Cha4-Unity3d 和 C#的双剑合璧 05

在前两课的内容中，我们学习了如何通过代码访问游戏对象及其组件，以及修改相关的属性。

在这一课的内容中，我们要基于之前的内容，学习如何让玩家操控游戏角色。

还等什么呢？让我们开始吧~

个人微信号：iseedo

微信公众号：vrlife

06 Unity 和 C#的双剑合璧-让玩家控制游戏角色

在游戏当中，作为“上帝”的玩家当然希望可以随心所欲的控制游戏角色了。那么，应该如何控制呢？最理想的状态当然是通过脑机接口~不要😄，再过三五十年，人类未来的游戏交互方式可能就是《黑客帝国》里面那样，直接让大脑和电脑连接在一起了。

不过在当前这个时间点，我们熟悉的玩家交互方式仍然是键盘、鼠标、触摸屏为主。当然，对于游戏主机和掌机还有游戏手柄，对于 AR/VR 游戏还有动作捕捉和面部识别、表情捕捉、眼部追踪等等~

在这一课的内容中，我们先来了解最简单的交互方式，也就是 PC 游戏中最常用的是用键盘和鼠标进行交互。在后续的内容中，我们还将学习如何基于智能移动设备的触摸屏进行交互，以及如何基于 AR/VR/MR 的语音识别、动作识别、面部识别等进行交互~

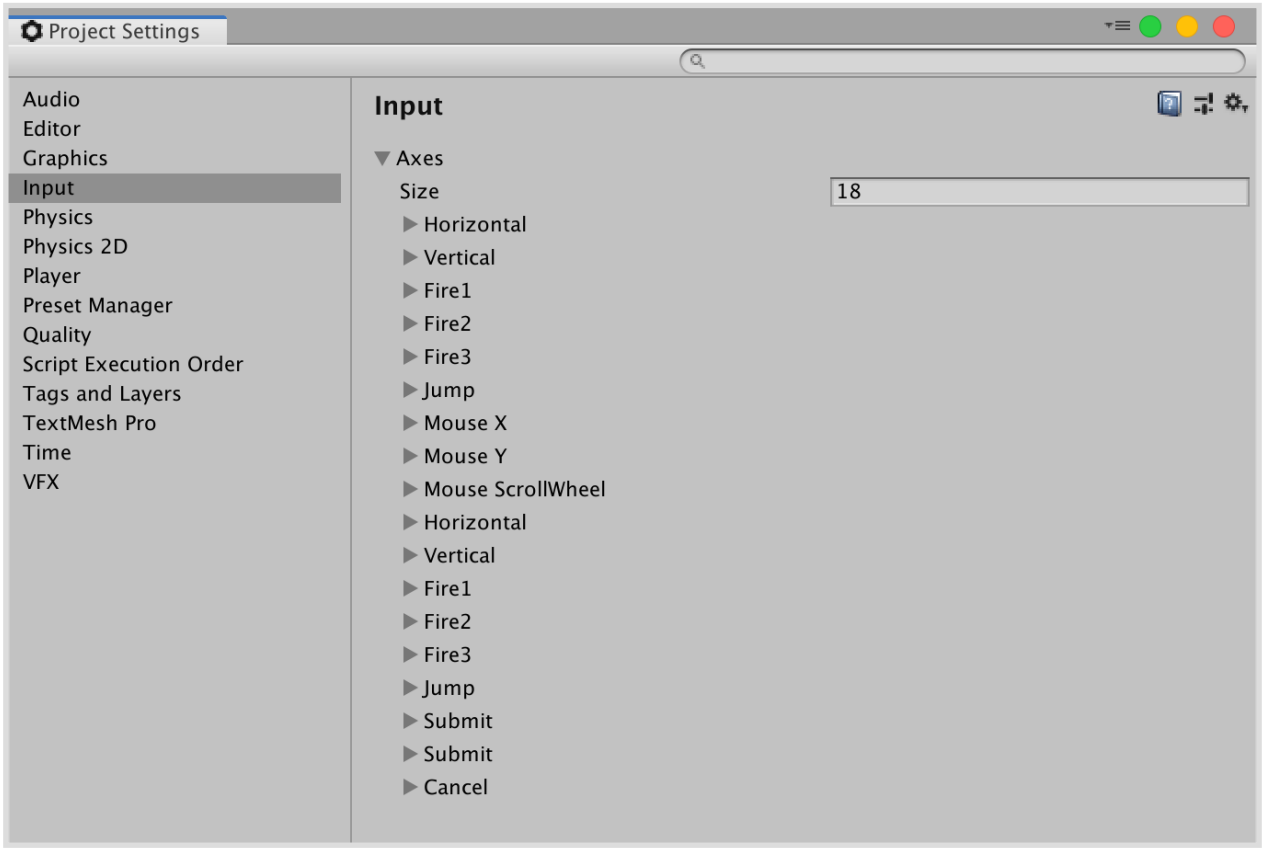
心动不如行动，让我们开始动手吧~

打开 Unity Hub，找到并打开 FairyLand 项目。

对于 PC 平台的游戏，其实设置玩家的输入非常简单直接。

在 Unity 编辑器中点击工具栏上的 Edit- Project Settings，然后从左侧切换到 Input 选项，就可以看到系统默认的输入设置了。

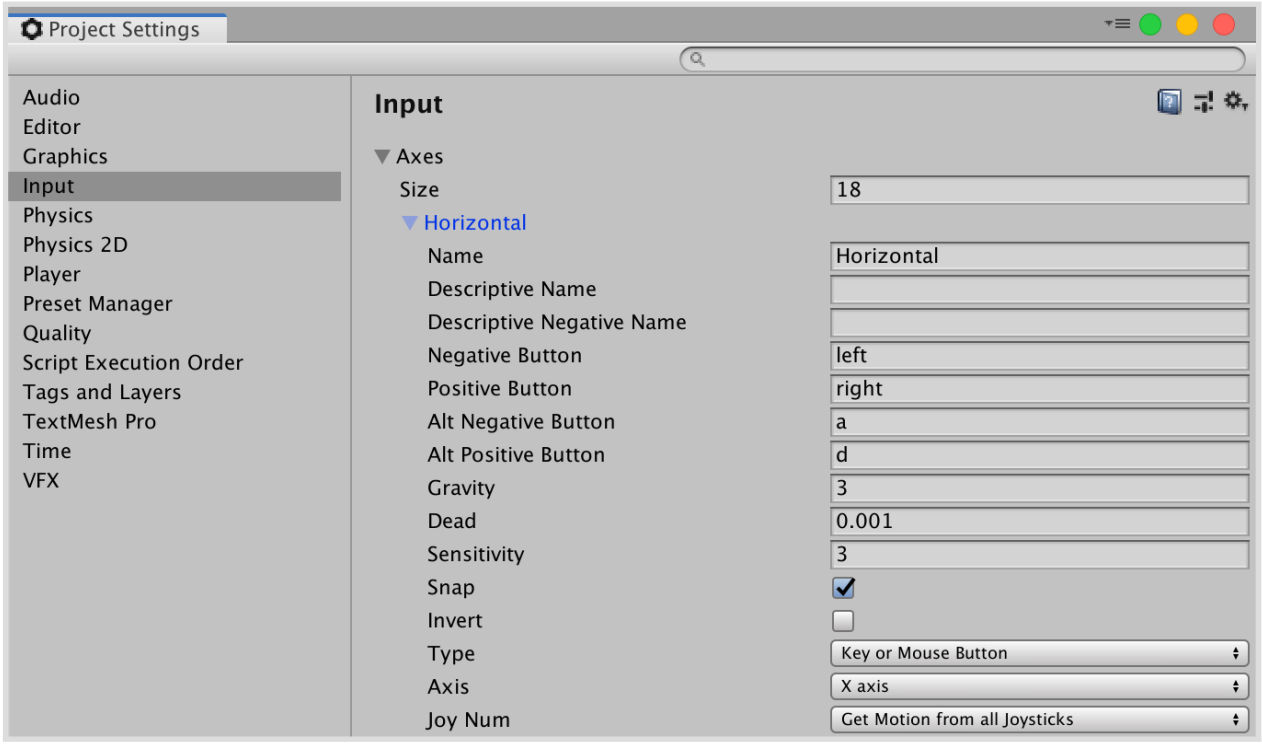
如果你的初中英语还过得去，那么看懂这里的设置选项完全不是问题。



首先 Size 代表所有可以用的输入设置，默认值是 18，也就是下面的 18 个设置选项。

接下来的 Horizontal 代表水平方面的操控，点击下三角符号可以展开。

在下面的这些属性中，挑几个重要的简单解释一下。



Name，这里设置的是 Horizontal，也就是说在代码中我们将使用这个字符串来指代水平方向的输入。

Negative Button 和 Positive Button 代表的是负向和正向的移动，分别设置成了 left（键盘的向左方向键）和 right(键盘的向右方向键)。

简单来说，按下键盘上向左的方向键，就会让所关联的游戏对象产生向左的移动（x 坐标的负向减少），反之就是向右的移动。

Alt Negative Button 和 Alt Positive Button 和刚才的设置类似，只不过使用了键盘上的 a 字母键和 d 字母键。也就是说，按下键盘上的这两个键，就等同于按下键盘的向左方向键和向右方向键。

下面的 Type 设置成了 Key or Mouse Button，也就是说这个设置是针对键盘或鼠标按键的。

关于其它属性的含义，这里就不展开讲了，感兴趣的童鞋可以查看官方文档：
https://docs.unity3d.com/Manual/ConventionalGameInput.html

对于其它 17 种输入设置，跟 Horizontal 类似，这里就不详细讲了。

对当前这个项目来说，最主要的就是 Horizontal 和 Vertical 的输入了。

好了，现在我们已经明确了键盘上的输入映射，接下来就需要用代码来实际获取玩家的输入，并驱动游戏角色做我们希望它做的事情了~

获取玩家的输入

在 Project 视图中找到_Script 目录下的 PlayerScript，双击在 Visual Studio 中将其打开。

在之前的内容中，我们提到过 Unity 所提供的事件函数。

目前基本上所有的代码都放到了 Start()这个事件函数中，这是因为我们只需要在游戏开始的时候执行一次代码就好了。

但是处理玩家的交互就不同了，我们不知道玩家会在什么时候按下键盘上的某个键。因此，只是在 Start()中调用相关代码肯定是不够的，我们必须在游戏的每一帧监控玩家的输入状况。而对于在游

戏的每一帧都要执行的代码，通常我们会在 Update()、FixedUpdate()和 LateUpdate()等事件函数中。

因为玩家的输入不涉及到对物理机制的模拟，所以在 Update()中添加对应的代码就 Ok 了~

在 PlayerScript 脚本文件中，更改 Update()的代码如下：

```
void Update()
{
    //7.获取通过玩家输入产生的虚拟位移
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical =   Input.GetAxis("Vertical");


    //8.测试所获取的结果
    Debug.Log(moveHorizontal);
    Debug.Log(moveVertical);
}
```

这里简单解释一下：

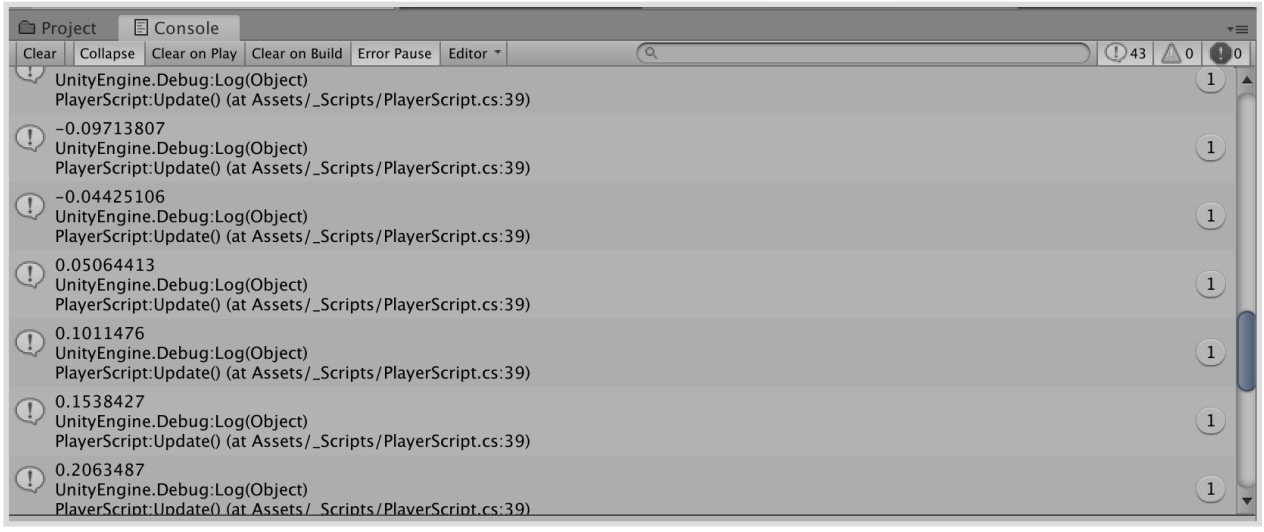
当我们希望获取通过玩家输入让游戏角色产生的水平和垂直位移时，需要用到 Input.GetAxis 方法，而这正是 Unity 内置的重要方法，详细信息可以参考：

https://docs.unity3d.com/ScriptReference/Input.GetAxis.html

问题来了：

作为新手小白，我怎么知道该用什么方法？

首先，因为这个操作涉及到玩家的输入，所以我们会需要用到系统的 Input 类，所以在官方文档里面找到 Input 类。



然后，我们需要获取通过玩家输入所产生的位移，在其方法中逐一找，会发现 GetAxis 是不二之选。当然，对于有经验的老手来说，这些就纯粹靠经验了。

最后，找到对应的方法，官方文档里面还会有详细的示例代码。即便你看不太懂，也可以先拿过来照着用~

好了，保存代码修改，回到 Unity 编辑器，点击预览播放按钮，看看 Console 视图中有什么新东西出现~

可以看到，当我们按下键盘上的上下左右方向键，或者 awsd 键时，在 Console 视图中会输出-1 到 1 之间的数值。

显然，Unity 已经成功捕捉到了玩家的交互输入。

让游戏角色走两步

仅仅获取玩家的输入还远远不够，因为我们真正想要的是让游戏角色可以在玩家的操控之下走两步~

那么，应该怎么办呢？

对于新手来说，可以模仿一下刚刚打开的官方文档：

https://docs.unity3d.com/ScriptReference/Input.GetAxis.html

这里提供了两个示例代码，在第一段示例代码中：

```
public class ExampleClass : MonoBehaviour
{
    public float speed = 10.0f;
    public float rotationSpeed = 100.0f;

    void Update()
    {
        // Get the horizontal and vertical axis.
        // By default they are mapped to the arrow keys.
```

```
        // The value is in the range -1 to 1
        float translation = Input.GetAxis("Vertical") * speed;
        float rotation = Input.GetAxis("Horizontal") * rotationSpeed;

        // Make it move 10 meters per second instead of 10 meters per frame...
        translation *= Time.deltaTime;
        rotation *= Time.deltaTime;

        // Move translation along the object's z-axis
        transform.Translate(0, 0, translation);

        // Rotate around our y-axis
        transform.Rotate(0, rotation, 0);
    }
}
```

当我们获取了玩家在 Horizontal 和 Vertical 方向的输入后，将所获取的结果乘上运动 的速度，然后又乘上 Time.deltaTime，最后，使用 Translate 方法让角色进行位移，使用 Rotate 方法让角色进行旋转。

回到 Visual Studio 之中，继续修改 PlayerScript 中的代码。

首先，在 Start()方法之前添加一个新的变量定义：

```
    //9.创建变量用来保存游戏角色的运动速度
     public float moveSpeed = 5.0f;
```

这一行代码没什么好解释的，就是定义了一个 float 类型的变量，保存角色的运动速度。因为是 public 类型的，我们可以在其它类，或者 Inspector 视图中进行访问和修改。

然后更改 Update()方法中的代码：

```
    void Update()
{
    //7.获取玩家通过键盘产生的水平和垂直位移
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical =   Input.GetAxis("Vertical");
```

////8.测试所获取的结果

//Debug.Log(moveHorizontal);
//Debug.Log(moveVertical);

//10.让游戏角色走两步~

transform.Translate(moveHorizontal * moveSpeed * Time.deltaTime, 0, moveVertical * moveSpeed * Time.deltaTime);
}

这里我们只添加了注释行编号为 10 下面的这段代码。

看起来有点复杂，其实也很简单。

首先，因为要移动游戏角色，所以我们使用了 transform.Translate 犯法。

当然，这里其实我们也偷懒了，严谨的写法应该是:

gameObject.transform.Translate(moveHorizontal * moveSpeed * Time.deltaTime, 0, moveVertical * moveSpeed * Time.deltaTime);

只不过因为我们要移动的游戏角色就是当前这个游戏对象，所以就把最前面的 gameObject 省略了~

而 Translate 方法则是 Transform 类的重要方法，它的作用就是让游戏角色产生位移。

假定我们不知道怎么用这个方法，那么当然还是参考官方文档了~

https://docs.unity3d.com/ScriptReference/Transform.Translate.html

可以看到，这个方法的定义是：

public void Translate(Vector3 translation);
public void Translate(Vector3 translation, Space relativeTo = Space.Self);

也就是说需要给它传递一个 Vector3 类型的变量。在之前的内容中我们提到过，Vector3 是一个结构体，它通常用来保存游戏对象在空间中的坐标或坐标位移，分别是 x 轴，y 轴和 z 轴。

所以，如果严谨起见，其实我们应该这样写刚才的代码：

```
Vector3 translation = new Vector3(moveHorizontal * moveSpeed * Time.deltaTime, 0, moveVertical * moveSpeed * Time.deltaTime);
transform.Translate(translation);
```

问题又来了？为什么角色的位移不直接用 moveHorizontal 和 moveVertical 的变量值？

刚才提到过，其实使用 Input.GetAxis()方法所获取的位移数值在-1 到 1 之间，是一个相对值。所以，如果我们希望使用这个值乘以角色的移动速度，才会有一个合理的结果。

那么 Time.deltaTime 又是啥东西？

这个也不难理解。因为我们是在 Update()方法中执行这行代码。如果不乘以 Time.deltatime，那么就意味着每一帧都要产生这么多的位移，这个就有点夸张了。

当我们乘以 Time.deltaTime 之后，就代表每一秒产生这么多的位移。

如果大家希望了解 Time.deltaTime 的更详细用法，可以参考：

https://docs.unity3d.com/ScriptReference/Time-deltaTime.html

当然，我们还有个非常直观的方式来认识 Time.deltaTime，那就是添加一行代码~

```
print(Time.deltaTime);
```

实际执行后，根据电脑的配置不同，会看到不同的结果，在我的电脑上 Time.deltaTime 的数值大概是 0.017~

具体的数值是多少并不重要，重要的是，通过在 Update（ ）方法中乘以 Time.deltaTime，就可以确保位移是以每秒为单位，而不再是以每帧为单位。

好了，保存代码，回到 Unity 编辑器，点击工具栏上的预览播放按钮，就可以使用键盘上的上下左右方向键和 awsd 键来控制游戏角色的移动了~

是不是有点小小的成就感了。

不过且慢，当前的游戏视角看起来似乎有点问题，因为游戏角色在场景中的位置并不是很合理。

这里有个小小的技巧。

让我们回到 Unity 编辑器，在 Scene 视图中手动将游戏角色调整到自己满意的位置，然后在 hierar 视图中选中 Main Camera,按下快捷键 Shift+Ctrl+F(Mac 下 Shift+Command+F)。

再次點擊預覽按鈕，就可以看到角色的視角看起來比較舒服了~

當然，還有一個小小的問題，現在玩家的活動空間似乎有點太小，而運動速度似乎有點太快了。

我們可以在 Inspector 視圖中調整 Move Speed 的值，到自己滿意的程度~

至於活動空間，可以在 Hierarchy 視圖中選中 Floor，在 Inspector 視圖中將 Transform 組建的 Scale 屬性設置為（10，1，10）.

好了，再次預覽遊戲效果，我們的小仙女終於可以愉快的在虛擬世界中飛來飛去了~

好了，这一课的内容到此结束。

让我们下一课再见。