返回什么信息了。

回到 PlayerScript.cs，在 Update 方法的最后添加代码：

```
//20.检测激光碰撞事件并返回信息
//21.如果激光射线碰撞事件发生
```




```
if(Physics.Raycast(ray,out hit))
{
    //22.如果/距离小于射线距离
    if(hit.distance < rayDistance)
    {
        //23.输出信息
        print("报告公主殿下，镭射激光命中了不明物体！");
    }
}
```

这里按照注释行编号来解释一下：

20.整段代码的作用是检测激光碰撞事件，并返回信息

21.如果激光射线碰撞事件发生，则执行方法体内容

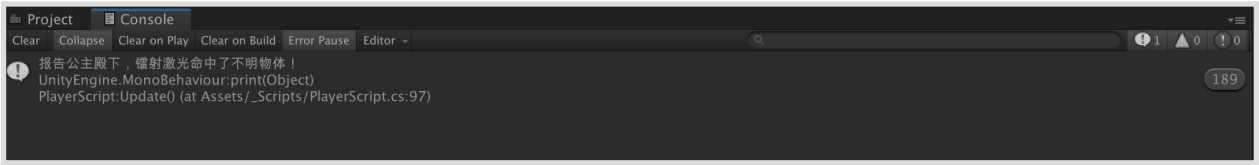
22.如果/距离小于射线距离，则执行操作

23.在 Console 视图中输出一行信息

返回 Unity，点击工具栏上的预览播放按钮，查看效果。

可以看到，在 Console 视图中，只要公主的前进方向上有敌人，且距离小于所设置的射线距离，

就会在这里输出信息。



需要提醒大家的是，射线不光会命中碰撞体，也会和 Trigger 类型的物体发生反应~

第二条任务已完成，现在进入最后一项任务，返回我们所需要的信息，也就是射线究竟碰到了什么游戏对象。

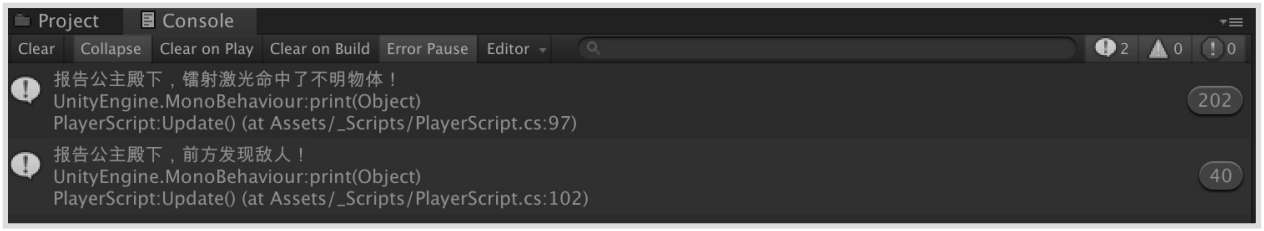
回到 PlayerScript.cs，更改 Update 方法中注释行为 20-23 的部分代码：

```
//20.检测激光碰撞事件并返回信息
//21.如果激光射线碰撞事件发生
if(Physics.Raycast(ray,out hit))
{
  //22.如果/距离小于射线距离
  if(hit.distance < rayDistance)
  {
    //23.输出信息
    print("报告公主殿下，镭射激光命中了不明物体！");
    //24.判断射线碰到的物体是否是敌人
    if(hit.collider.gameObject.tag == "Enemy")
    {
      //25.输出信息
      print("报告公主殿下，前方发现敌人！");
    }
  }
```

```
    }
```

这里添加了一个新的条件判断。

如果射线碰到的物体是否是敌人，就输出信息。



返回 Unity 编辑器，点击工具栏上的预览播放按钮，查看效果。

只有当精灵公主的前方是敌人的时候，才会输出下面的信息。

试着调整 rayDistance 的属性值，可以试试看怎样才会有反应~

好了，本课的内容就到这里了。

让我们下一课再见。