开始前的废话

本教程适用对象：
1.对使用虚幻 4 引擎开发 AR/VR 游戏感兴趣的童鞋；
2.此前几乎没有过任何平台的开发经验，当然，如果有任何平台的游戏或应用开发经验就更好不过。

本教程将包含的内容：
Part 1.使用虚幻 4 引擎中的蓝图系统开发简单的游戏
Part 2.使用虚幻 4 引擎中的蓝图系统和 C++开发游戏
Part 3.使用虚幻 4 引擎开发 ARKit 平台的游戏
Part 4.使用虚幻 4 引擎开发 HTC Vive 平台的游戏

所使用的软硬件开发环境：
硬件：
Macbook Pro Retina（Part1，2，3）
定制 PC（显卡 1080,其它随意，Part4）
HTC Vive （Part 4)
iPhone X(Part 3)

操作系统：
Mac 10.12.6(Part1,2,3）
Win 10(Part4）

开发工具：
Unreal Engine 4.18.1
Visual Studio Community 2017
Xcode 9.2

新手的虚幻 4 引擎指引

很多朋友对 Unity3d 比较熟悉了，因为大量的 3D 手游都是用 unity 开发的，比如我们最熟悉的《王者荣耀》和《炉石传说》。而以往在次时代 3A 游戏开发中有着重要地位的虚幻 4(Unreal Engine4）在手游横行的时代似乎有点落寞。实际上无论是 Unity3d 还是虚幻 4，在跨平台开发适配方面都有着不错的兼容性和延展性。虽然使用虚幻 4 来开发手游仍然不是主流的选择，但是在 VR/AR 时代，虚幻 4 引擎会有着更为广阔的空间。
当然，别忘了在主机游戏和次时代 PC 游戏开发方面，虚幻 4 引擎仍然是值得重点推荐的商业引擎。比如最近火遍全球的《绝地求生》吃鸡游戏就是用虚幻 4 引擎开发的。

使用虚幻 4 引擎开发的知名游戏还有很多，比如最终幻想 7(Final Fantasy)，铁拳 7，王国之心，DQ 等等。

在十年之前，使用虚幻引擎开发游戏基本上属于大型游戏工作团队的特权，一个很重要的原因就是，虽然虚幻引擎很强大，但是在那个年代它的授权费用也很强大。
在这方面不得不感谢老对手 Unity，正是这款把"人人都是 Unity3d 游戏开发者"概念推向极致的引擎让 Epic Games 不得不低下头来，重新思考并切换了此前高昂授权费用的商业模式，并在 2014 年宣布将虚幻 4 引擎完全免费！有诗为证，"旧时王谢堂前燕，飞入寻常百姓家"。当然，这里引用这句诗略有些不妥，因为虚幻 4 引擎的免费并非走向没落的象征，反而让这款强大而优雅的引擎迎来了新的春天。

好了，闲话不多扯，接下来让我们开始学习虚幻 4 引擎。

笨猫学编程QQ群：375143733
知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

对于新手来说，使用虚幻 4 引擎开发游戏并没有想象中的那么难。特别是虚幻 4 中引入的蓝图可视化编程系统让游戏开发变得更加容易上手。对于某些类型的游戏，我们甚至可以不写一行代码就创建一款完整的游戏。

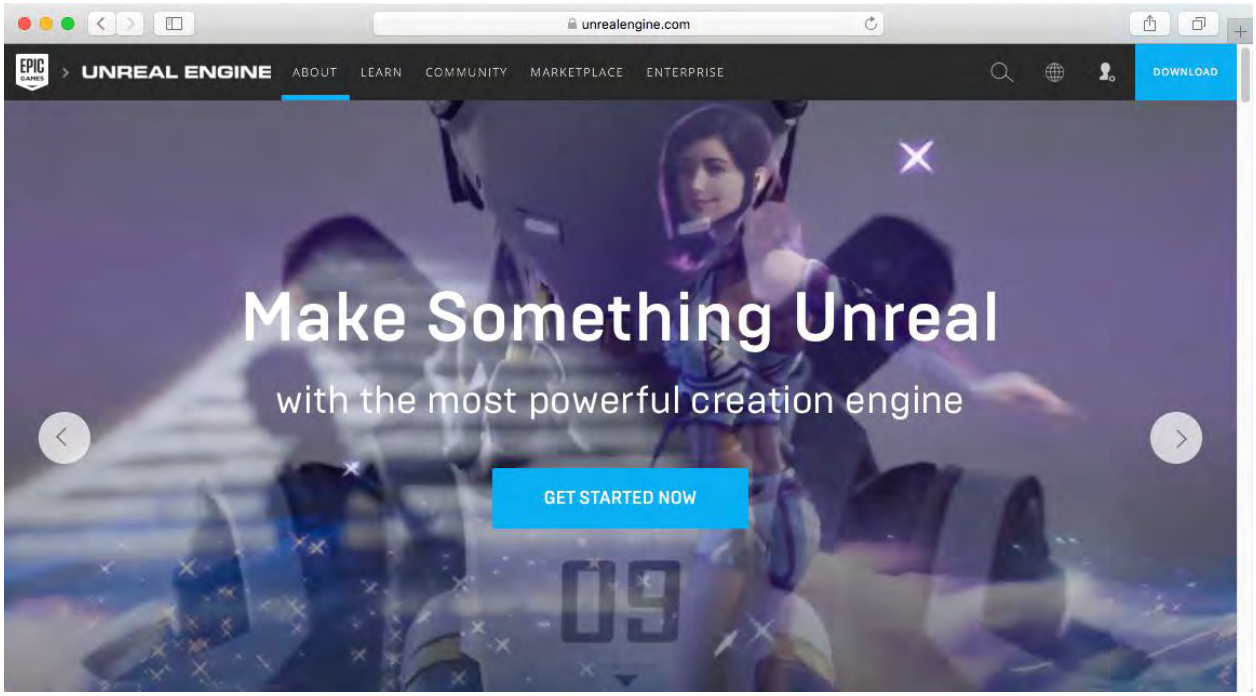
在 Part 1 的教程中，我们主要学习如何使用虚幻 4 引擎的蓝图系列来开发一款简单的游戏。Part 1 的教程又可以分为以下 7 个部分：
1.开始前的准备
2.蓝图系统
3.材质
4.UI
5.如何创建一款简单的游戏
6.动画
7.音效

开始前的准备

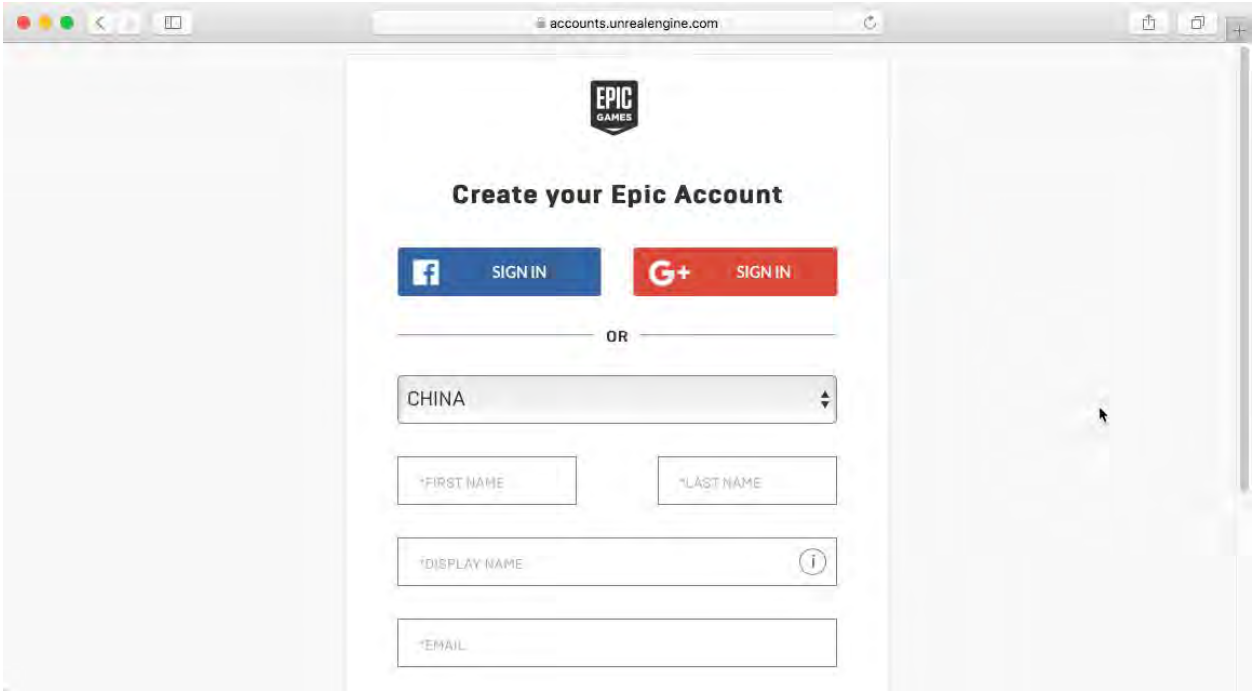
这个小节的内容主要是为了让大家对虚幻 4 引擎有个基本的概念，我们将要学习如何安装虚幻 4 引擎，如何导入游戏资源，如何创建材质，以及如何使用蓝图系统创建具有基本功能的游戏对象。

安装虚幻 4 引擎

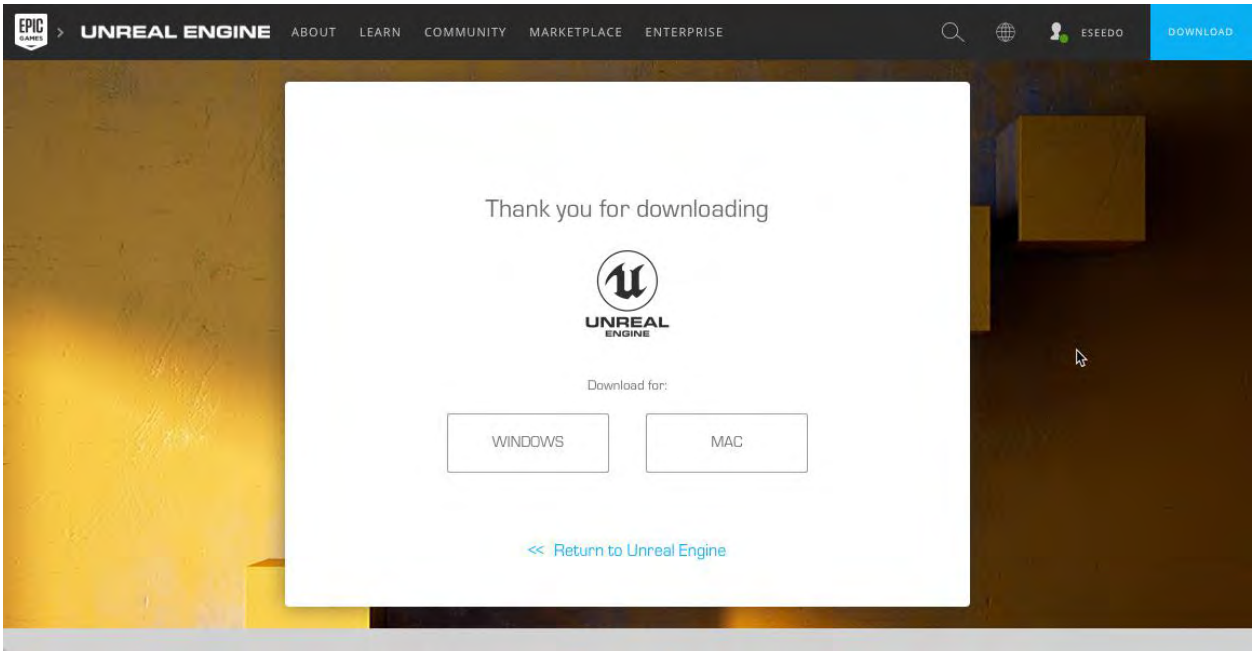
要安装虚幻 4 引擎，我们需要借助 Epic Games Launcher。在浏览器中打开 Unreal Engine 的官网 (https://www.unrealengine.com)，然后点击页面右上角的 DOWNLOAD 按钮。



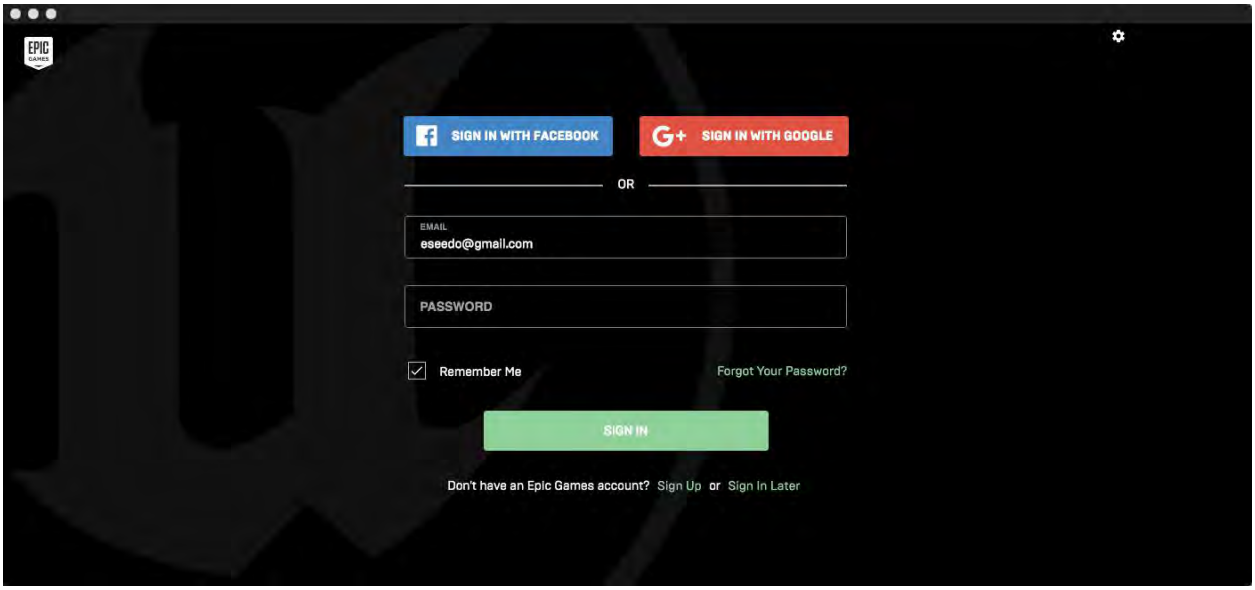
在下载之前，我们需要创建一个 Epic 账号，或者也可以使用 Facebook 和 Google 的账户来登陆。



在创建账户并登陆后，就可以选择下载 Windows 或 Mac 版的 Launcher 了。

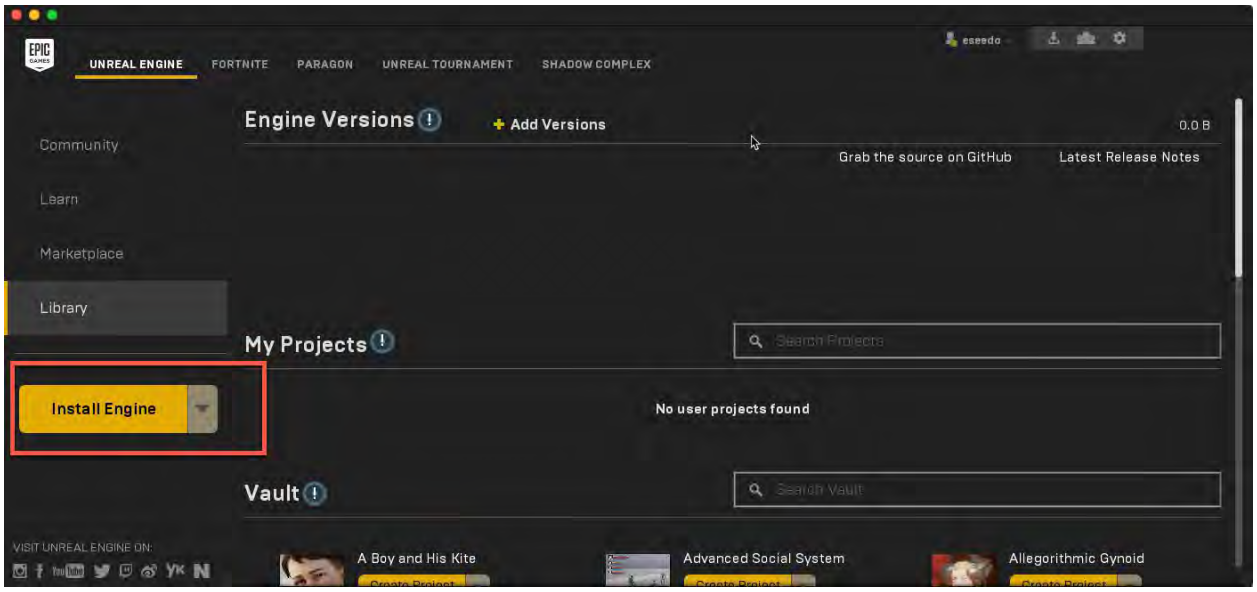


下载并安装完成之后，打开 Launcher，可以看到如下的界面。

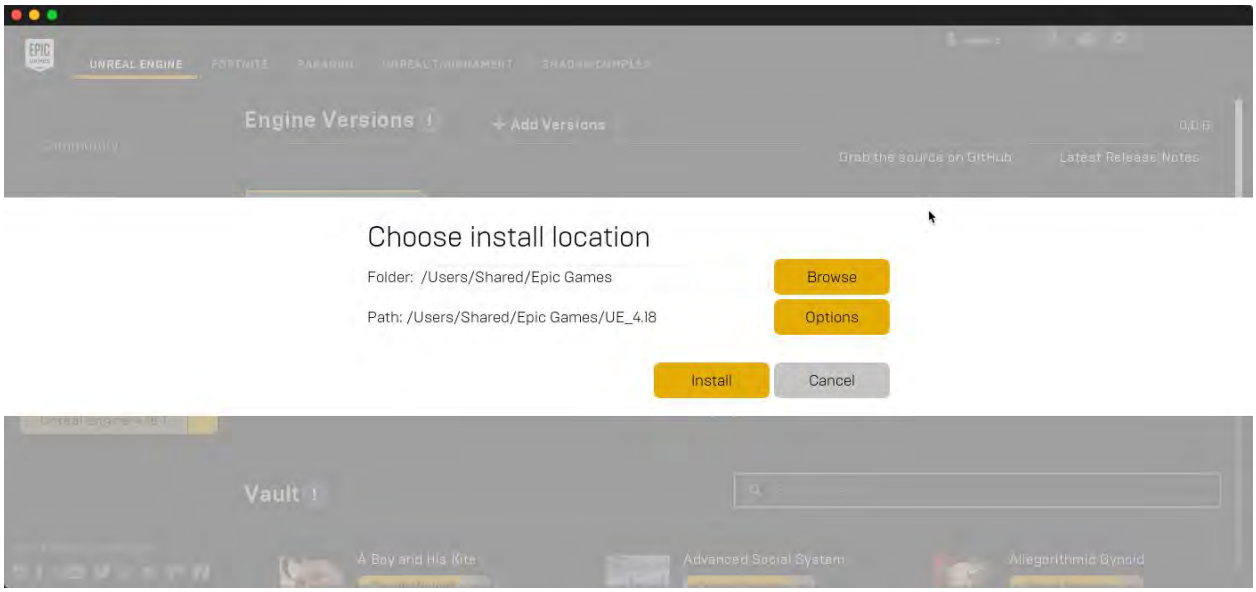


使用自己的 Epic 账号登陆即可。

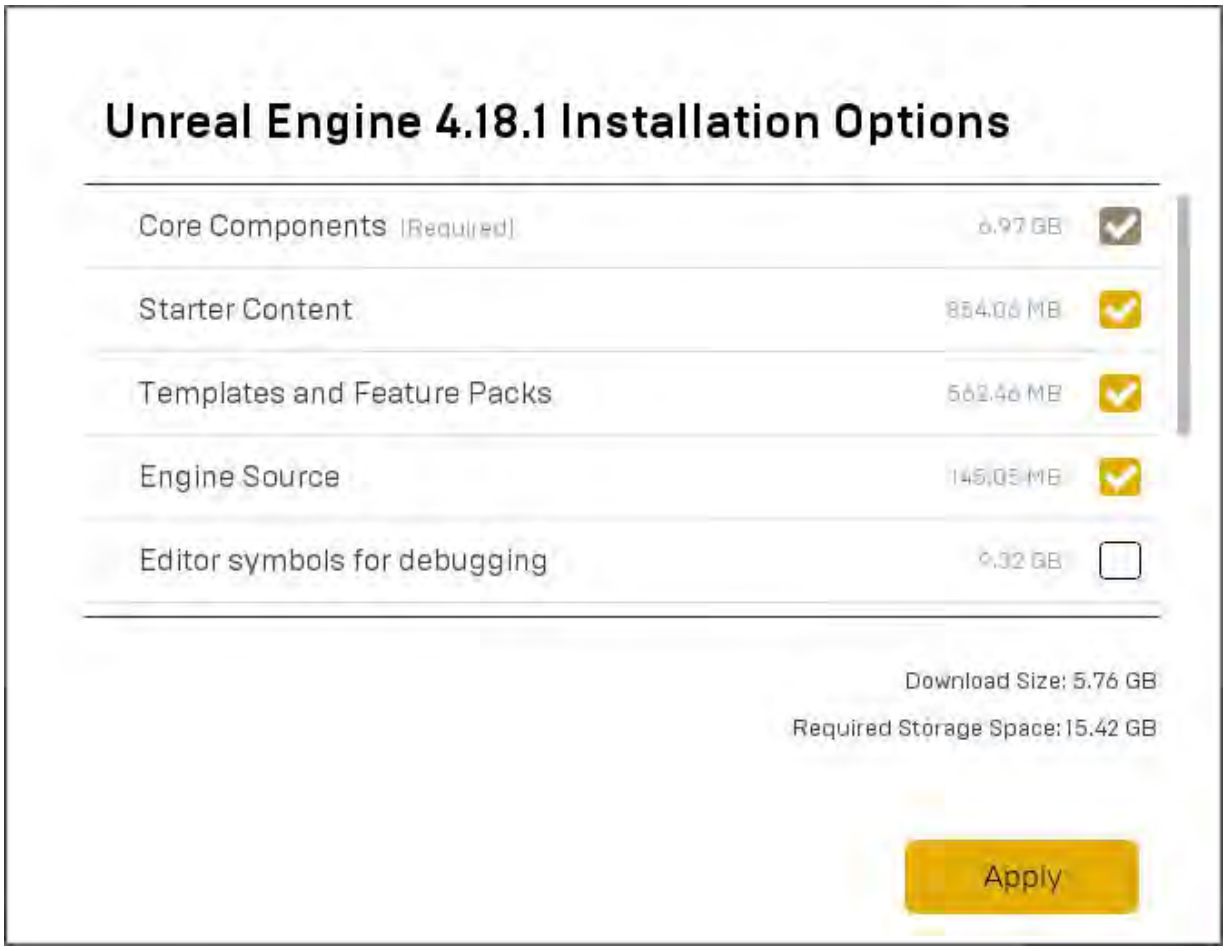
此时会看到类似下面的界面



点击 Install Engine 开始安装虚幻 4 引擎，会看到提示选择安装路径，使用默认的就好，



点开 Options，会看到更多的安装选项，默认情况的安装选项如下图。
其中 Core Components 是必选项，也是引擎最核心的部分。



接下来简单介绍了三个默认选项的内容：
（1）Starter Content
其中提供了初学者用来开发自己项目的免费资源，包括一些模型、材质等等。在项目的开发过程中，我们可以临时使用这些资源来代替最终的游戏资源。

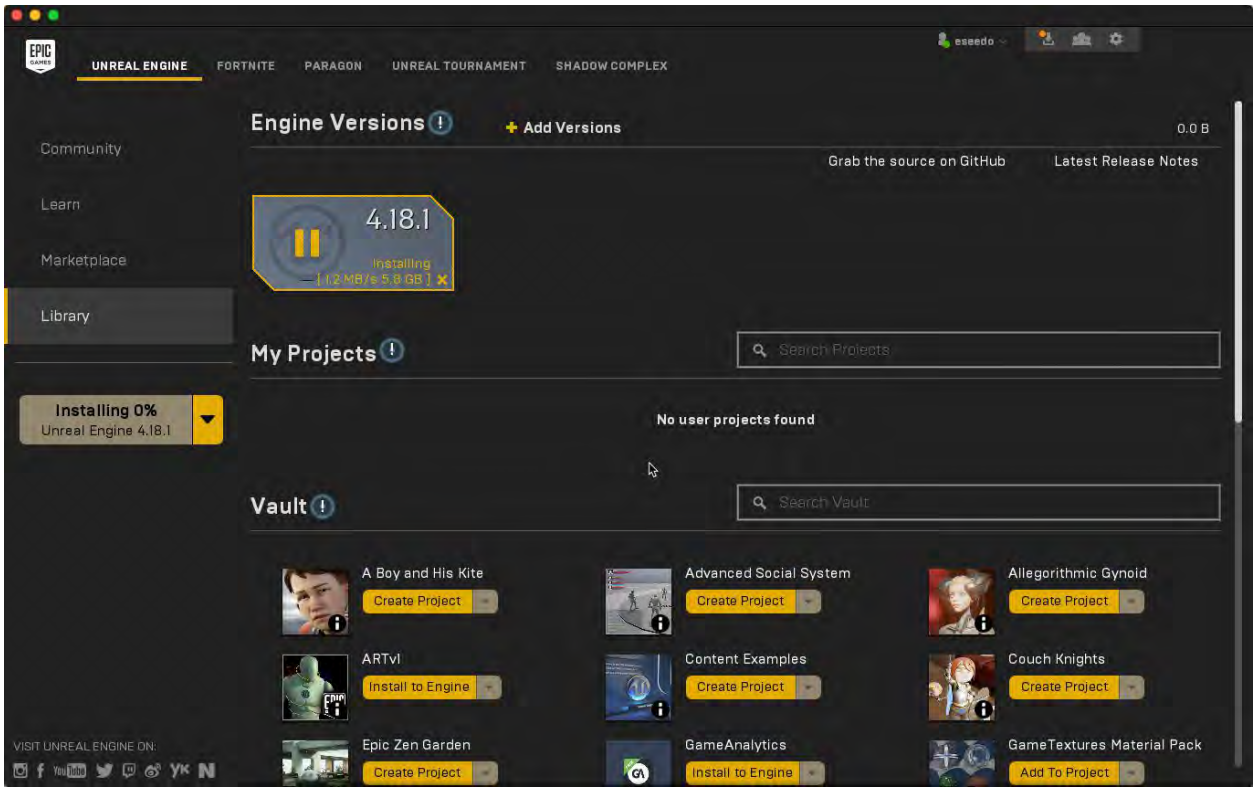
（2）Templates and Feature Packs
虚幻 4 引擎提供了一系列的模板，在开发项目的时候，根据开发者所选的模板类型，项目模板中会提供一些基本的功能特性。比如，当我们选择 Slide Scroller 模板来创建游戏的时候，系统就会帮我们创建一个基本的横版过关游戏模板，其中包含了一个游戏角色，角色的基本运动，以及一个固定的 plane camera。

（3）Engine Source
需要特别强调的是，虚幻 4 引擎是完全开元的，也就是说只要你愿意，可以阅读和修改引擎的任何一行代码。因此，如果我们需要在编辑器中添加自定义的按钮或其它元素，理论上只要修改相关的源代码就好。

当然，除了这三个默认的选项，虚幻 4 引擎还提供了其它的一些选项，比如在 Engine Source 之下有一个 Editor symbols for debugging，而且要占用 9.32GB 的空间。那么它的作用是什么呢？简单来说，如果你的项目需要用到 C++语言，那么勾选该选项后将可以在 Visual Studio 中为项目的 C++代码创建断点，以及其它调试所需要的特性。
另外，如果我们在首次安装时没有勾选该选项也完全不用担心。在 Epic Games Launcher 中点击所安装引擎的版本号，会出现一个下拉菜单，我们可以勾选 Editor Symbols for Debugging，点击 apply 就可以安装了。当然，在安装完成后需要重启引擎。
考虑到我们 part1 的内容将完全使用蓝图系统，所以这里就暂时不需要勾选该选项了。

接下来继续滚动滑动条，可以看到所支持的不同平台。默认情况下系统自动勾选了 IOS，Android 和 HTML5，如果我们确定不会在某个特定的平台开发，比如 HTML5，那么可以禁用即可。
这里我们全部保持默认选择，点击 Apply 就可以开始安装了。
此时系统会回到刚才选择安装路径的界面，在该界面中点击 Install 开始安装。
接下来就是漫长的等待，安装的速度直接取决于网速。这里必须吐槽一下我这个破电信网络，下载速度简直是不忍直视，直接消耗了大量的生命。

顺便说明一下，虽然我这些操作是在 Mac 系统下进行的，但是跟 Win10 系统上并没有什么太大的区别。
另外需要特别说明的是，和 Unity3d 引擎类似，UE4 引擎的更新速度也很快，所以如果你在这里看到的版本号不同，并没有太大的关系。



只要你安装的引擎版本是 4.18.1 及其以上的，那么至少在相当一段时间里面（3-6 个月），教程里面的各种操作不会发生太大的变化。

安装完成之后，会看到下面的界面，点击 Launch 就可以打开引擎了。

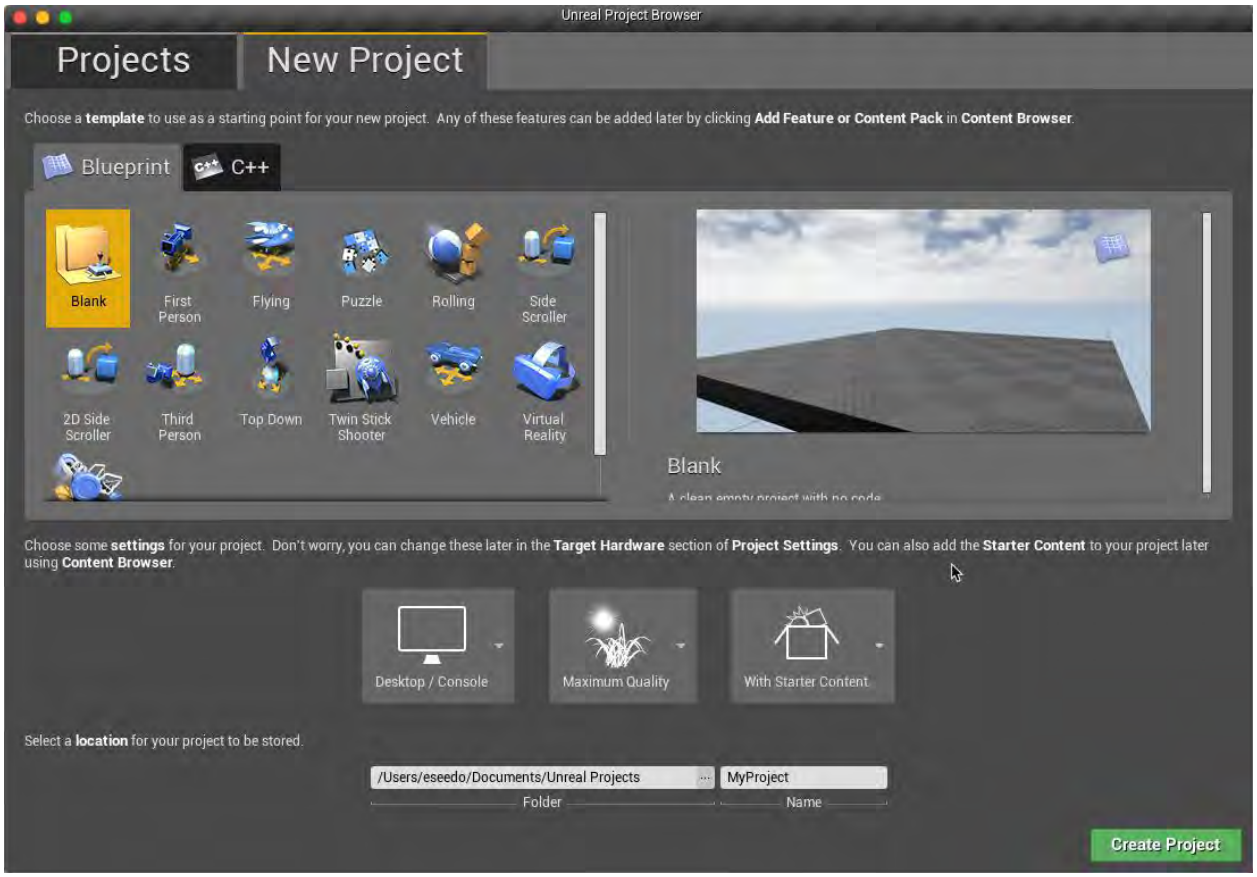
需要特别说明的是，正常情况下，我们都是通过 Epic Games Launcher 来打开引擎的。

创建新项目

接下来我们将创建自己的第一个虚幻 4 游戏项目。

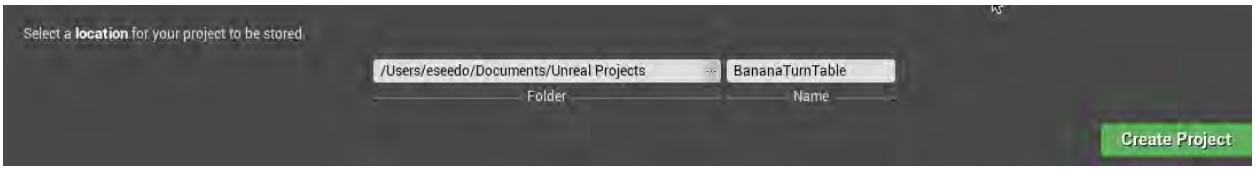
在等待引擎打开之后，首先我们会看到以下界面。

点击 Blueprint 选项卡，这里提供了多个模板可供使用。不过考虑到我们要从零开始学虚幻 4，因此这里选择 Blank。在界面下面的部分会看到有几个其它选项设置，



这里介绍一下每个选项的意思：
1.最左侧的用来选择目标硬件平台：
如果选择 Mobile/Tablet，那么就会禁用一些后期处理特效，同时还会在开发时启用鼠标作为触摸输入。这里我们将其设置为 Desktop/Console。
2.中间的用来设置画质。
如果选择 Scalable 3D or 2D 选项，那么就会禁用一些后期处理特效。这里我们选择默认的 Maximum Quality（最佳画质）。
3.是否包含默认提供的游戏资源。
为了简单起见，这里我们将其设置为 No Starter Content。

在界面的最下方，我们可以选择项目存放的路径，以及项目的名称，这里将其命名为 BananaTurntable。



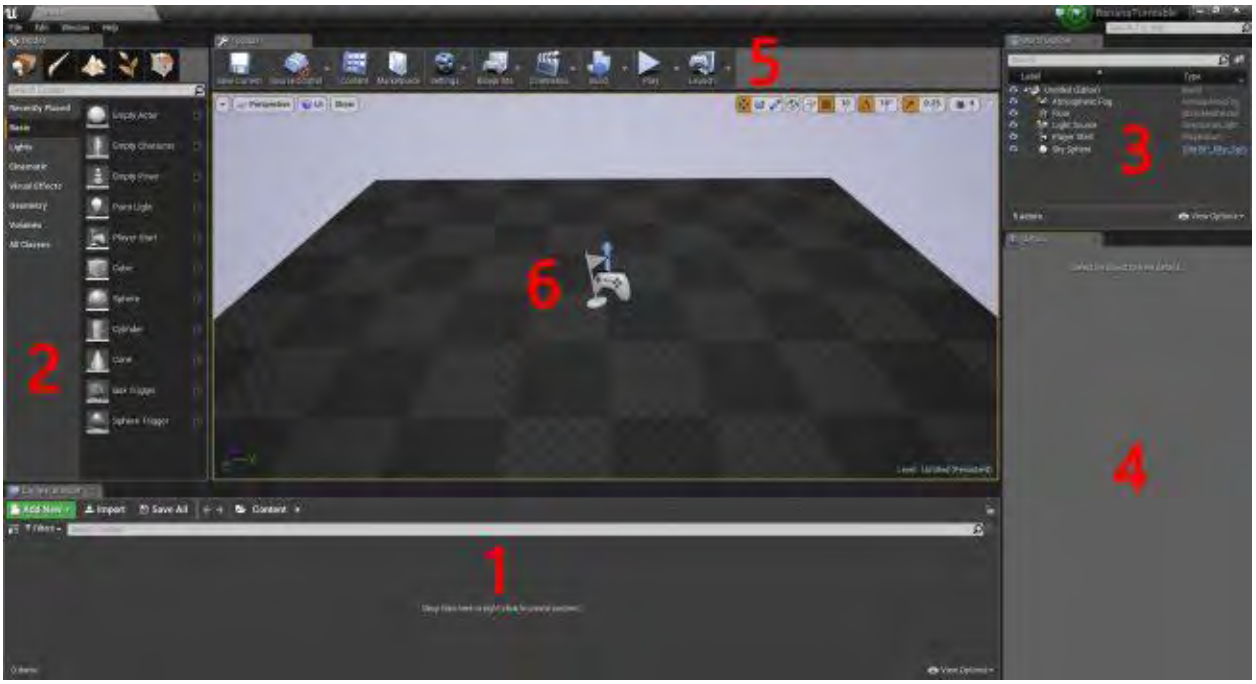
如果需要更改存放路径，我们可以点击文件路径后的三个点。

需要说明的是，项目名称跟最后游戏的名称不是一回事，所以我们不必担心这一点。

输入完成后，点击 Create Project 即可。

玩转虚幻 4 编辑器的用户界面

一旦我们创建完项目，编辑器就会自动打开，可以大致把它分为 6 个部分：



1.Content Browser:

这个非常类似于 Unity3d 中的 Project 视图，里面显示了所有的项目文件。我们可以在这里创建新的文件和文件夹，并管理所有的项目文件。此外，我们也可以使用搜索栏或者过滤器来找到所需的文件。

2.Modes

从这里我们可以选择一些有用的工具，比如 Landscape Tool 和 Foliage Tool。默认情况下这里显示的是 Place Tool，我们可以使用它来将多种不同类型的游戏物体放入场景中，比如灯光、摄像机等等。

3.World Outliner

这个比较类似于 Unity3d 的 Hierarchy 视图，我们可以在这里看到当前游戏场景中的所有游戏对象。

4. Details

这个有点类似 Unity3d 中的 Inspector 面板，这里会显示我们所选择的游戏对象的属性。在这里，我们可以编辑游戏对象的一些属性设置。不过需要注意的是，对游戏对象的属性设置只会改变该

对象的实例。例如我们有两个球体，当我们改变了其中一个球体的大小时，只会影响所选择的球体对象，而不会影响另外一个球体。

5.Toolbar

工具栏中提供了一系列的功能，我们最常用到的就是 Play。

6.Viewport

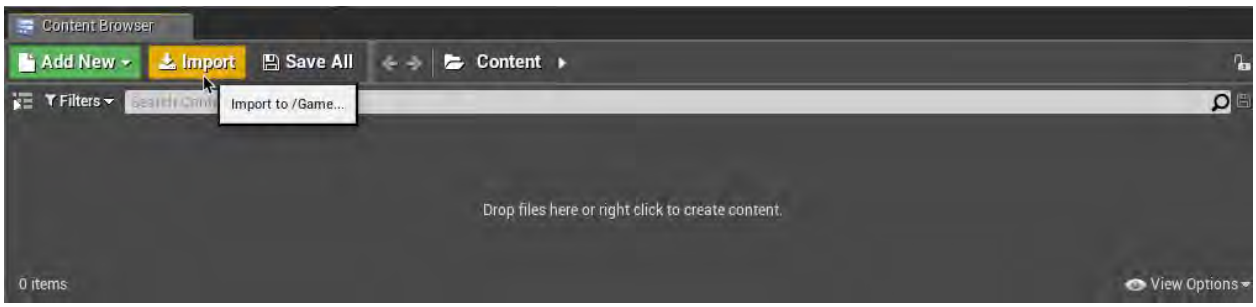
这里用可视化的形式显示了当前的游戏场景，我们可以单击右键拖动鼠标查看场景。如果想要在查看的同时移动，那么只需要按住鼠标右键，然后使用键盘上的 WASD 键就好。

导入所需的游戏资源

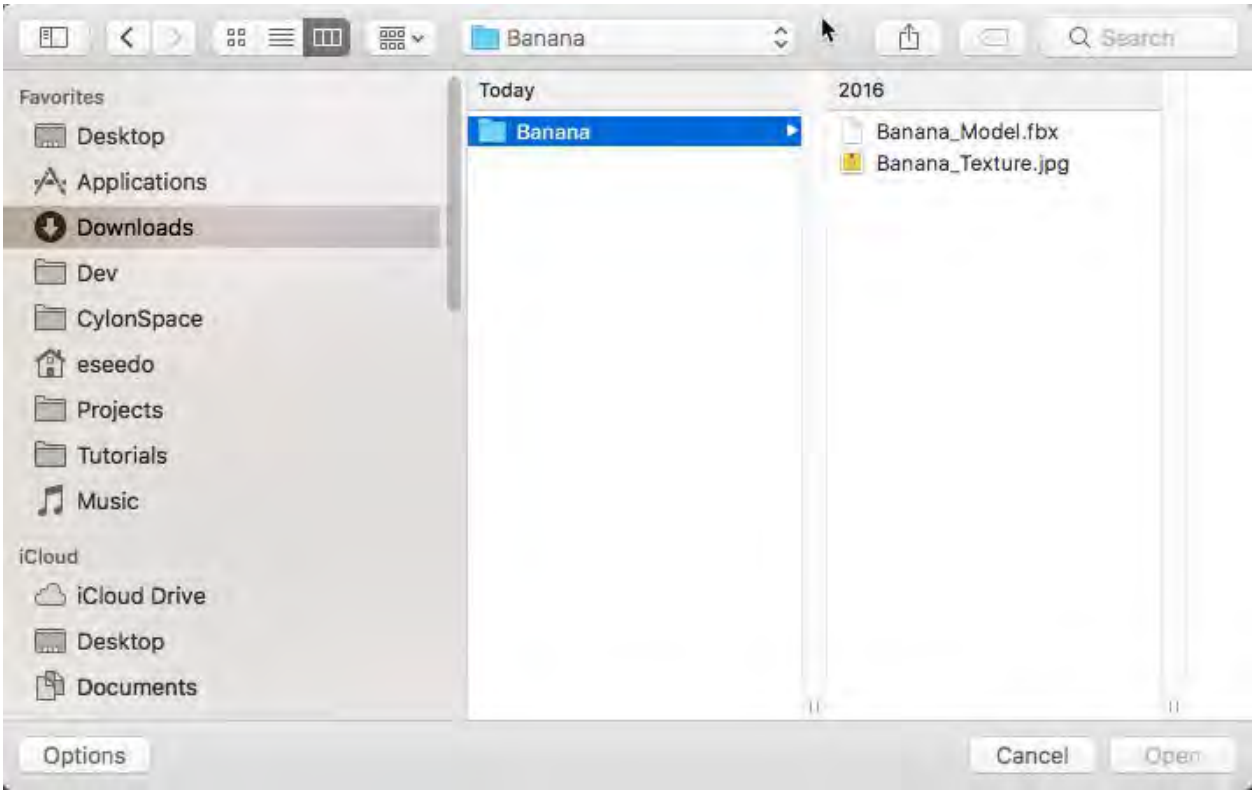
如果没有相关的游戏资源，我们的游戏必然是苍白而又乏味的。在这里下载香蕉的模型(链接:https://pan.baidu.com/s/1pLMbejT 密码:ocz8)，其中包含了两个文件，分别是 Banana_Model.fbx 和 Banana_Texture.jpg。当然，如果你本人会 3D 美术设计，那么完全可以使用自己的游戏模型。

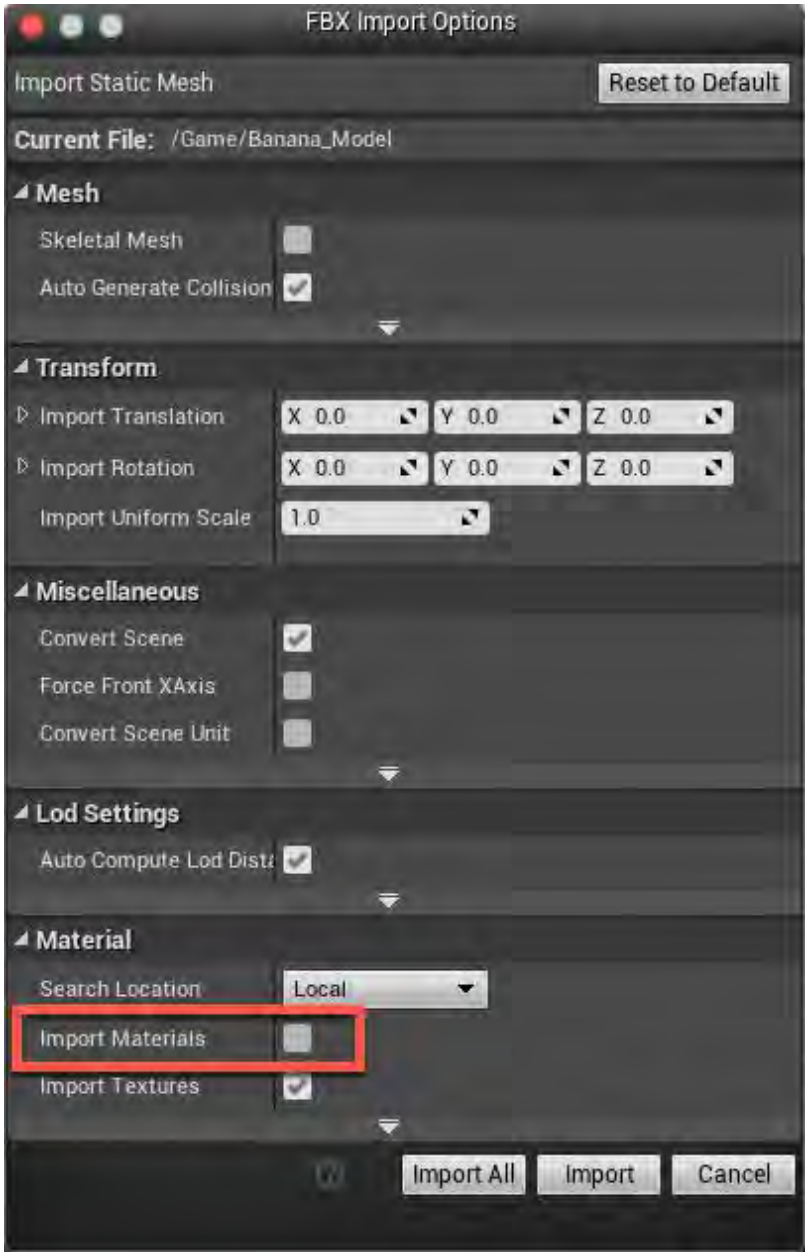
下载完成之后，接下来我们需要将文件导入到虚幻 4 引擎中。在编辑器中找到 Content Browser，然后点击 Import。

然后找到刚才所下载的两个文件，选中相关文件，然后点击 Open 即可。

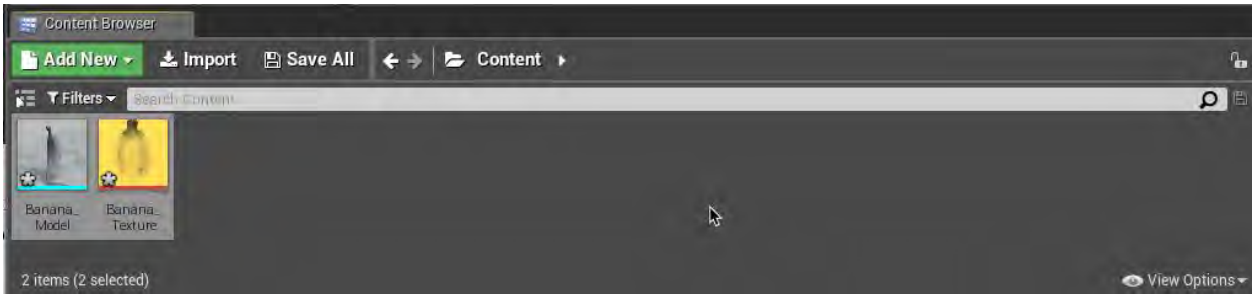


此时虚幻会提供 FBX 文件导入的相关选项，确保这里要取消对 Import Materials 的勾选，因为我们将创建自己的材质。其它选项保持默认即可。

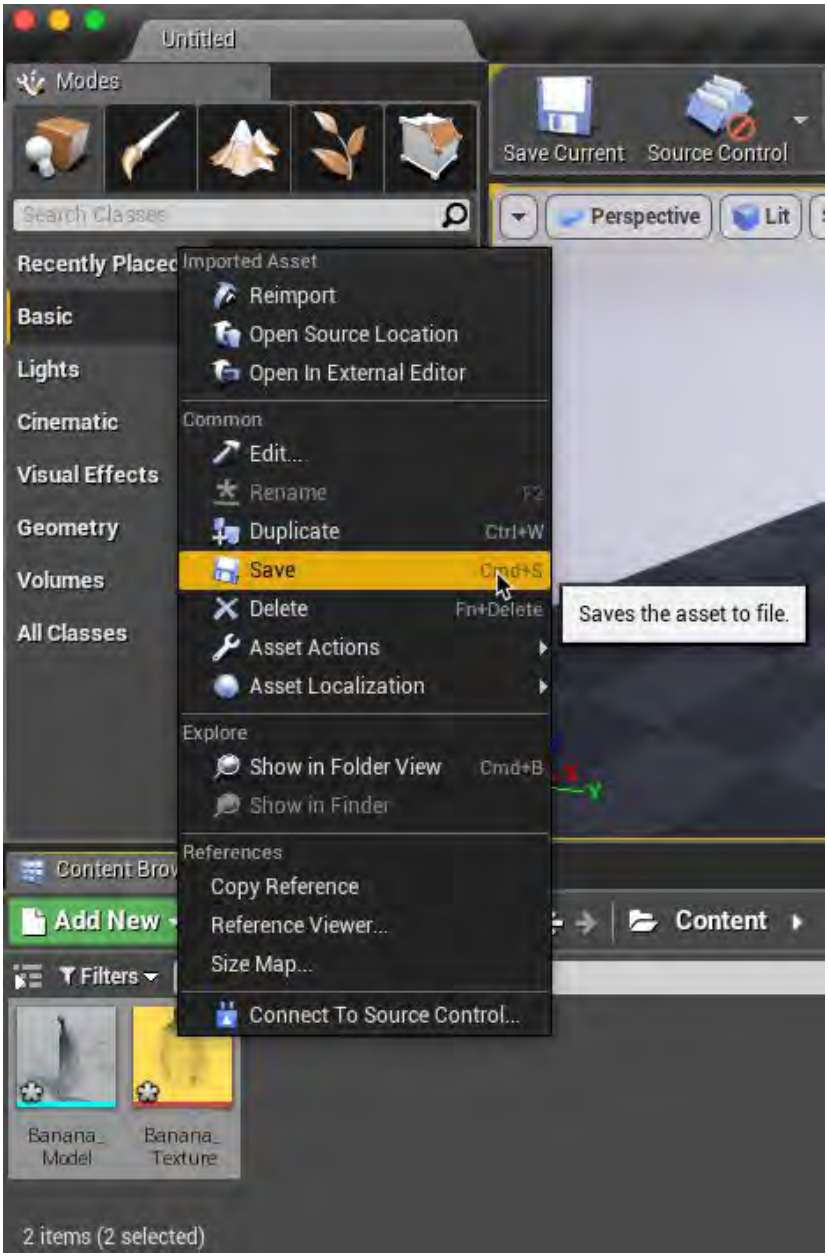




设置完成后，点击 Import 即可。此时可以在 Content Browser 中看到所导入的美术资源文件。



需要注意的是，当我们导入一个文件时，有些时候引擎并不会帮我们自动将其保存到项目中。我们可以右键点击所导入的文件，然后选择 Save。



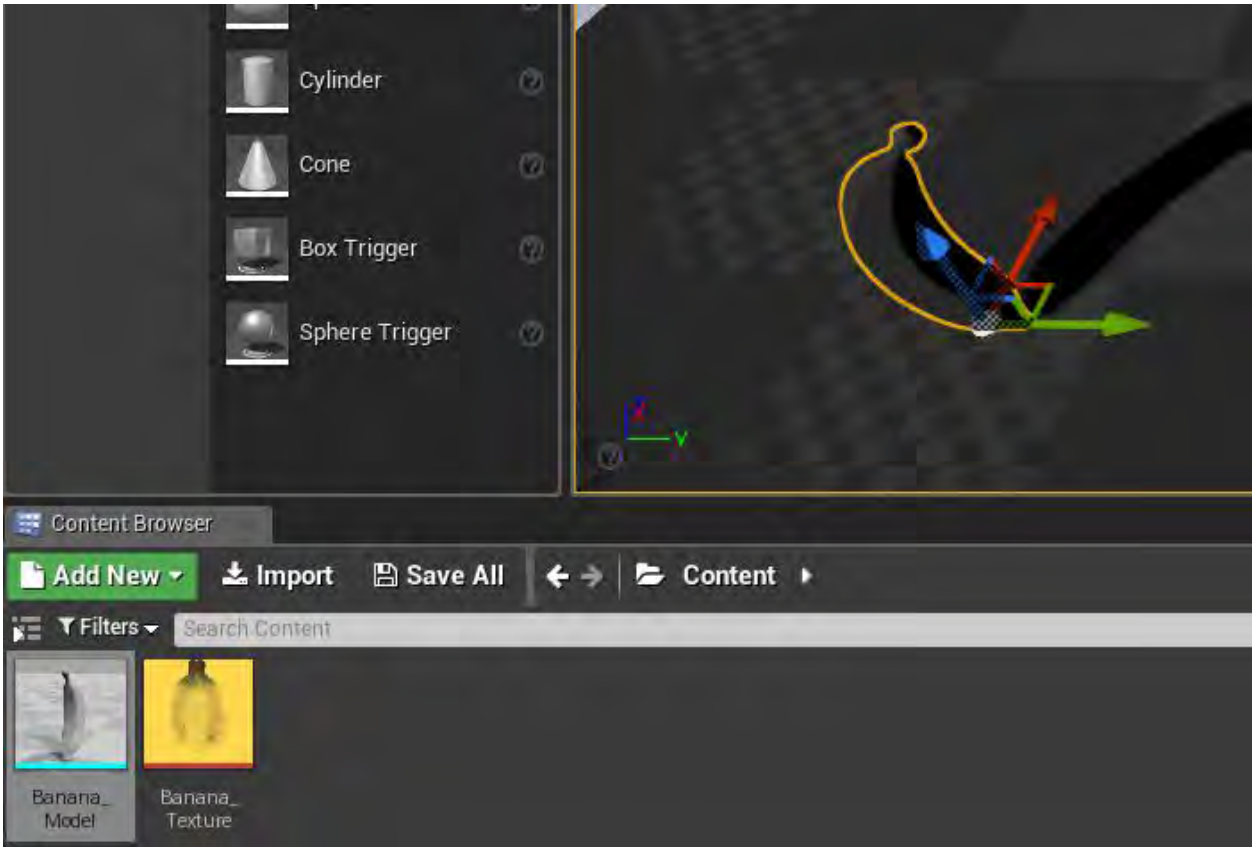
当然，更保险的方法是从菜单栏中选择 File-Save All。

另外有一点要注意的是，在虚幻 4 引擎中，3D 模型又被称为 meshes。所以现在我们已经有了香蕉的 mesh，那么接下来要做的事情就是将其放入到游戏场景之中。

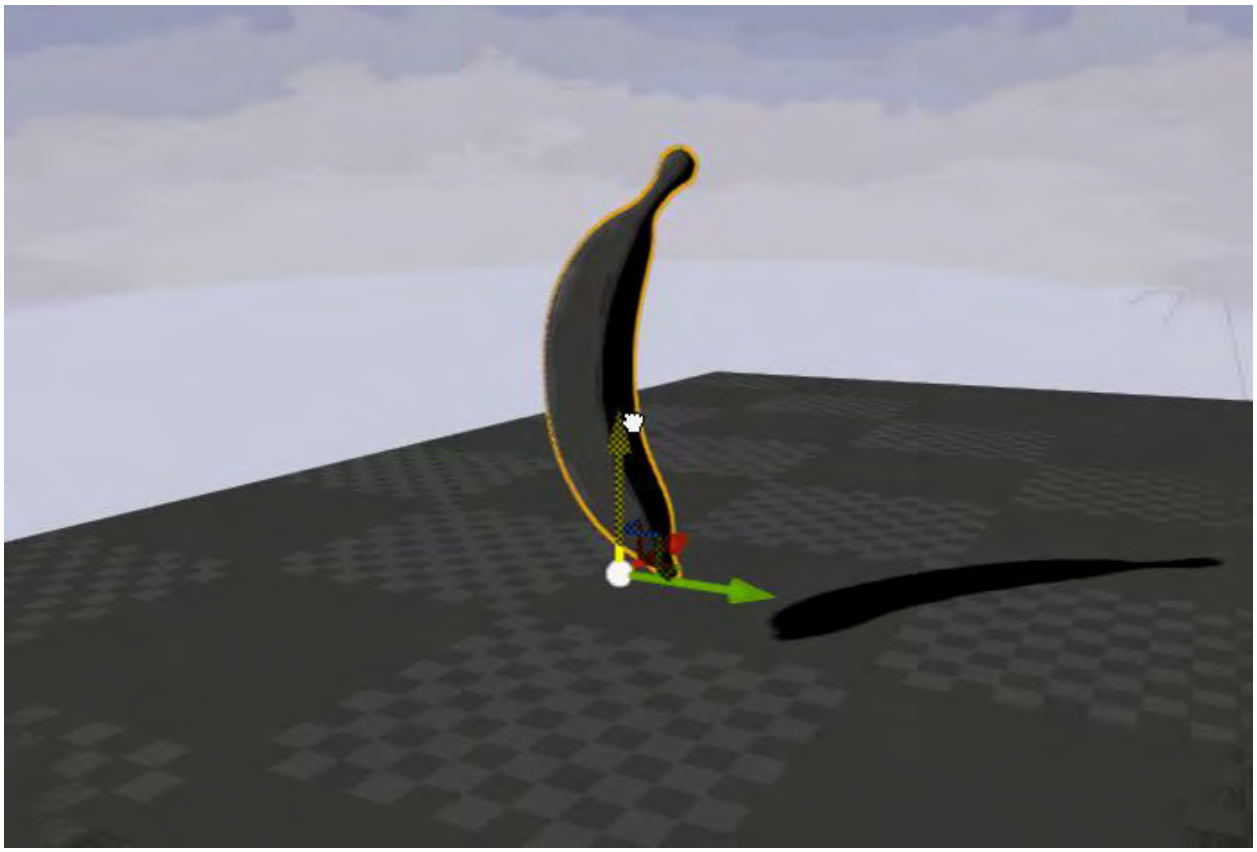
将 meshes 添加到场景之中

当前关卡中的游戏场景看起来空荡荡的，接下来让我们往里面添加内容。

往关卡里面添加模型比较简单，只需要用鼠标左键选中 Content Browser 中的 Banana_Model 文件，然后把它拖曳到 Viewport 中就好。

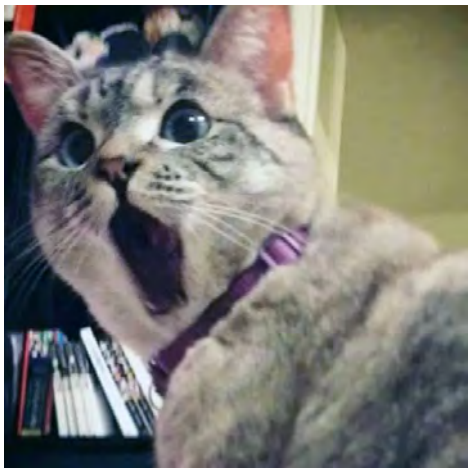


放置到游戏场景之中的游戏对象可以移动、旋转和缩放，对应的键盘快捷键分别是 W,E 和 R。当然，我们也可以直接在游戏对象上面使用鼠标进行操作。



关于材质（Materials）

如果我们仔细观察游戏场景中的香蕉，会发现这根本就不是我们日常所熟悉的黄色香蕉好吗？！难道是传说中的暗黑料理？！



什么是材质（Material）

材质定义了某个物体表面的外观。从最基本的层面来看，材质定义了以下几个方面：

（1）Base Color：

物体表面的基本色彩或材质贴图，我们可以用它来添加物体表面的外观细节，或者色彩的变化。

（2）Metallic:

代表了物体表面的“金属化”程度，或者说表面有多像金属。通常来说，纯金属有最高的 Metallic 数值，而纤维的 Metallic 值则为 0。

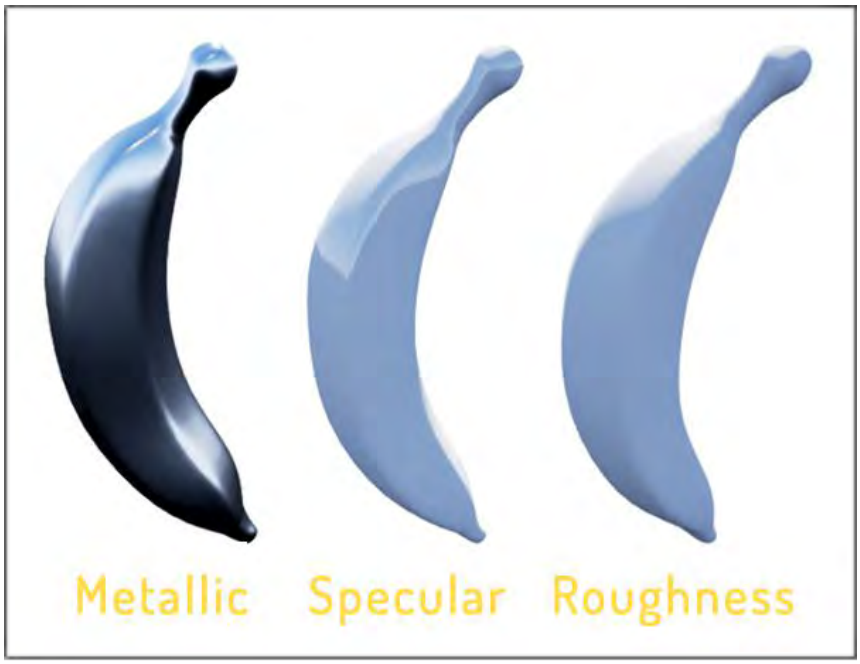
（3）Specular:

代表了非金属物体表面的镜面反射程度。例如，陶瓷物体的表面拥有比较高的 Specular 数值，而粘土的 Specular 数值则较低。

（4）Roughness:

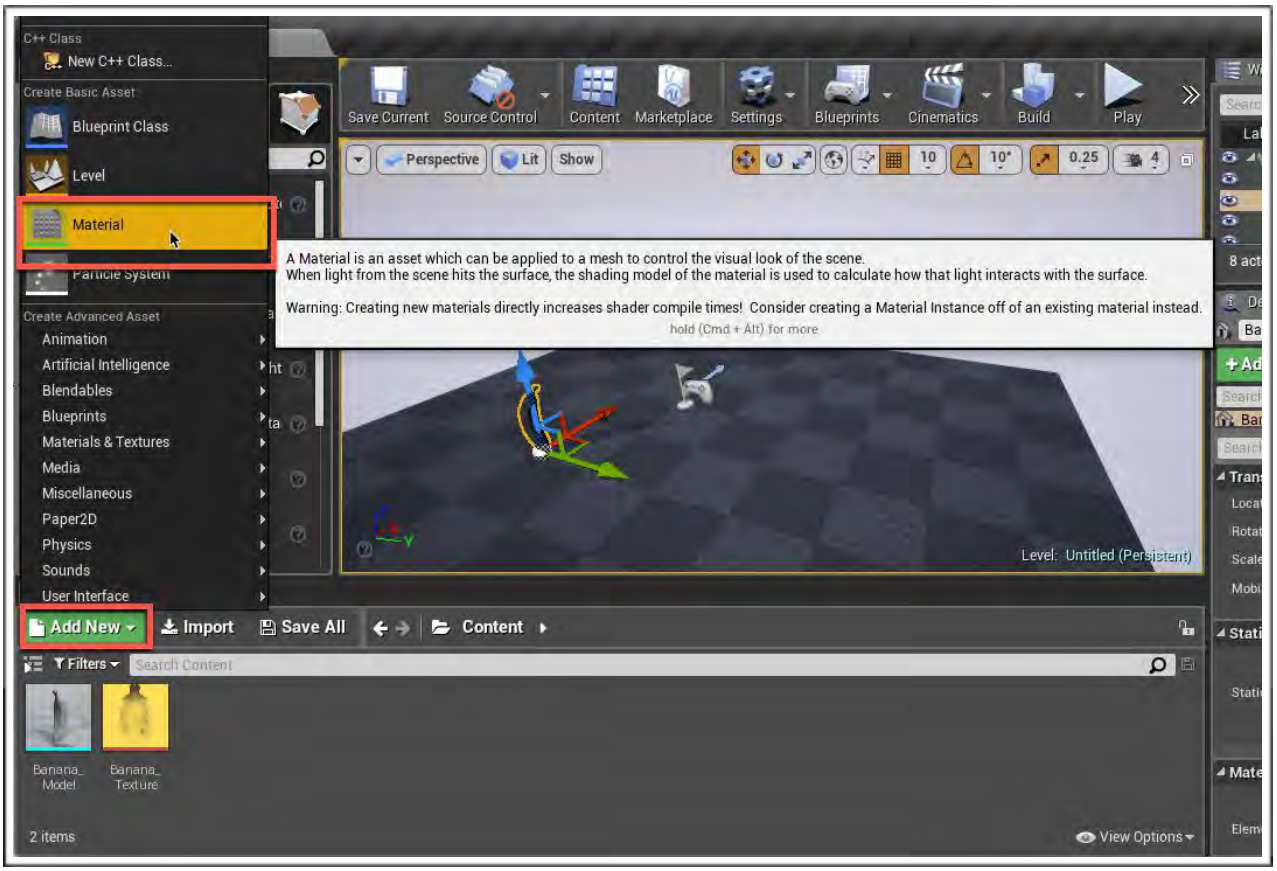
代表了物体表面的粗糙程度，拥有最大 roughness 数值的表面完全不会发亮。这个数值通常用于类似岩石或木头的物体表面。

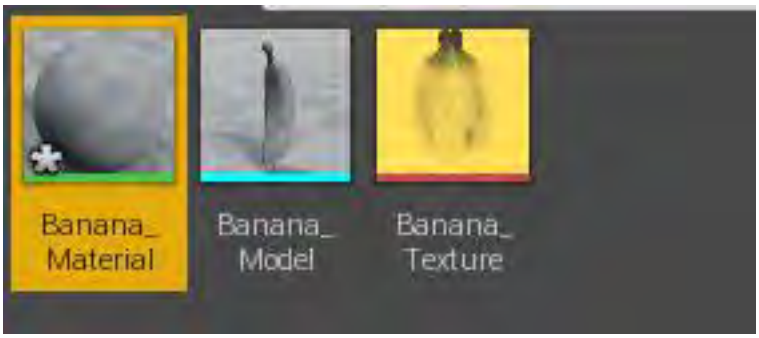
下图显示了三种不同材质的示例。它们有相同的颜色，但是特征不同，每种材质都将其所对应的特征值调至最高，而其它的属性值则被设置为 0。



创建材质

为了创建一种材质，在编辑器中切换到 Content Browser，然后点击绿色的 Add New 按钮。此时会弹出一个菜单，其中显示了我们可以创建的游戏资源列表。点击 Material 即可。

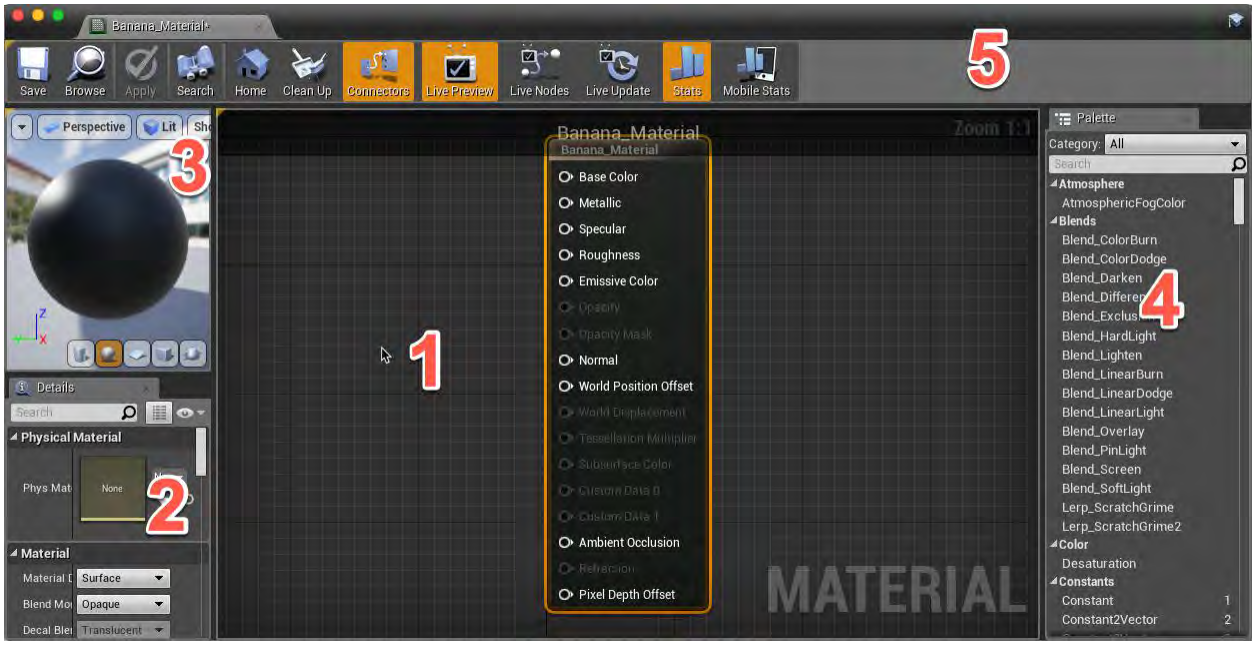




将新创建的材质命名为 Banana_Material，然后双击该文件，就可以在材质编辑器中将其打开。

和材质编辑器(Material Editor)的第一次亲密接触

材质编辑器大概可以分为五个大的区域，如下图所示：



1.Graph 区：

这个面板中包含了材质中的所有节点，以及 Result 节点。我们可以点击右键，然后移动鼠标来平移该视图。此外，还可以通过鼠标滚轮来缩放该视图。

2.Details 区：

此处显示了所选择的任何节点的详细属性。如果没选中任何节点，那么此处将显示材质的属性。

3.Viewport 区：

此处显示了材质的预览效果。我们可以按住鼠标左键，然后移动鼠标来旋转摄像机。同时，我们也可以通过鼠标滚轮来缩放预览效果。

4.Palette 区：

此处显示了材质中可用的所有节点清单。

5.工具栏

这里显示了设置材质所用的相关工具。

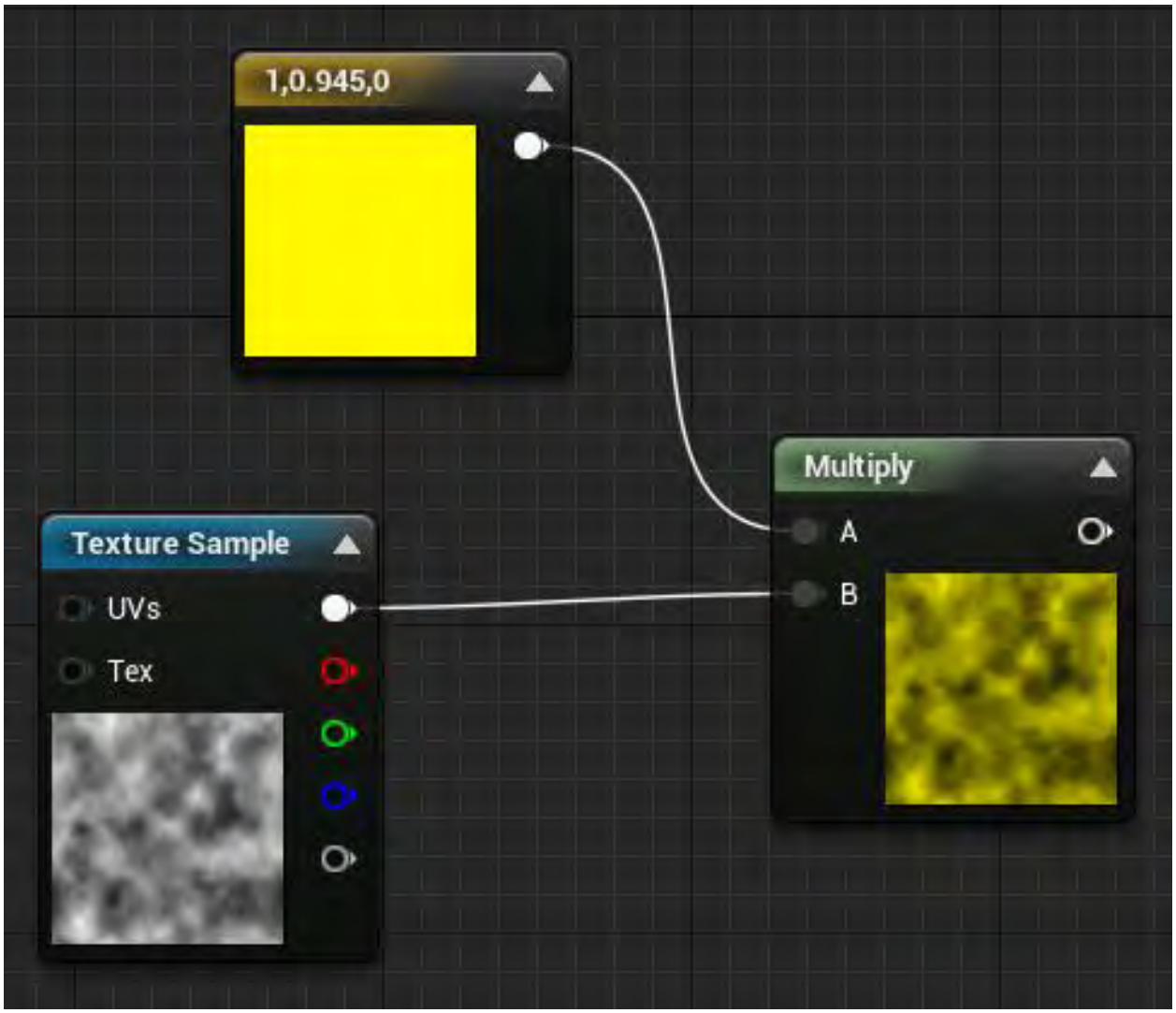
什么是 Node(节点）？

在开始创建我们自己的材质之前，需要了解用于创建材质的对象：nodes（节点）

在虚幻 4 中，材质主要是由节点组成的，我们可以选择多种不同类型的节点，而这些节点也具备不同的功能。

节点可能有自己的输入和输出，输入在节点的左侧，而输出则在节点的右侧。这些输入和输出点使用带箭头的原点来表示。

下面这个图显示了如何使用 Muliply 和 Constant3Vector 节点为某个材质添加黄色。



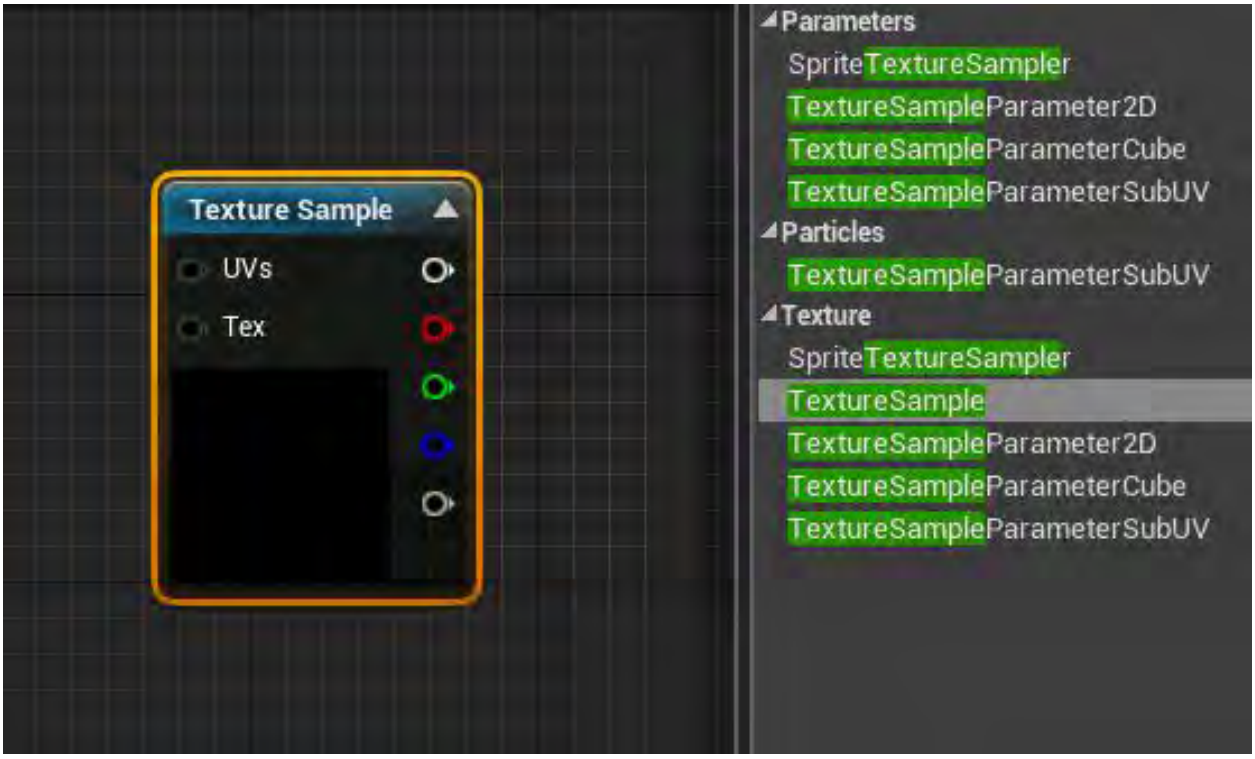
材质有一个特殊的节点，名为 Result 节点，在我们这个例子中就是 Banana_Material。这个节点也是所有节点的终点。我们在这个节点中所插入的东西将决定材质最终的外观。

添加 Texture（材质贴图）

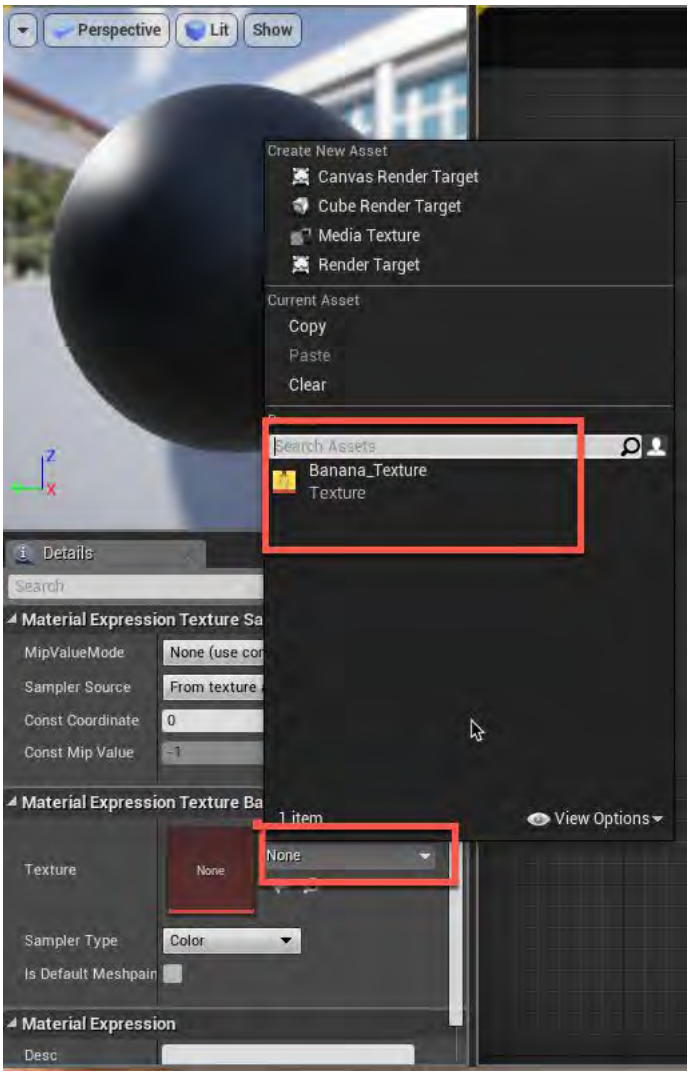
为了给模型添加色彩和外观细节，我们需要 texture(材质贴图）。texture 本质上是一个 2D 突破。通常情况下，我们会将材质贴图投射到 3D 模型的表面，从而让其具备色彩和外观细节。

为了给　模型添加材质贴图，我们需要用到之前所添加的 Banana_Texture.jpg。在虚幻 4 中，我们可以使用 TextureSample 节点来给材质添加贴图。

在材质编辑器右侧的 Palette 面板中搜索 TextureSample，左键单击选中它，并将其拖动到 graph 区。



接下来让我们选择需要用到的贴图。首先选中刚才的 TextureSample 节点，然后在 Details 面板中点击 Texture 右侧的下拉列表，从中选择 Banana_Texture 即可。



为了在材质效果预览处看到贴图的效果，我们需要把刚刚设置的贴图连接到 Result 节点中。左键点击 TextureSample 节点的白色输出端口，从那里拖曳出一条线到 Banana_Material 的 Base Color 输入端口。

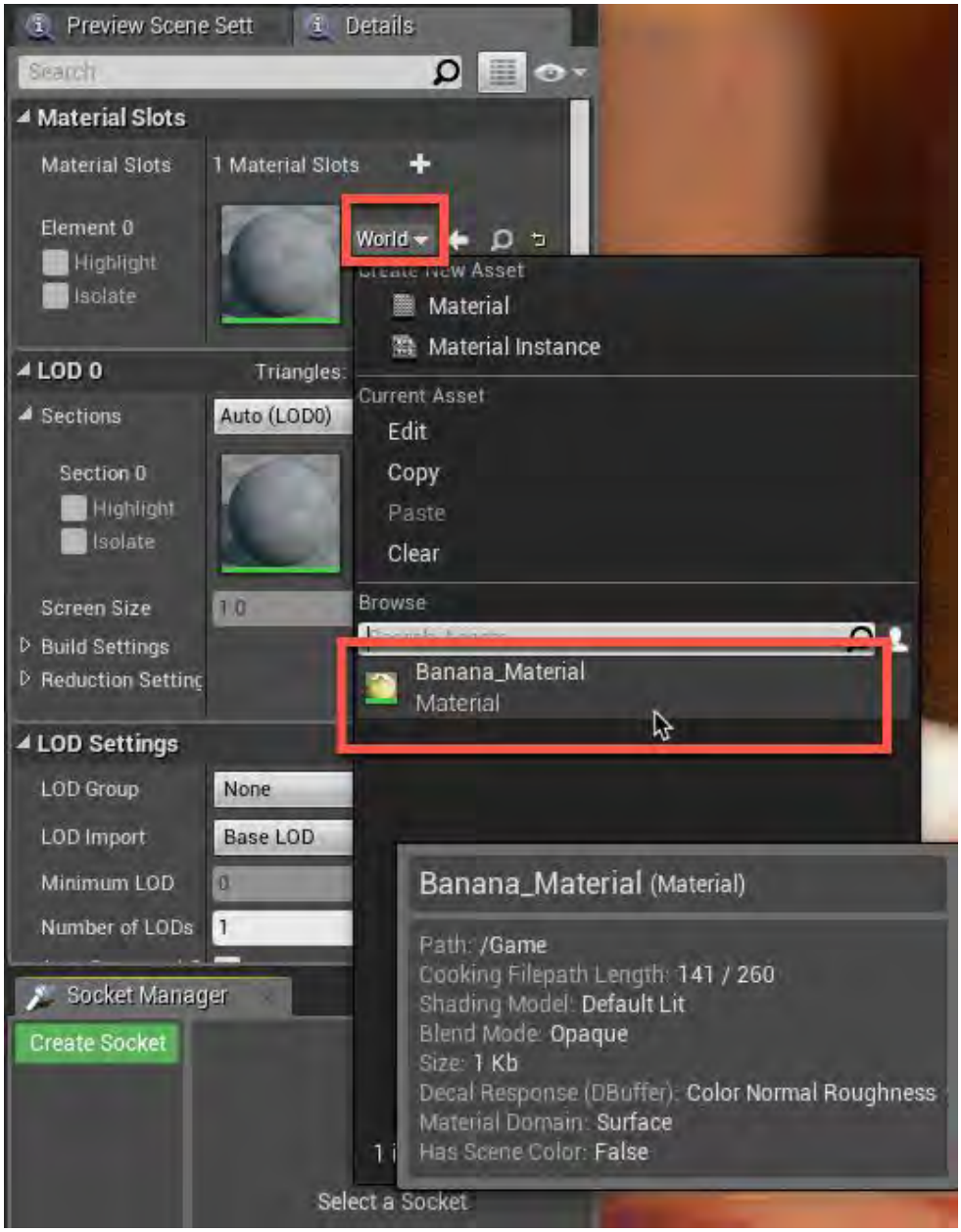
等待一会儿，就会在 Viewport 中看到材质的预览效果了。按住鼠标左键不放并拖曳可以旋转预览效果，从而查看其它的细节。

点击工具栏上的 Apply 按钮来更新材质，然后关闭材质编辑器即可。

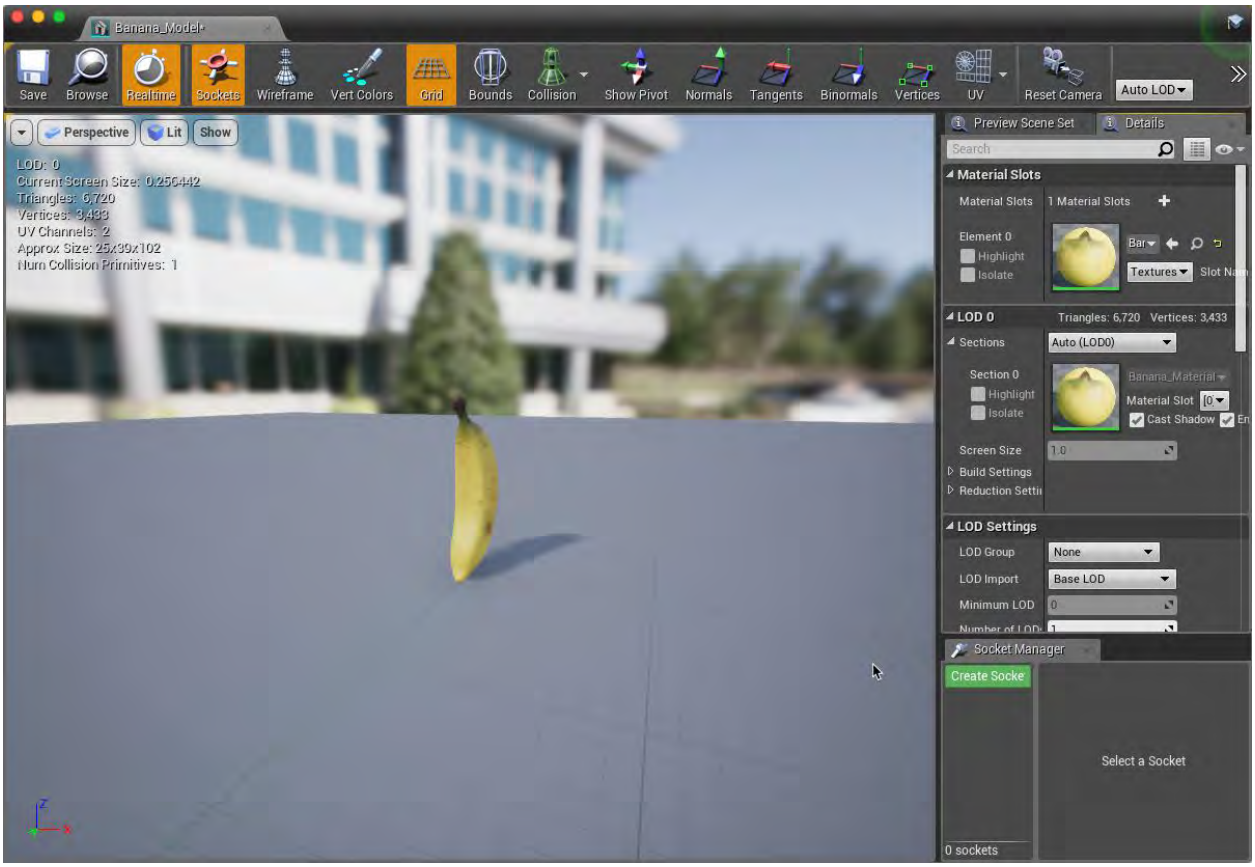
使用所创建的材质

接下来我们将把刚刚创建的材质应用到香蕉模型上。回到虚幻 4 编辑器的 Content Browser 区，双击 Banana_Model，会打开类似下面的编辑器：

在 Details 面板找到 Material 部分，点击 Element 0 右侧的下拉列表，从中选择 Banana_Material。

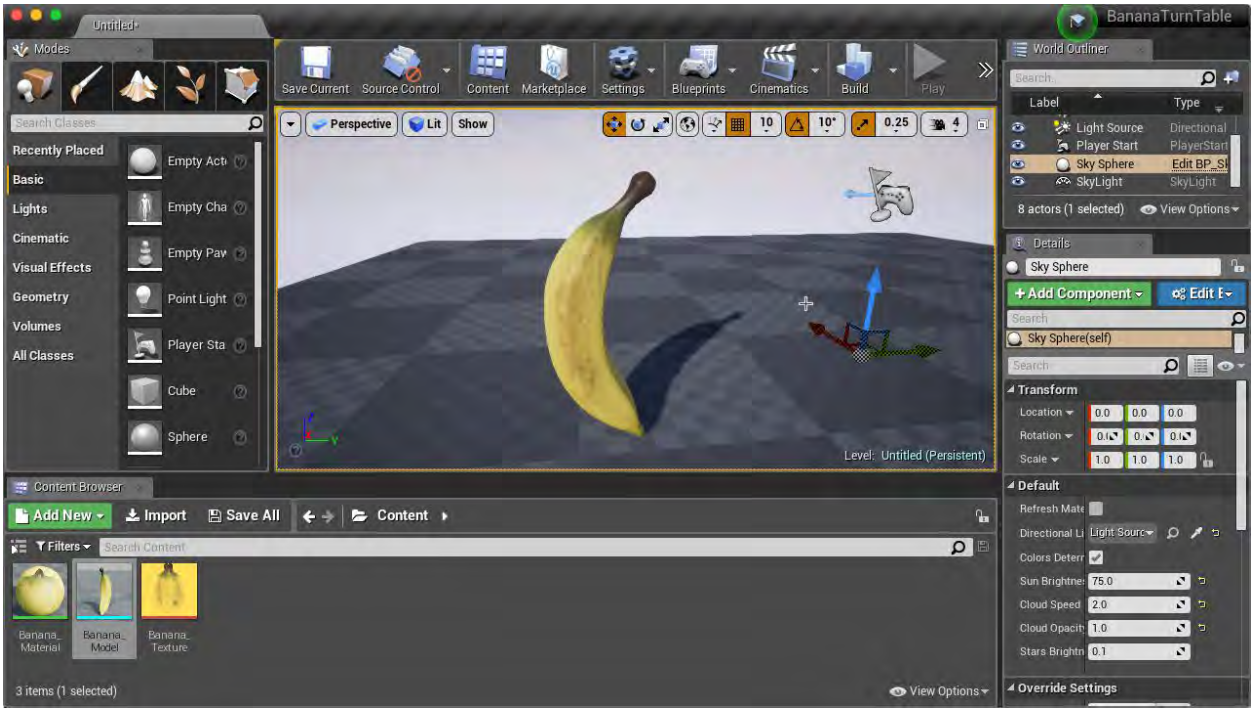


此时会看到预览区的　已经非常接近真实了。



关闭模型编辑器，返回虚幻 4 的主编辑器，并查看 Viewport 区。此时，你会看到场景中的香蕉已经具备一个贴图了。恭喜你，现在你已经具备了成为一个关卡设计师的所有所需的基本技能~

注意：如果你觉得环境中的光照太暗，那么可以在编辑器中的 World Outliner 视图中找到 Light

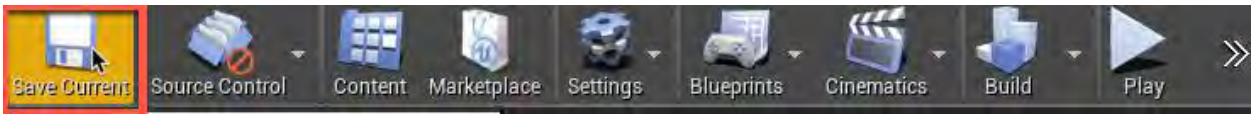


Source，在 Details 面板中找到 Intensity 属性，并将其设置为一个更高的数值即可。

在结束今天的课程之前，让我们先保存并关闭当前的关卡。

点击工具栏上的 Save Current，或是从菜单栏中选择 File-Save All，

然后在弹出的界面中，在 Name 处输入 FirstMap，或是任何你喜欢的其它名称，点击 Save 即可。



关闭编辑器，就可以结束今天的学习了。

好了，今天的课程到此结束，先休息一会儿吧~

笨猫学编程QQ群：375143733
知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

新浪博客:
http://blog.sina.com.cn/eseedo

微博：
http://weibo.com/eseedo

让我们继续学习吧。

关于蓝图系统

现在我们的 已经有模有样了，但是如果把它放置在一个上旋转会更有意思。而要创建一个转台，我们需要用到蓝图。

用最简单的话来首，蓝图代表着一个"事物"。我们可以用蓝图来创建游戏对象的定制化行为。游戏对象可以是实物（比如转台），也可以是某种抽象的概念，比如健康系统。

想要设计一辆会飞的汽车吗？用蓝图吧。想要设计一个飞猪？用蓝图吧。想要一个 的 ？用蓝图吧。

和材质类似，蓝图使用的也是基于节点的系统。这就意味着使用蓝图系统开发要做的就是创建节点，然后在节点之间创建关联，不需要写一行代码~！

注意：如果你是个代码狂魔，那么当然可以选择使用 C++。
而且在虚幻 4 的 4.18 版本中，使用蓝图创建的游戏是可以转化为 C++代码的。
尽管蓝图用起来很贴心很方便，但是在执行效率上当然是不如 C++代码的。所以如果我们需要用到某种复杂的算法，那么就必须考虑使用 C++。

但即便你是个代码狂魔，那么也仍然可以考虑使用蓝图系统，大不了两个一起用吗。
使用蓝图系统有以下的好处：

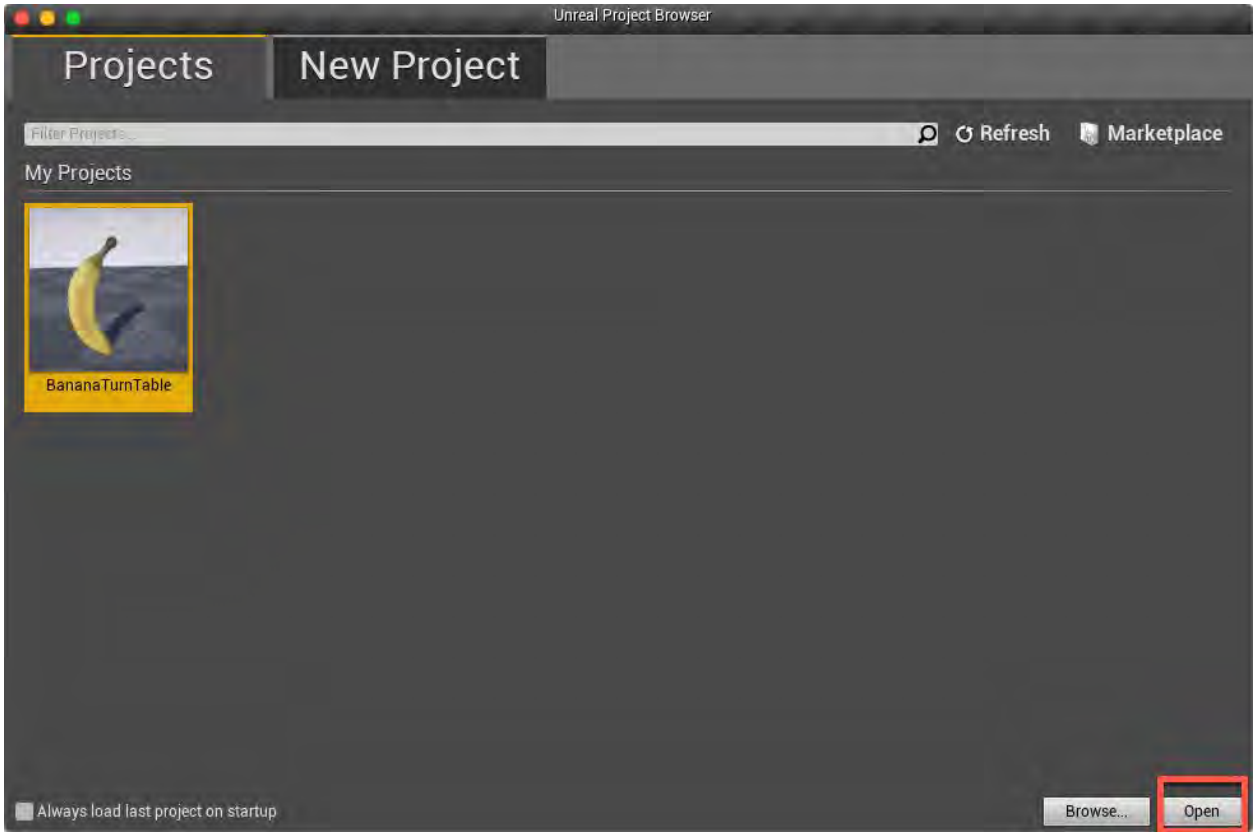
（1）通常来说，使用蓝图开发要比 C++快。
（2）蓝图更便于项目的组织。我们可以把节点很方便的分隔成不同的区域，比如功能和 graph。
（3）如果你是个 3D 美术设计人员，或者需要跟 3D 美术设计人员密切合作，那么使用蓝图系统会更便于修改项目。

通常来说，建议大家使用蓝图系统来创建对象。当我们需要用到比较复杂的算法，或者需要提升运行效能时，可以将其转换为 C++代码。

打开项目

在上一课的内容中我们已经保存了项目。因此，当需要重新打开项目时，首先还是打开 Epic Games Launcher，然后点击 Launch 按钮打开引擎。
等待加载完成后，会看到类似下面的界面：

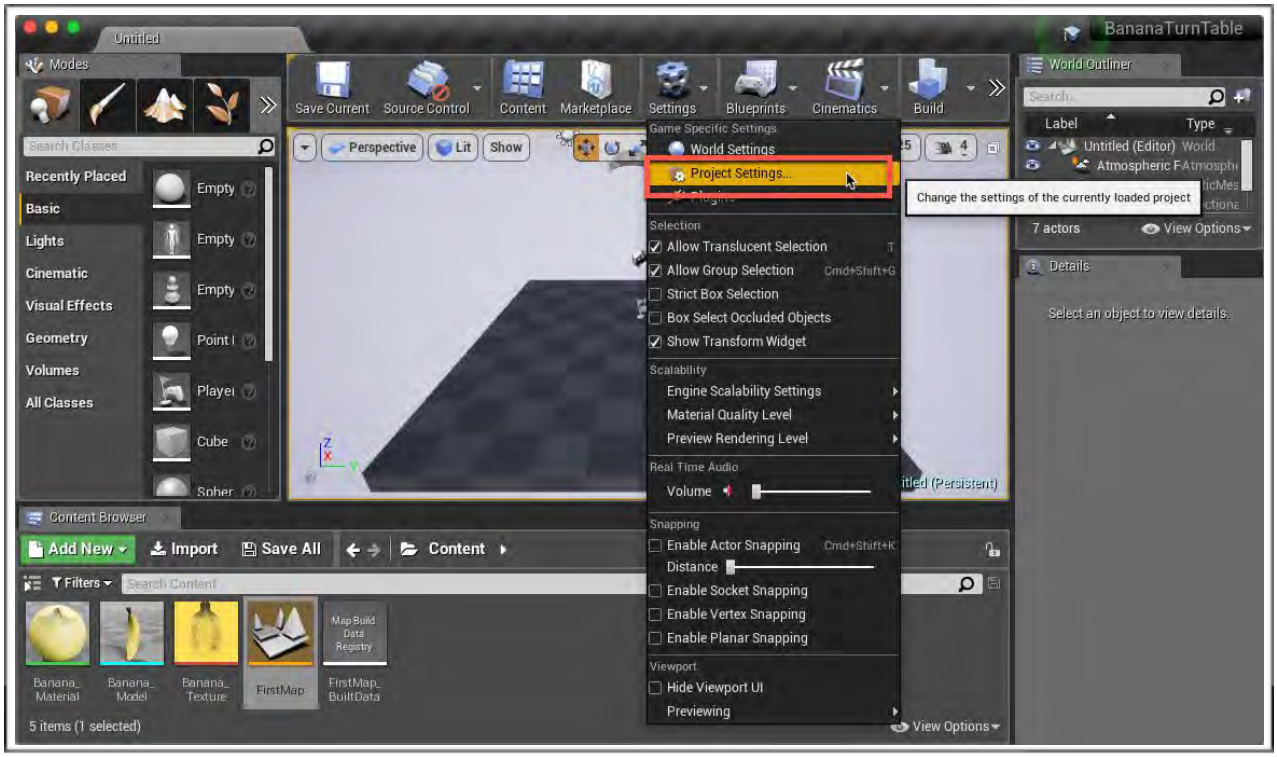
点击 Open 即可打开项目。此时你会看到我们之前的游戏场景不见了，只有在 Content Browser 中



双击打开 FirstMap 才能进入游戏场景。

为了方便起见，我们可以对项目做一点设置。点击工具栏上的 Setting，选择 Project Settings，

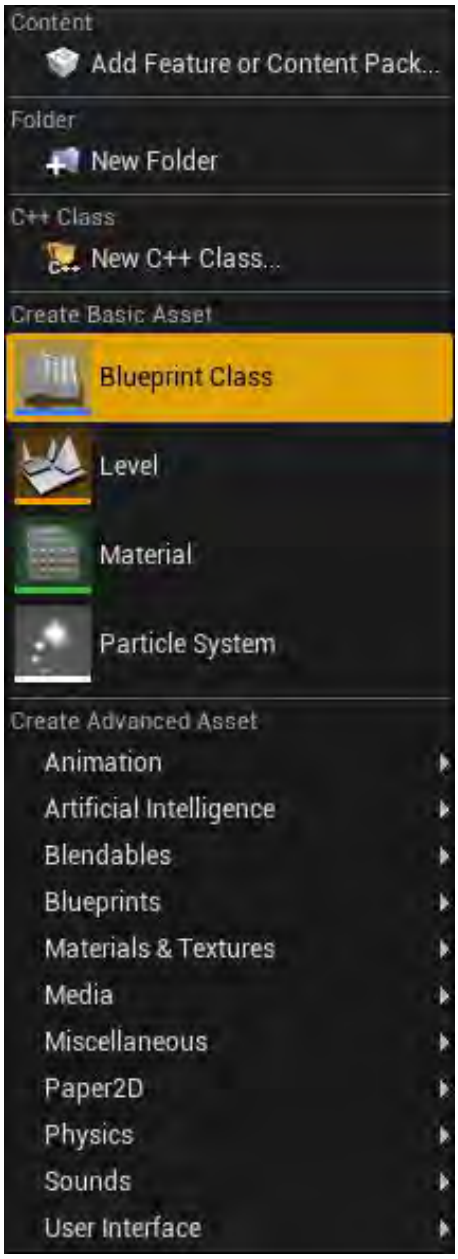
在打开的新界面中，从左侧选择 Maps&Modes，然后在 Default Map 处将 Editor Startup Map 和 Game Default Map 右边的属性选择为 FirstMap，也就是我们在上一课中所保存的游戏场景。关闭



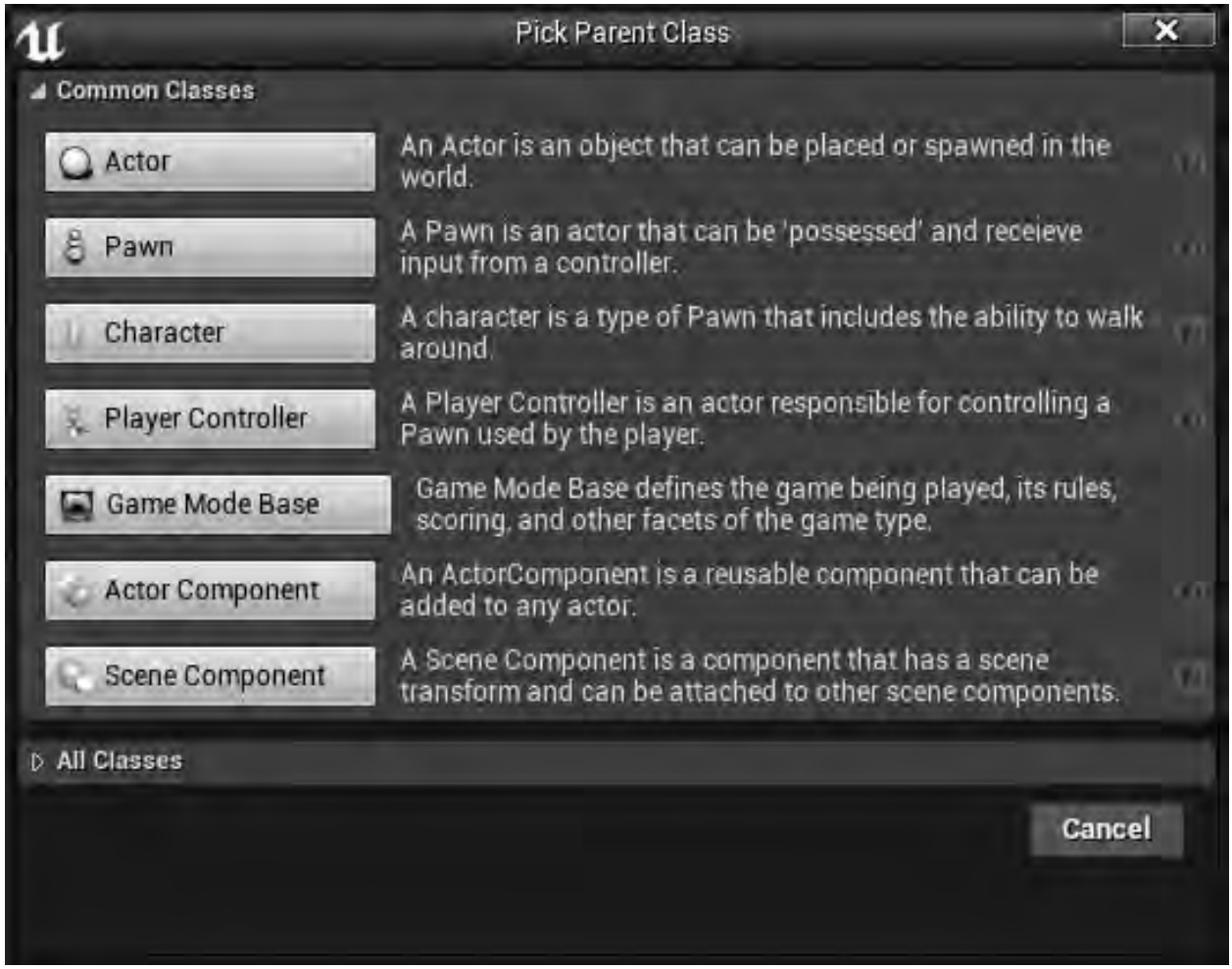
虚幻编辑器，从 Epic Games Launcher 中重新打开引擎，然后打开项目，就会自动打开之前所创建的游戏场景了。

创建蓝图

在虚幻 4 引擎中切换到 Content Browser，然后点击 Add New，从列表中选择 Blueprint Class。



此时会弹出一个窗口，提示让我们选择蓝图的父类。关于父类和继承的概念，我们会在后面详细介绍，这里只需要知道，当我们选择某一种父类后，蓝图系统将继承该父类的所有变量、函数和组件。在继续之前，建议大家仔细阅读以下每个父类的描述。



注意，对于 Actor,Pawn 和 Character 来说，Character 继承自 Pawn,而 Pawn 又继承自 Actor。此外，Player Controller 也继承自 Actor。

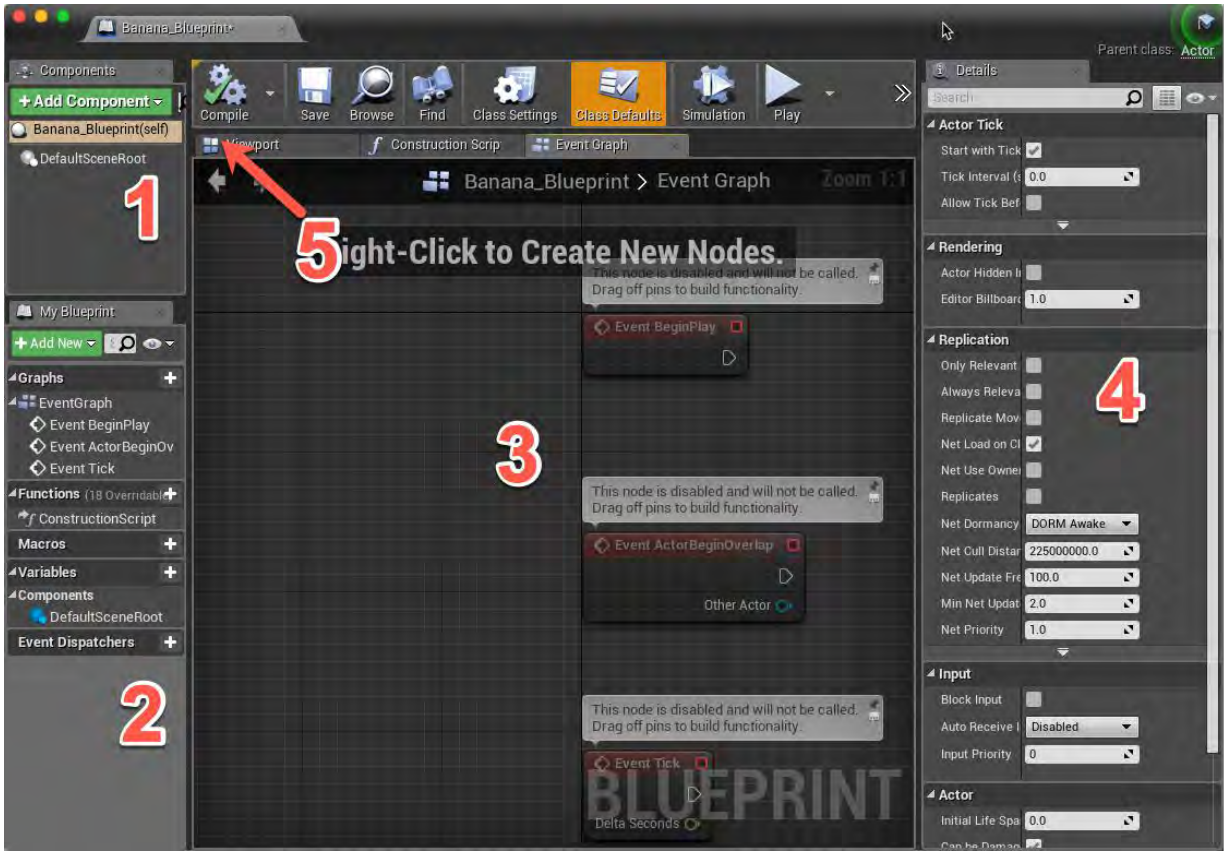
简单来说，Pawn，Character 和 Player Controller 也是 Actor。

考虑到我们这里要添加的转台会在游戏场景中的固定位置，Actor 类显然最合适。因此我们在这里选择 Actor，并将新的蓝图类命名为 Banana_Blueprint。

最后，双击 Banaba_Blueprint 打开这个蓝图文件。

如果你看到的是类似下面的界面，那么请点击 Open Full Blueprint Editor，

完全状态下的蓝图编辑器如下图所示



其中各个区域的作用如下：
1.Components 区：
包含了当前组件的列表。
虚幻 4 中的组件和 Unity3d 中的组件有一定的类似之处，但又有一定的区别。

2.My Blueprint 区：
这个部分用于惯例蓝图中的 graph，函数和变量。我们可以按住鼠标右键不放，拖动鼠标以平移，也可以通过鼠标滚轮来缩放视图。

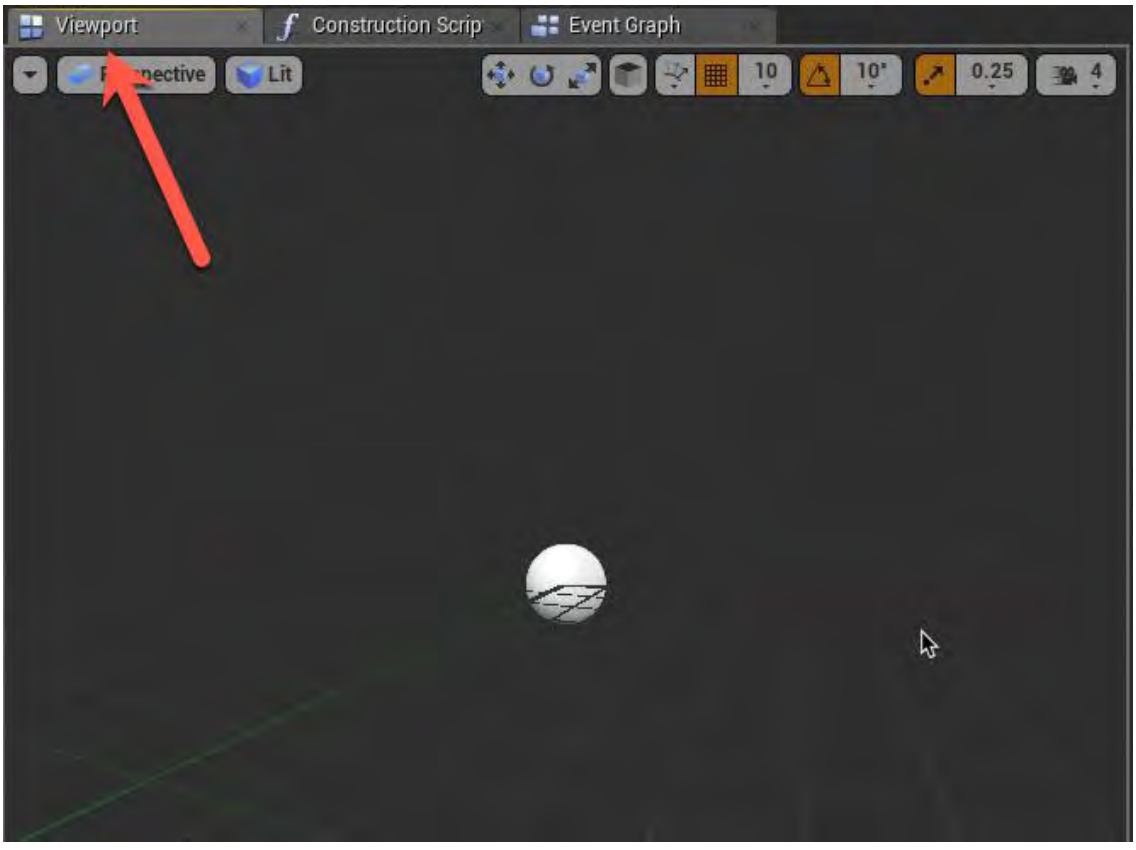
3.Graph 区：
该部分是蓝图系统的精华所在。这里会显示蓝图中所有的节点和逻辑机制。
当然，要注意此时我们选择的是 Event Graph

4.Details 区：

该部分用于显示所选择项目的具体属性

5.Viewport 区：

当我们在选项卡上选择的是 Viewport，那么就可以看到带有视觉元素的所有组件。我们可以使用和主编辑器类似的操作在移动和查看视觉元素。
创建转台



想要创建转台,我们需要两个部分：底座和 display。我们可以使用组件来创建这两个部分。

什么是组件？

如果把蓝图系统比喻成一辆车，那么组件就是组成车的各种零部件，比如车门、车轮和引擎都是组件的例子。

不过组件并不局限于实际的物体。

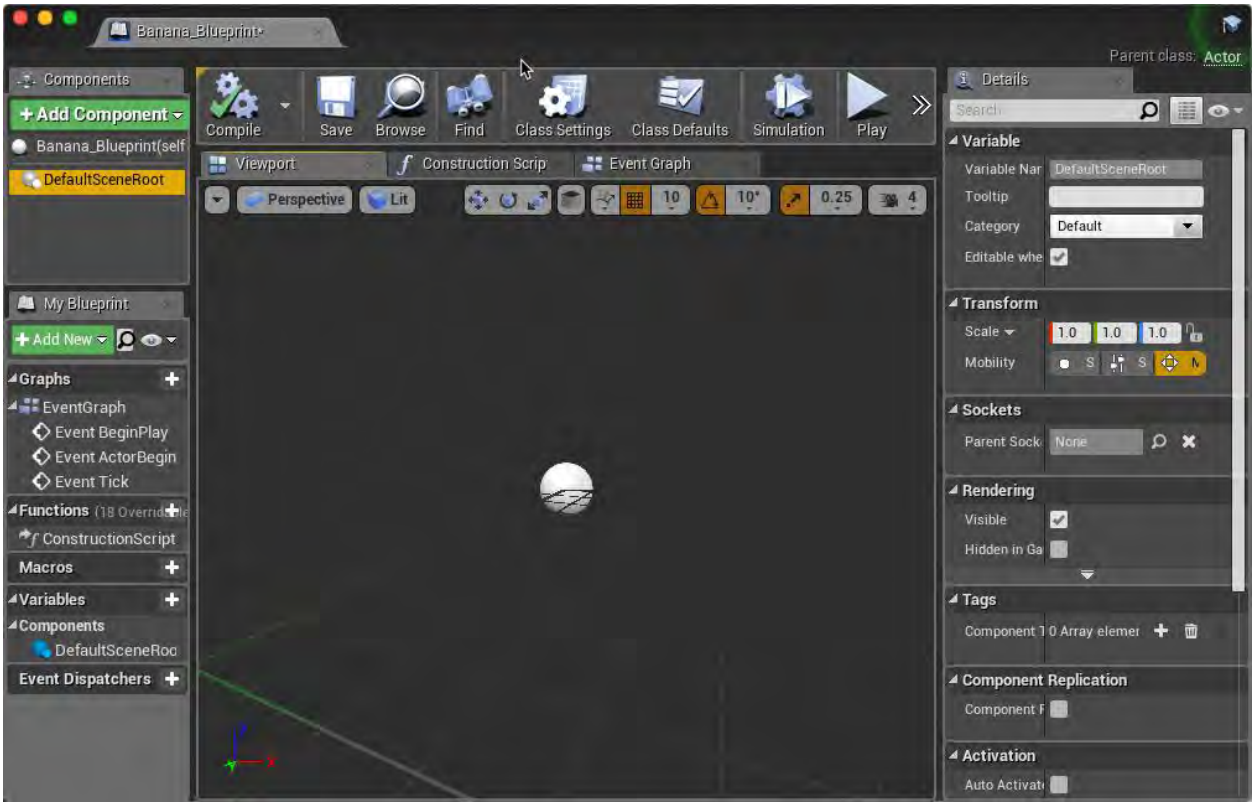
举例而言，为了让汽车移动，我们需要添加一个移动组件。为了让汽车飞行，我们又需要添加一个飞行组件。

学过 Unity3d 的朋友可能会对其中的组件系统有一定的了解。虽然和虚幻 4 中的组件系统有一定的差异，但大致的作用是类似的。

添加组件

如果我们需要查看组件的可视化元素，那么就需要切换到 Viewport 视图。点击 Viewport 选项卡以切换到该视图。

注意：DefaultSceneRoot 组件在游戏运行的时候并不会显示，它只会显示在编辑器中。

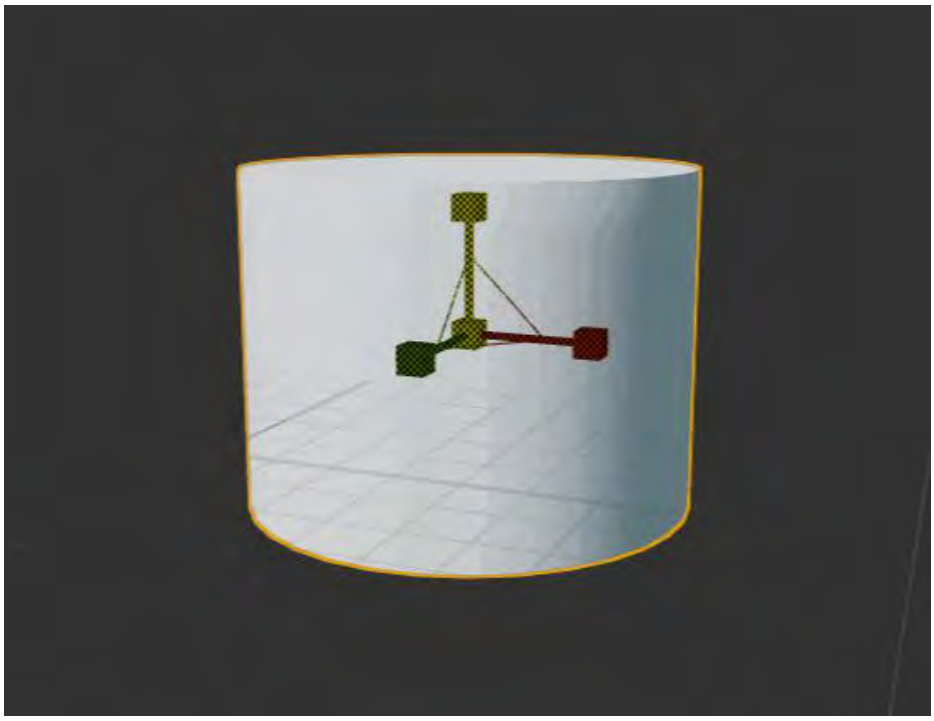


转台包含了两个组件：

（1）Cylinder：
一个简单的白色圆柱体，它将构成底座。
（2）Static Mesh:
该组件将显示香蕉的模型

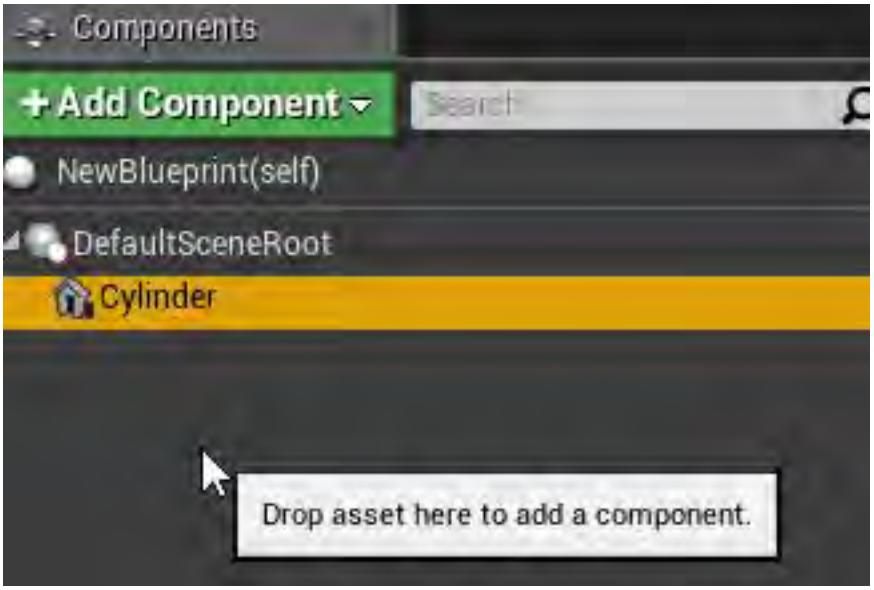
为了添加底座，让我们在蓝图编辑器中找到 Components 面板，点击 Add Component，然后选择 Cylinder。
此时在 Viewport 视图中可以看到多了一个圆柱体。
当然现在它的高度有点超出预期，我们需要把它变得扁平一点。按下键盘上的 R 键，然后把圆柱体的高度降低，具体的尺寸大小不是那么的重要，因为我们后续随时都可以继续调整。

现在该是添加模型的时候了。



返回 Components 面板，左键单击空白区，以取消选中 Cylinder 组件。这样确保我们所添加的下一个组件不会关联到 Cylinder 组件上。
注意：

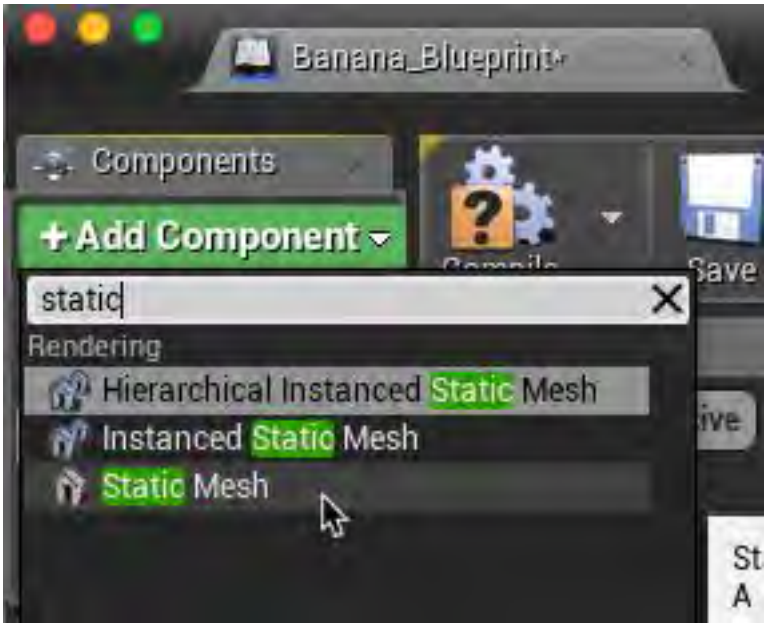
如果你不这么做，那么所添加的下一个组件就会关联到 Cylinder 组件上。这就意味着所添加的新组件将继承 Cylinder 组件的比例。因为我们曾经调整过圆柱体的比例，那么新添加的组件比例也

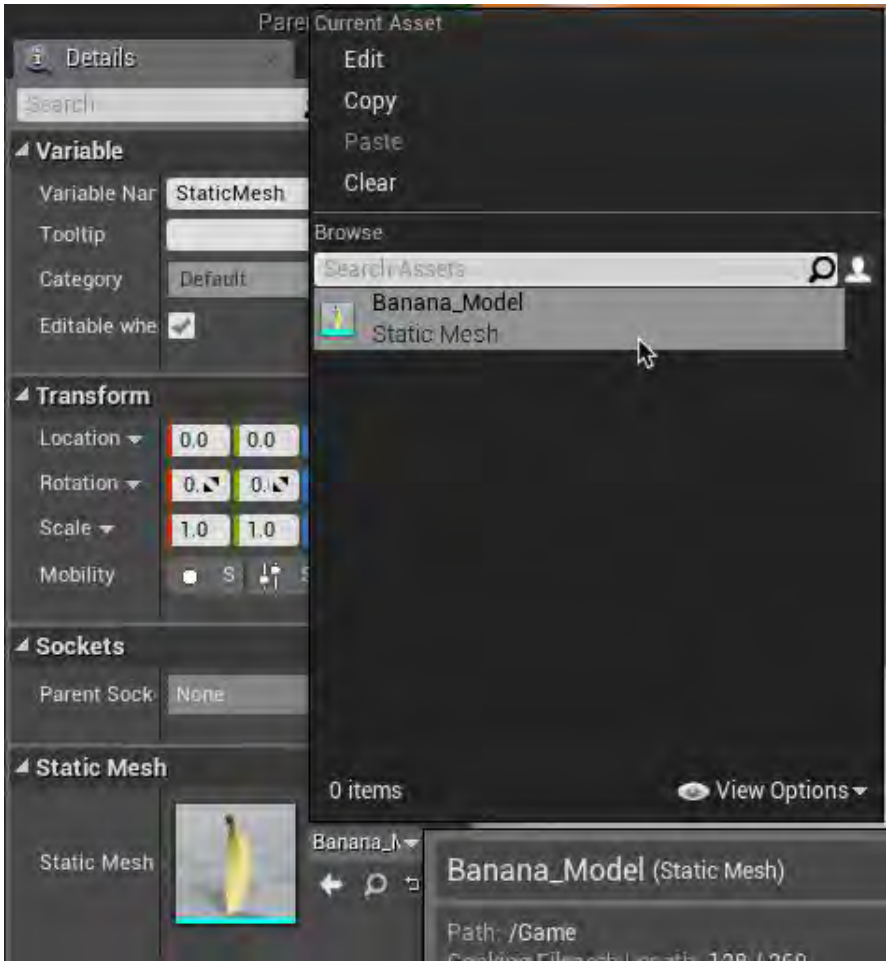


会被调整。

接下来，点击 Add Component，从列表中选择 Static Mesh。

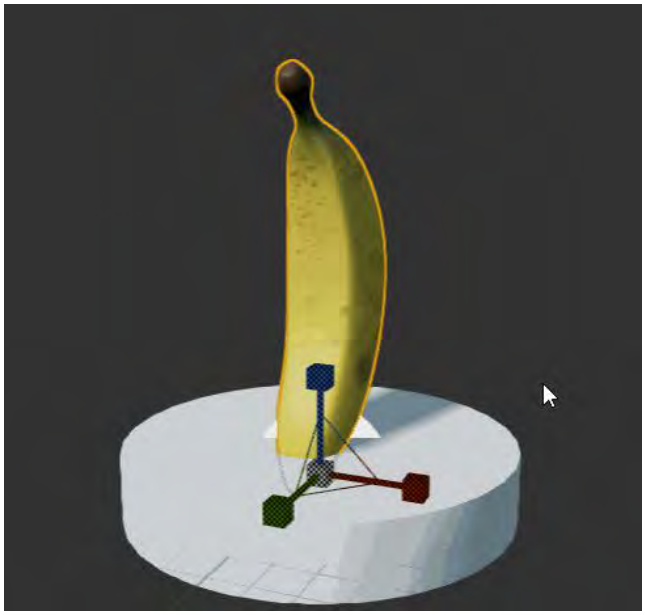
为了显示香蕉，需要选中 Static Mesh 组件，然后在 Details 面板中点击 Static Mesh 右侧的下拉列表，然后选择 Banana_Model。





接下来需要把香蕉移动到合适的位置。按下键盘上的 W 键，然后把香蕉向上移动到合适的位置。

点击蓝图编辑器工具栏上的 Compile 按钮和 Save 按钮，以保存所做的修改。



好了，现在我们已经在蓝图中添加了所需的视觉元素。

在下一课的内容中，我们将学习如何让转台旋转。

笨猫学编程QQ群：375143733
知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

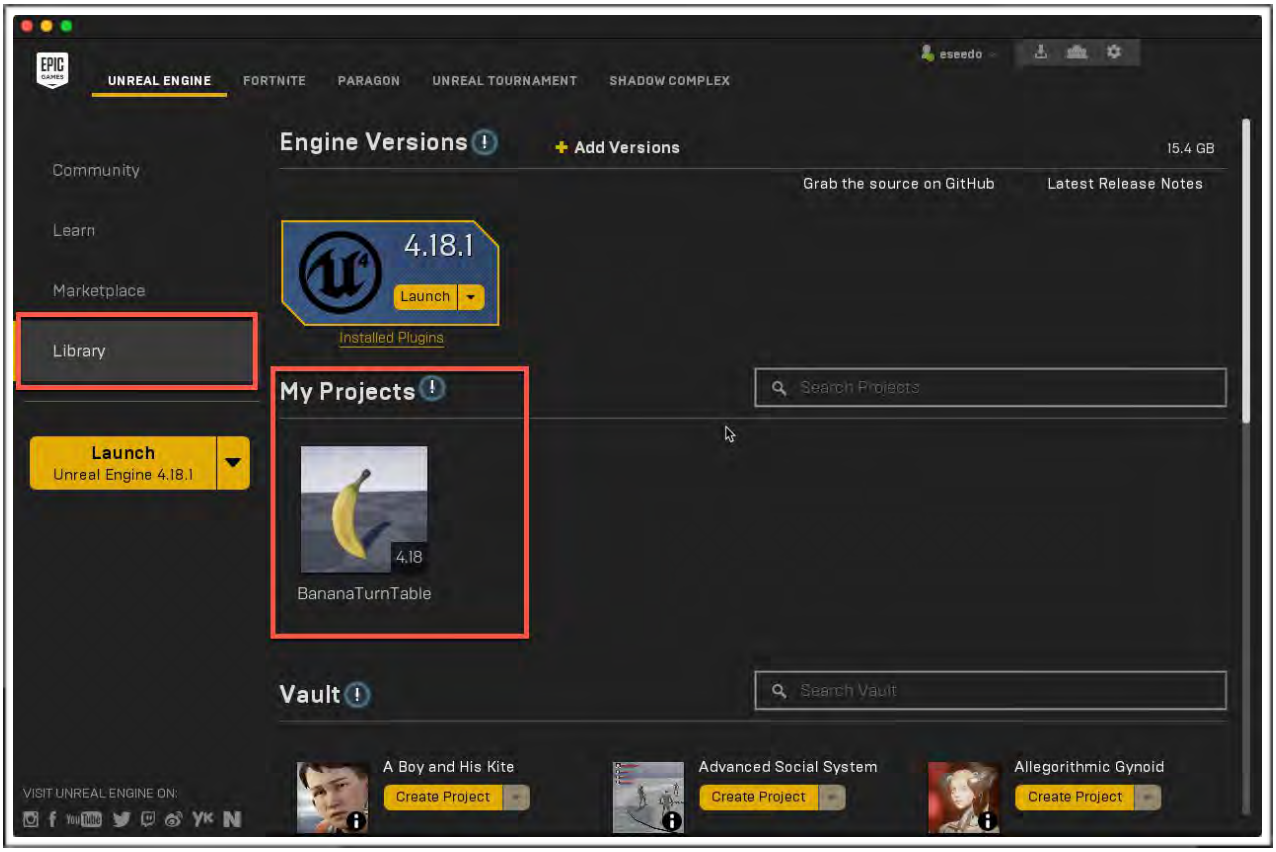
新浪博客:
http://blog.sina.com.cn/eseedo

微博：
http://weibo.com/eseedo

欢迎回到我们的课程，在这一课的内容中，我们将学习如何让 turntable 旋转。

为了实现这一点，我们需要了解蓝图节点的作用。

首先打开 Epic Games Launcher，在之前我们是先打开引擎，然后再打开项目的。其实也可以直接在 Epic Games Launcher 中打开项目。
在 Launcher 左侧的选项卡中切换到 Library，然后在 My Projects 中可以看到



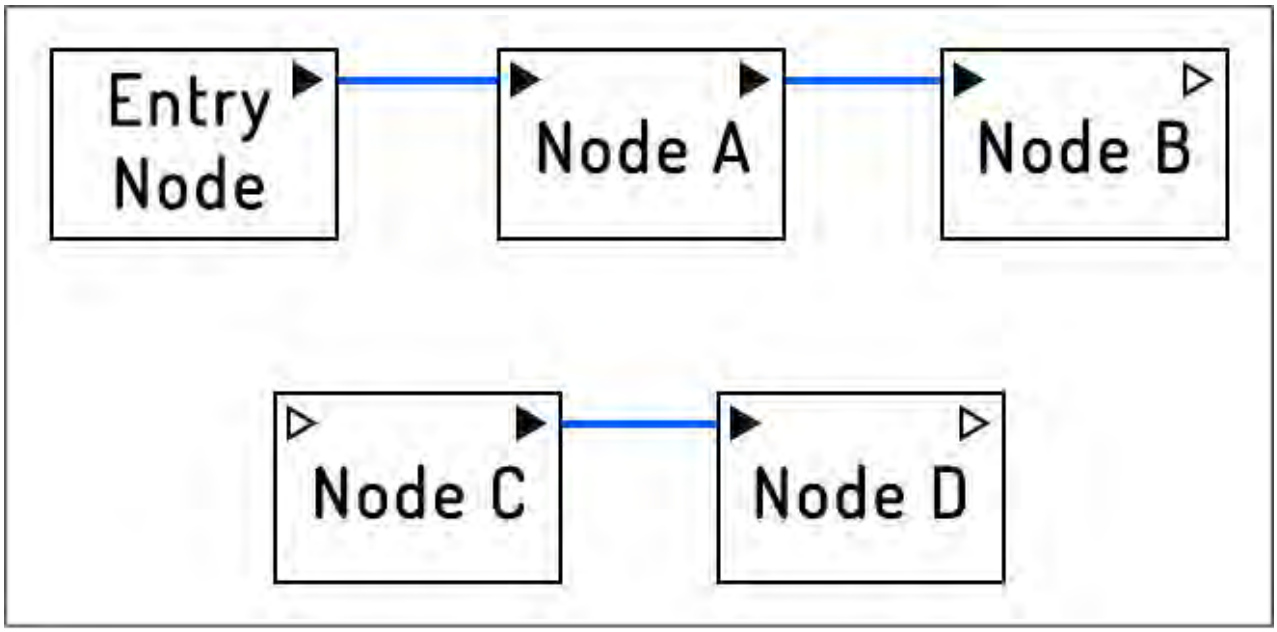
BananaTurnTable 项目，双击将其打开即可。

和之前学过的材质编辑器中的节点类似，蓝图节点也有特殊的接口，名为 Execution pins，或者说执行接口。

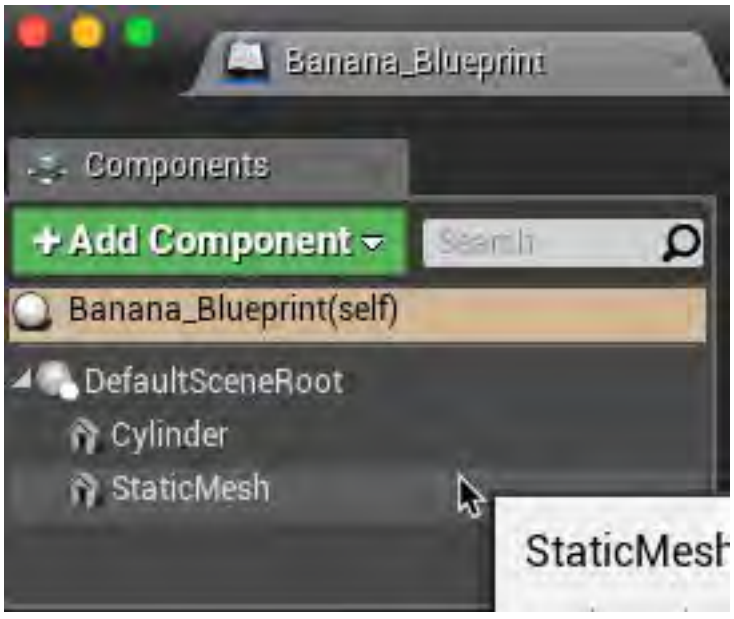
和材质节点类似，左侧的接口为输入接口，而右侧接口为输出接口。所有的节点都至少有一个接口。

如果某个节点有一个输入接口，那么在其被执行前必须有一个连接到该节点上。如果某个节点没有被连接，那么所有其后的节点都不会被执行。

下面是一个简单的示例：

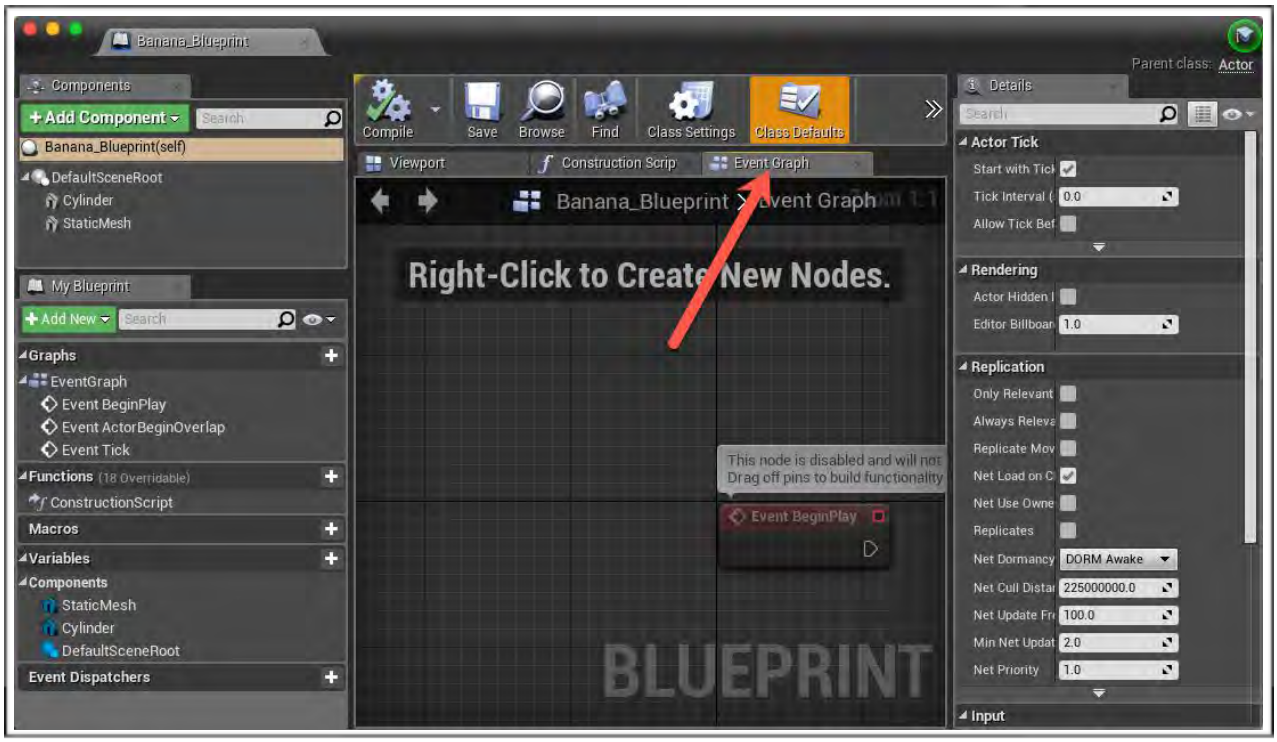


在上图中，节点 A 和 B 是可以执行的，因为它们的输入接口之间有一个连接。但是节点 C 和 D 无法执行，因为节点 C 有一个输入接口，但是没有连接。



旋转 Turntable

在我们开始之前，先打开 Banbaba_Blueprint 这个蓝图，然后查看一下 Components 面板。可以看到这里 Cylinder 和 StaticMesh 是缩进显示的，但 DefaultSceneRoot 却没有。这是因为 Cylinder 和 StaticMesh 是附属于 DefaultSceneRoot 上的。
当我们移动、旋转或缩放某个根组件时，附属于它的所有组件也会做同样的操作。通过这样，我们就可以同时旋转 Cylinder 和 Static Mesh 两个游戏对象了。

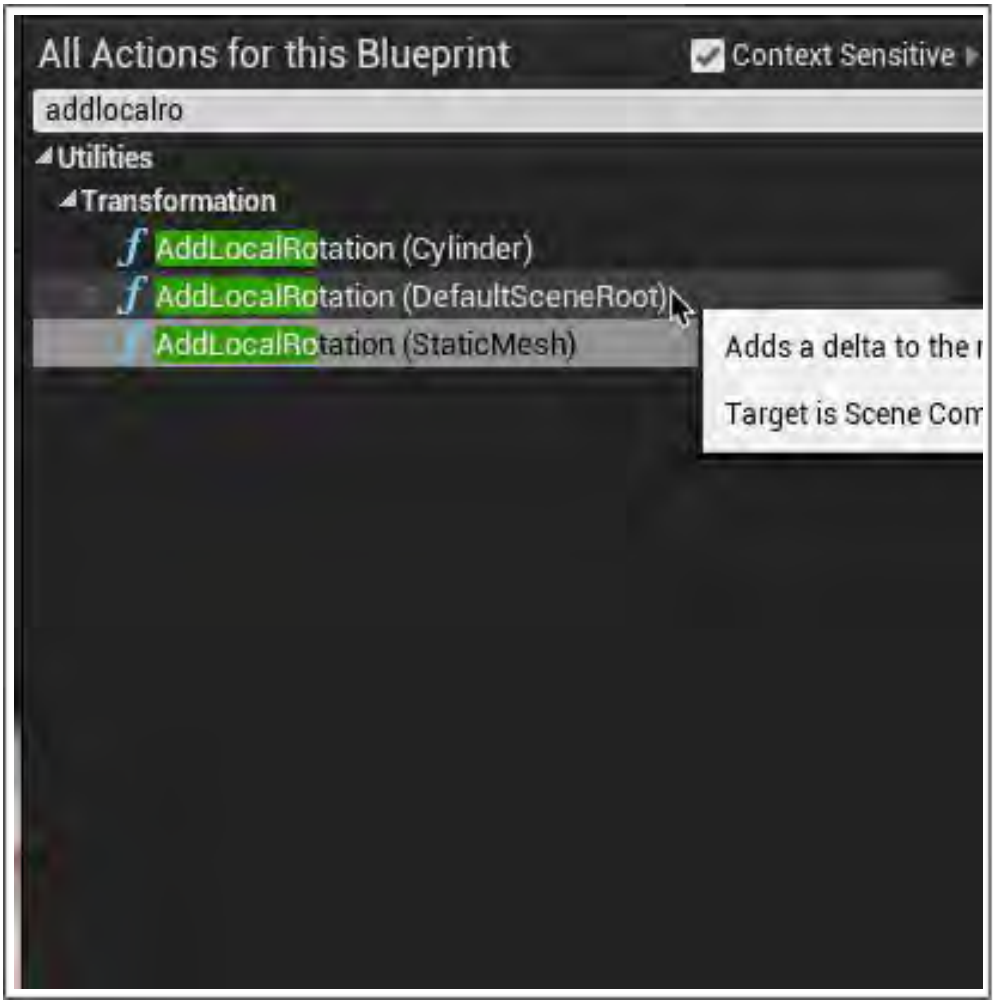


创建节点

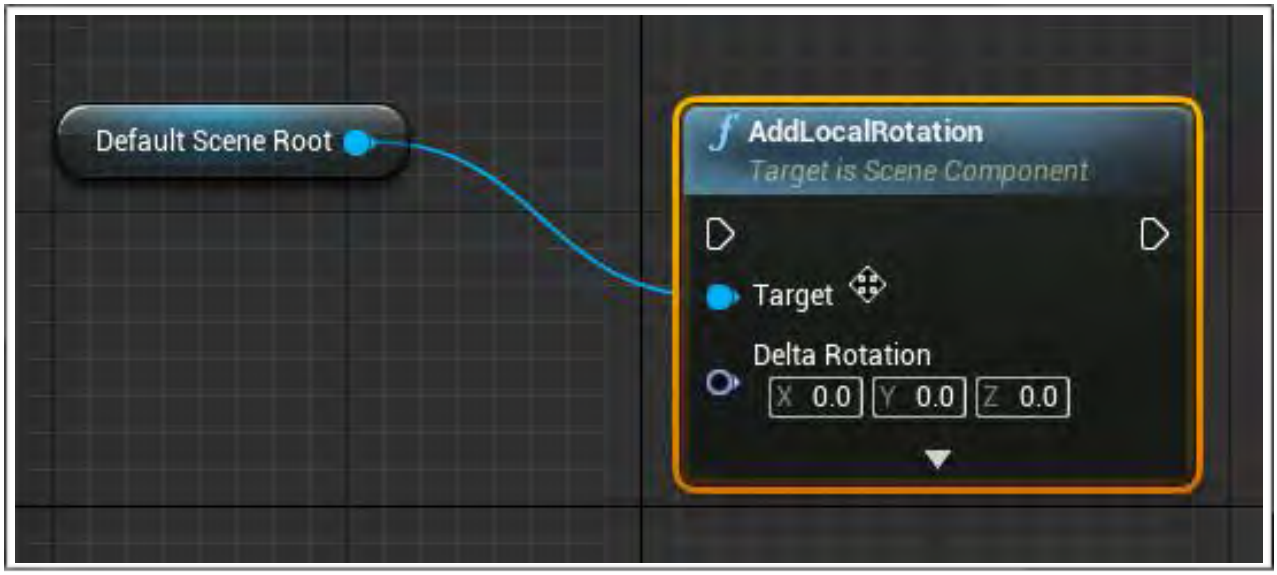
为了编写蓝图的逻辑，让我们切换到 Event Graph 选项。

让某个游戏对象非常简单，我们只需创建一个节点就可以了。
在 Event Graph 的空白区域右键单击，然后可以看到一系列可用的节点列表，在里面搜索 AddLocalRotation。因为我们要同时旋转底座和香蕉，所以只需要旋转根组件即可。这里我们选择 AddLocalRotation(DefaultSceneRoot）。

注意，如果在搜索的时候没有发现自己需要的节点，那么请取消对 *Context Sensitive*（大小写敏感）的勾选。



此时在节点中可以看到类似下面的画面：

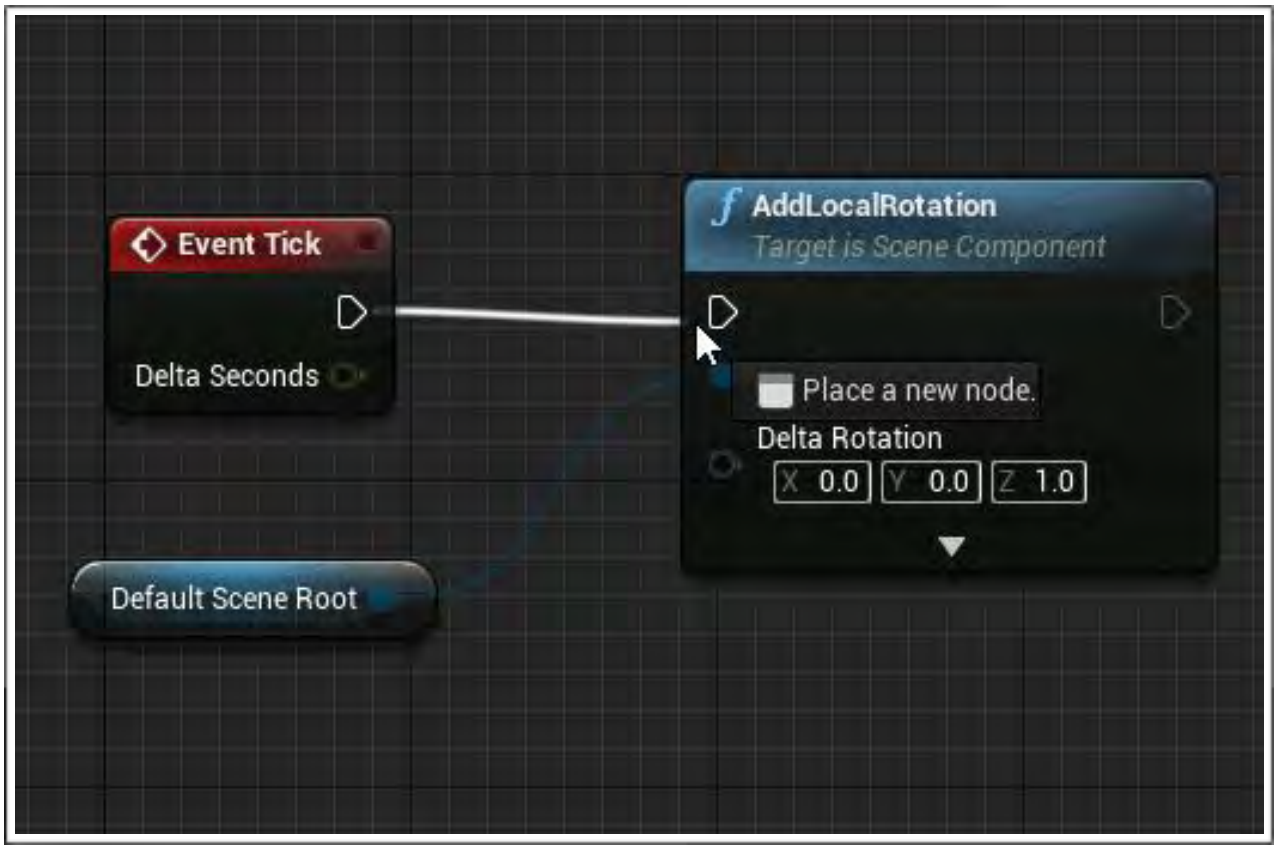


可以看到，在 Target 输入接口有一个连线，自动连接到所选的组件。

为了设置旋转角度，可以在 Delta Rotation 中将 Z 值更改为 1.0，这样蓝图就会围绕 Z 轴旋转。数值越大，turntable 旋转的速度也就越快。

为了让 turntable 持续旋转，我们需要让 AddLocalRotation 每一帧都执行。为此，需要用到 Event Tick 节点。

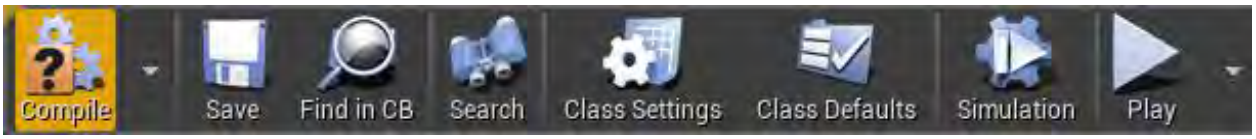
默认情况下在 Event Graph 中已经存在该节点了，如果没有，我们可以用刚才的方法来添加一个。



从 Event Tick 节点的输出接口连一条线到 AddLocalRotation 节点的输入接口，如下图。

需要注意的是，turntable 的旋转速度跟帧速有关。换句话说，如果你的电脑配置比较低，运行速度比较慢，那么 turntable 的旋转速度也会比较慢。对本教程来说，这样就可以了。在后续的教程中，我们将学习如何解决这个小问题。

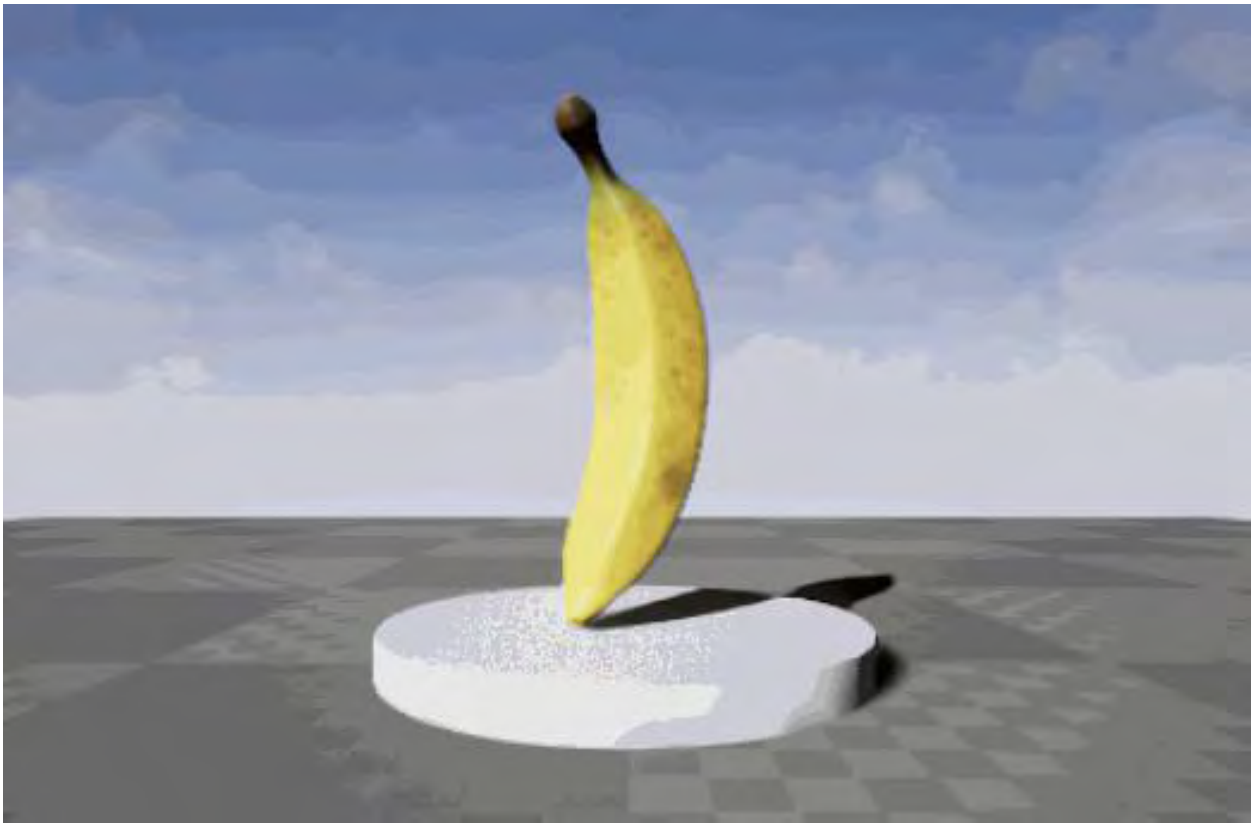
最后，在工具栏上点击 Compile 来更新蓝图，然后关闭蓝图编辑器。



在场景中添加蓝图

在向场景中添加蓝图之前，首先返回虚幻 4 主编辑器，然后从 Viewport 中删除之前所添加的香蕉模型，
选中该物体（最保险的方法是在 World Outliner 中选择），然后使用 Edit-Delete 将其删除，或者直接使用 Delete 键来删除。

接下来的事情就很简单了，在 Content Browser 中找到 Banana_Blueprint 这个蓝图文件，



然后使用鼠标左键选中，把它拖到场景中就可以了。

从编辑器的工具栏中点击 Play 按钮，就可以预览游戏的效果了~

注意：如果你没有删除之前的　模型，那么就会收到一个警告，大概意思是光照需要 rebuild。当我们删除此前的　模型后，一切就会正常了。

好了，本系列教程的第一个小环节就到此结束了。

在接下来的教程中，我们将进一步了解虚幻 4 的蓝图系统。

在上一部分的课程中，我们学习了如何创建一个最简单的虚幻 4 项目，并且成功的让香蕉在转台上自由旋转。

在这一部分的内容中，我们将重点介绍虚幻 4 的蓝图系统。

蓝图系统是虚幻 4 引入的一个非常重要的可视化编程系统，它可以让我们在极短的时间里创建游戏圆形。有了蓝图系统，我们无需一行行手写代码，可以完全使用可视化的方式来创建游戏逻辑：拖曳和放置节点，在用户界面中设置节点的属性，并创建连线在节点之间创建关联。

除了是一个快速游戏原型开发工具之外，蓝图系统还可以让不懂编程的人很快开始自己的编程之旅。

在本部分的内容中，我们将使用蓝图系统实现以内效果：

1.设置一个自顶向下的摄像机

2.创建一个可以实现基本运动的玩家控制的游戏角色

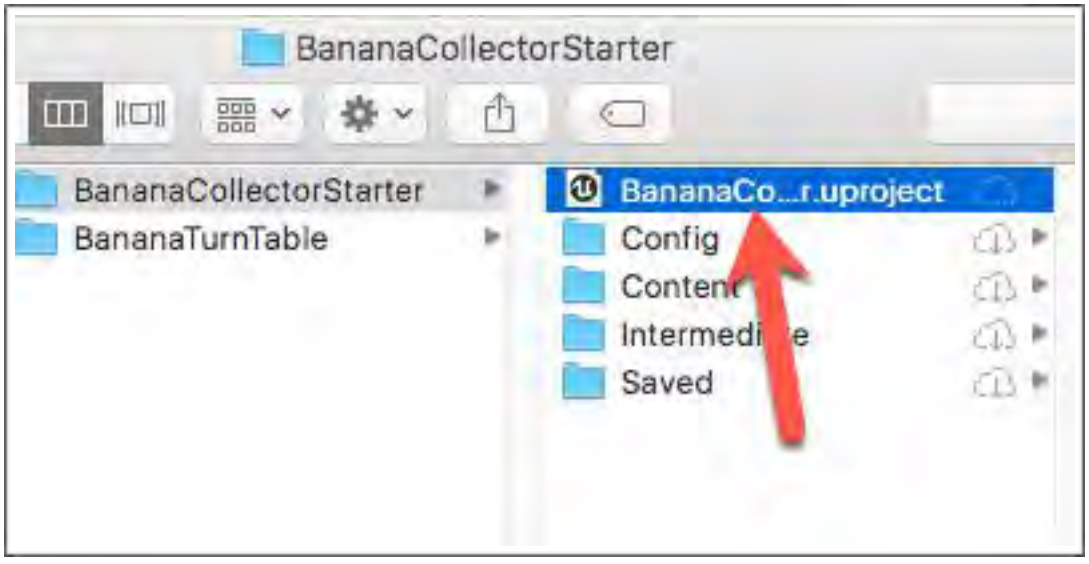
3.设置玩家的输入

4.创建一个物品，当玩家碰到它的时候就会消失。

注意：本部分教程假定你已经完成了之前的内容学习，或者具备虚幻 4 的基本开发技能。如果你对虚幻 4 一无所知，建议先看看 01-04 的内容。

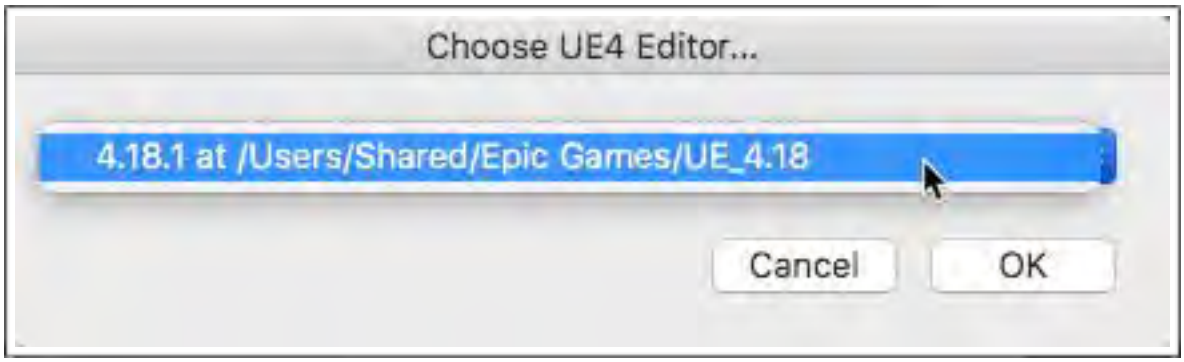
此外，这篇教程还涉及到对向量的基本使用。如果你对向量一无所知，建议可以阅读 gamedev.net 上的此文（https://www.gamedev.net/resources/_/technical/math-and-physics/practical-use-of-vector-math-in-games-r2968 ）

开始前的准备



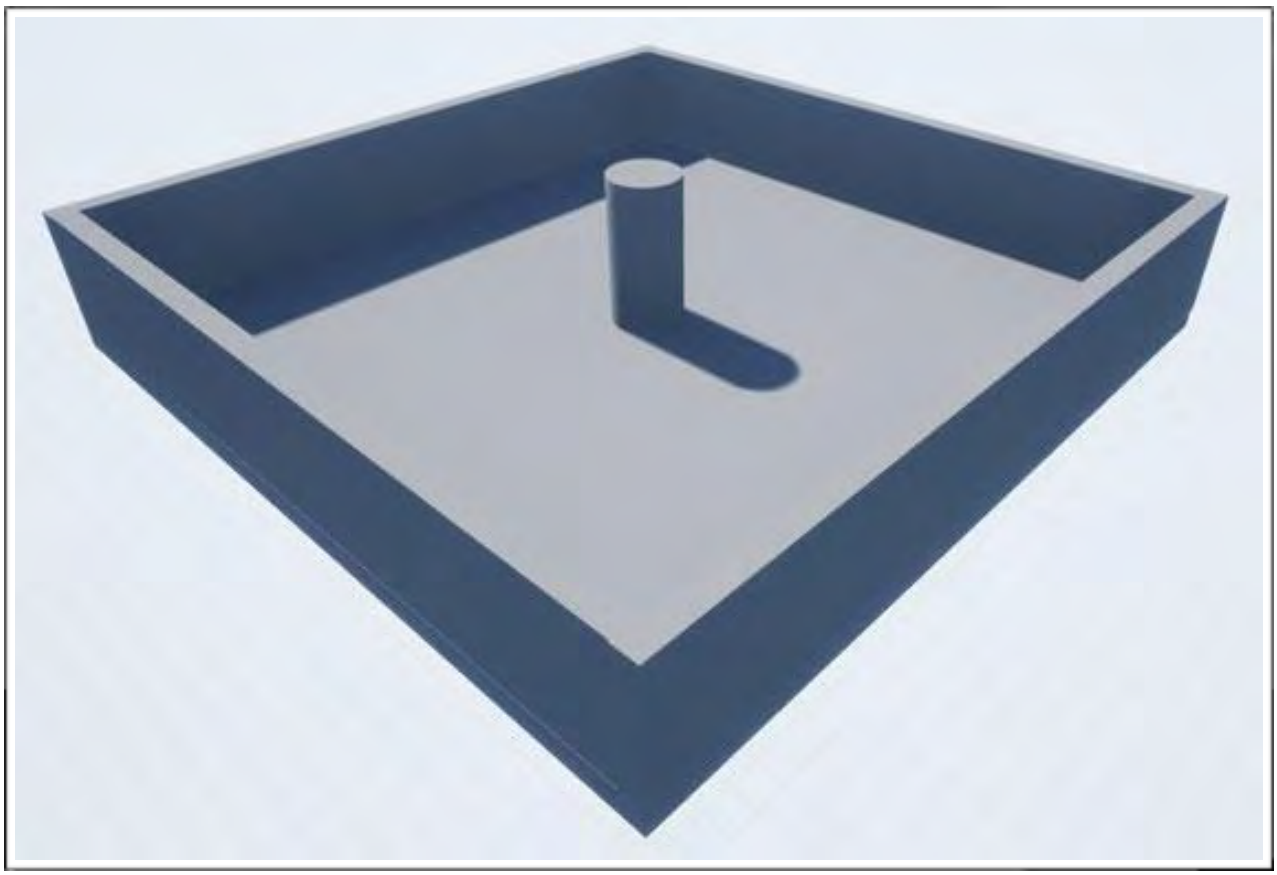
在开始之前，先从这里下载起始项目（链接:https://pan.baidu.com/s/1pL3KpKn 密码:yvxu），然后将其解压缩。找到项目文件夹，然后双击 BananaCollector.uproject 以打开项目。
此时可能会提示你选择编辑器，选择自己所安装的版本即可。

此外，如果你看到提示，告诉你这个项目使用了之前版本的虚幻编辑器，也没有关系，因为虚幻



4 引擎的更新还是很频繁的。你可以选择打开一个拷贝，或是转换该项目。

打开后，可以在 Viewport 视口中看到类似下面的场景，玩家将在其中移动，并收集物品。



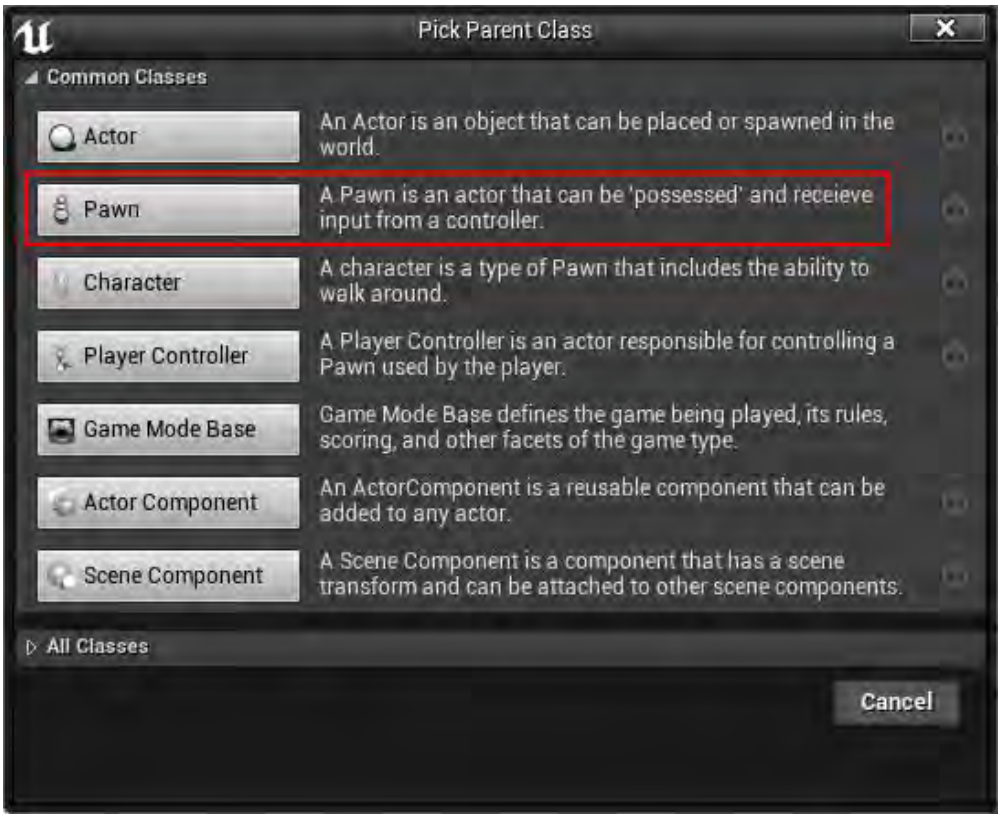
可以看到，我们这里已经把项目文件做了分类，如下图所示：

可以点击红框的按钮来显示或隐藏 sources 面板。

创建玩家

在 Content Browser 面板中，找到 Blueprints 文件夹，点击 Add New 按钮，然后选择 Blueprint Class。

因为我们希望游戏角色可以从玩家那里接收输入指令，因此 Pawn 类比较合适。从弹出的窗口中



选择 Pawn，然后将其命名为 BP_Player。
当然，这里要说明一下的是，如果你选择 Character 类也是可以的，这个类里面甚至包含了让角色移动的组件。但是考虑到这里我们需要学习如何实现自己的运动系统，所以选择 Pawn 类就足够了。

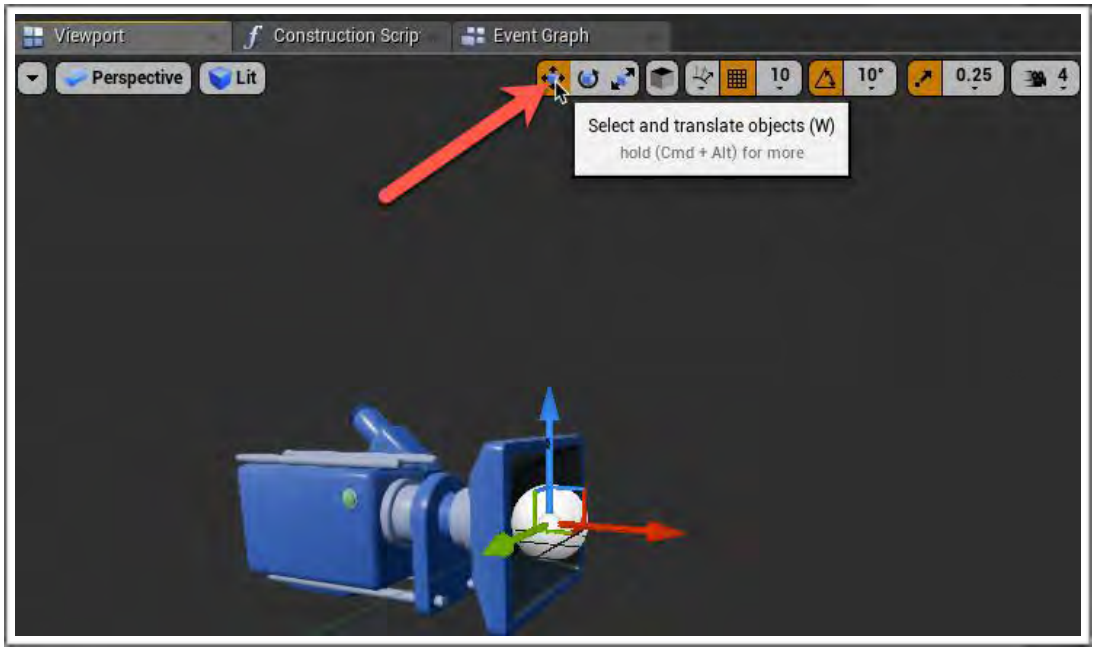
关联一个摄像机

摄像机是玩家观察游戏世界的方式。我们希望可以创建一个摄像机，让它向下朝着玩家。

在 Content Browser 中，双击 BP_Player，在蓝图编辑器中打开这个蓝图文件。

为了创建摄像机，让我们在蓝图编辑器的 Components 面板中点击 Add Component，然后选择 Camera。

为了让摄像机以自顶向下的视角显示，我们需要把它放置在玩家的上面。在 Components 面板

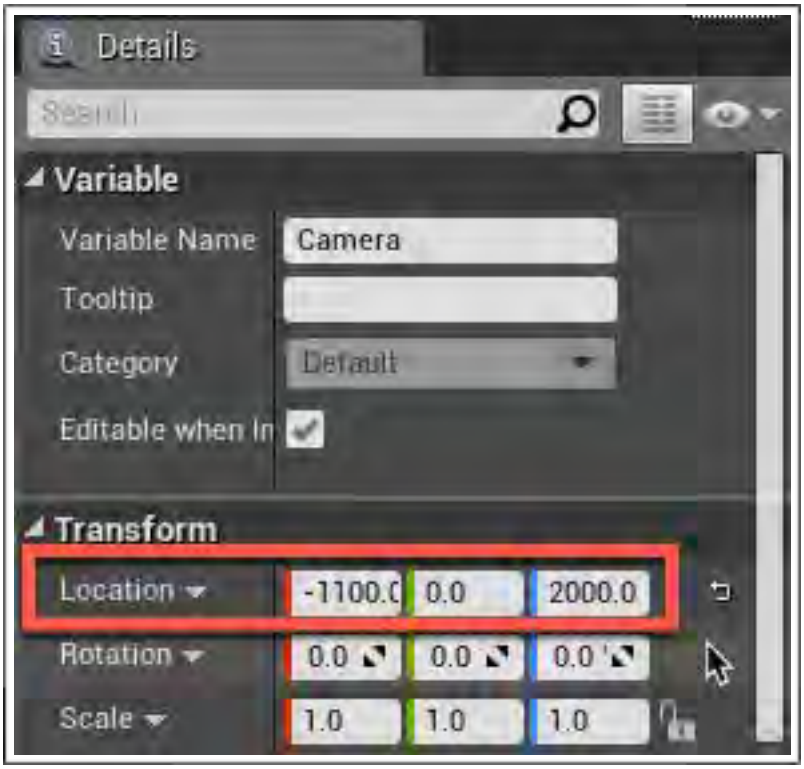


中保持选中摄像机组件，然后切换到 Viewport 视图。

使用键盘上的 W 键，或者在工具栏上点击选择移动工具，然后把摄像机的位置拖动到（-1100,0,200）。

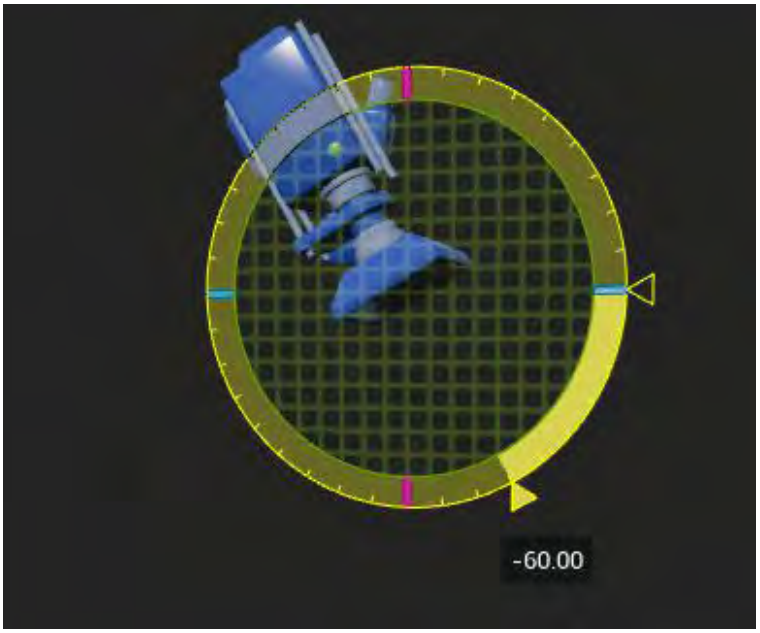
当然，你也可以直接在 Details 面板的 Location 字段中输入以上数值，如下图所示。

如果你在 *Viewport* 里面看不到摄像机，那么按下 *F* 键可以重新聚焦到它上面。



接下来按下键盘上的 *E* 键，将摄像机沿着 *Y* 轴向下旋转 *-60* 度。当然，同样的我们可以直接在 *Details* 面板的 *Rotation* 字段中输入以下数值。

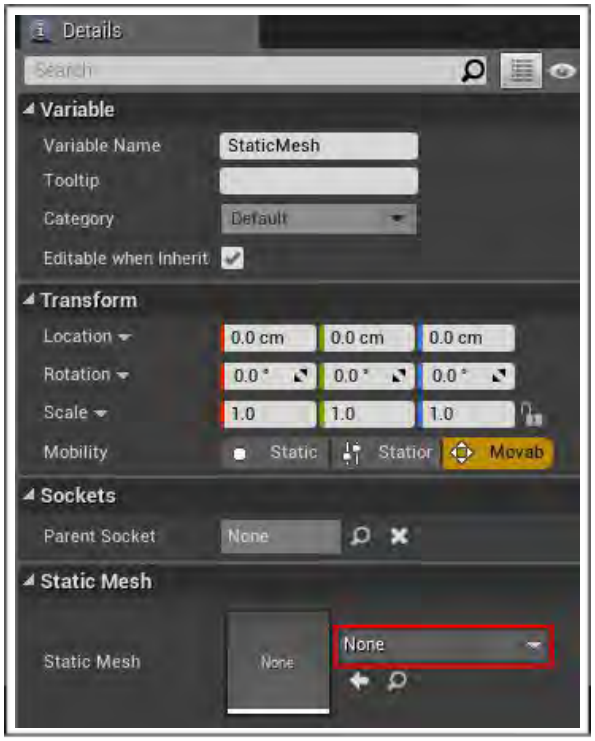
玩家登场



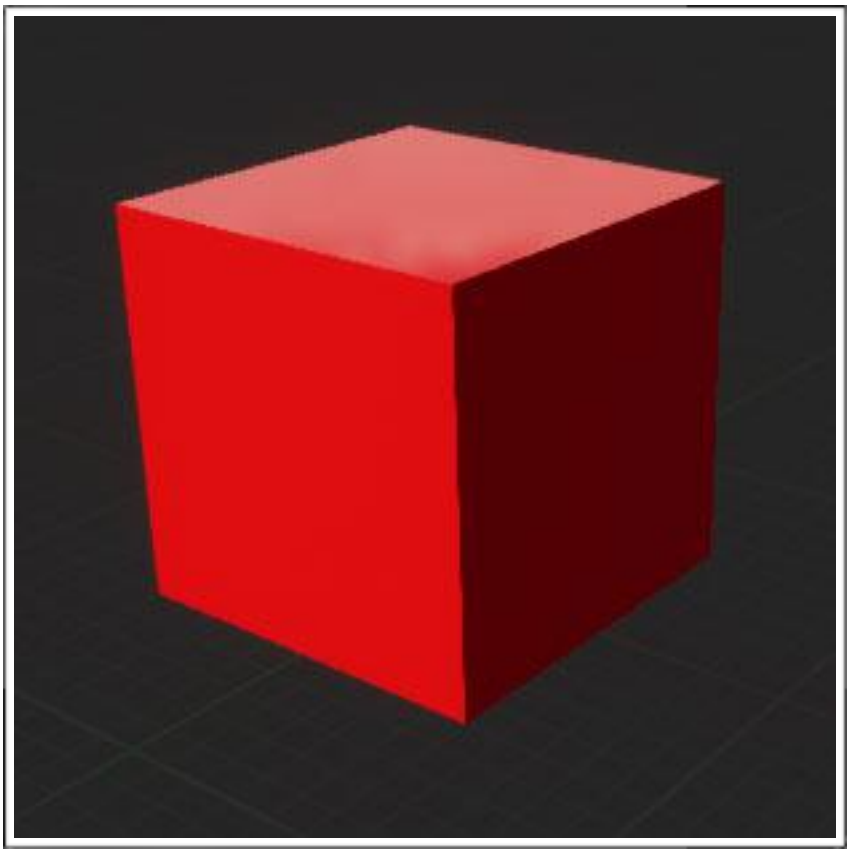
在这个游戏中，我们将使用红色的方块来代表玩家，这样我们就可以使用 *Static Mesh* 组件来显示它。

首先，在 *Components* 面板的空白处左键单击，以取消选择 *Camera* 组件。如果不这样做，那么下一个所添加的组件将成为 *Camera* 组件的子对象。

点击 *Add Component*，然后选中 *Static Mesh*。



为了显示红色方块，保持选择 Static Mesh 组件，然后切换到 Details 面板。在 Static Mesh



右侧的下拉列表中选择 SM_Cube。
在添加完成后，在 Viewport 视口中点击 F 键，可以看到类似的物体。

接下来该是让玩家登场的时候了，点击工具栏上的 Compile 按钮，然后返回主编辑器。

让玩家出现

在让玩家控制 Pawn 对象前，我们需要确定两个信息：
1.玩家需要控制的 Pawn 类
2.Pawn 对象将出现的位置

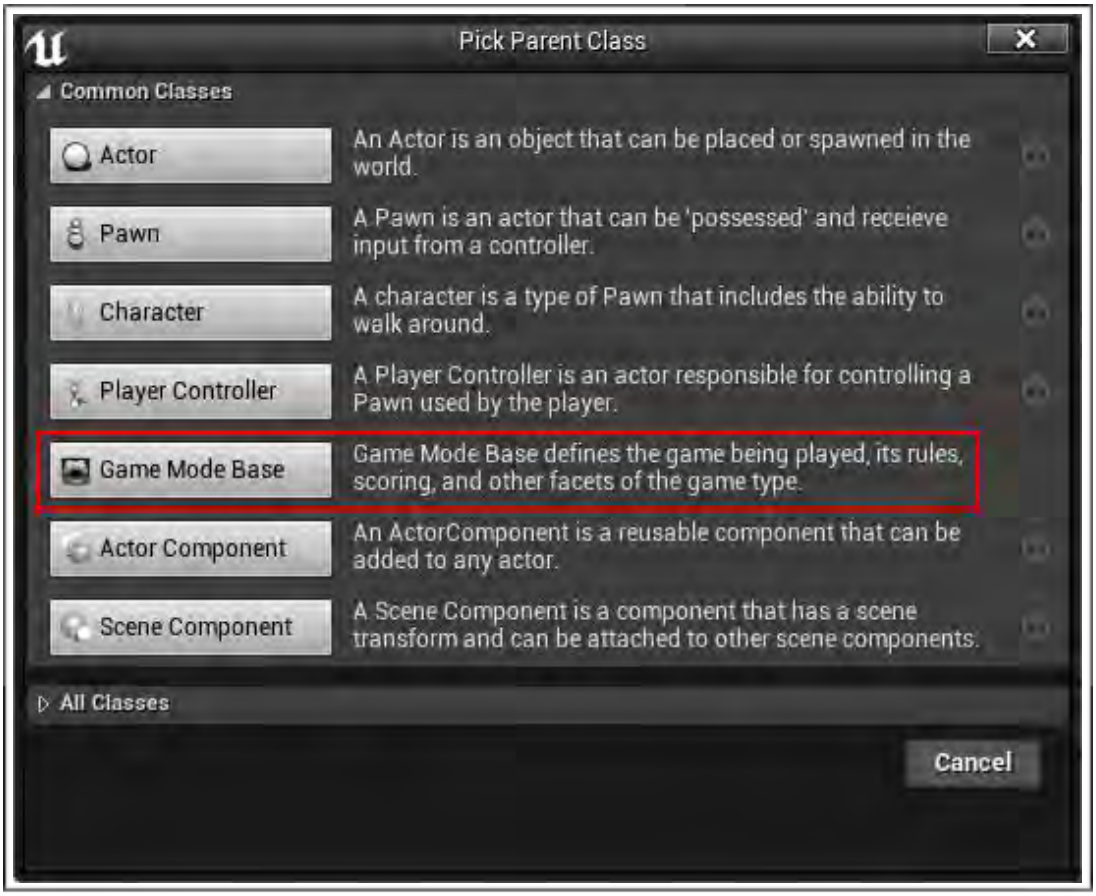
对于第一点，我们将通过创建一个新的 Game Mode 类的方式来实现。

创建 Game Mode

Game Mode 是虚幻 4 引擎中一个特殊的类，用于控制玩家如何进入游戏。例如，在多人在线游戏中，我们可以使用 Game Mode 来决定每个玩家出现的位置。当然，更重要的是 Game Mode 还将决定玩家使用哪个 Pawn 类。

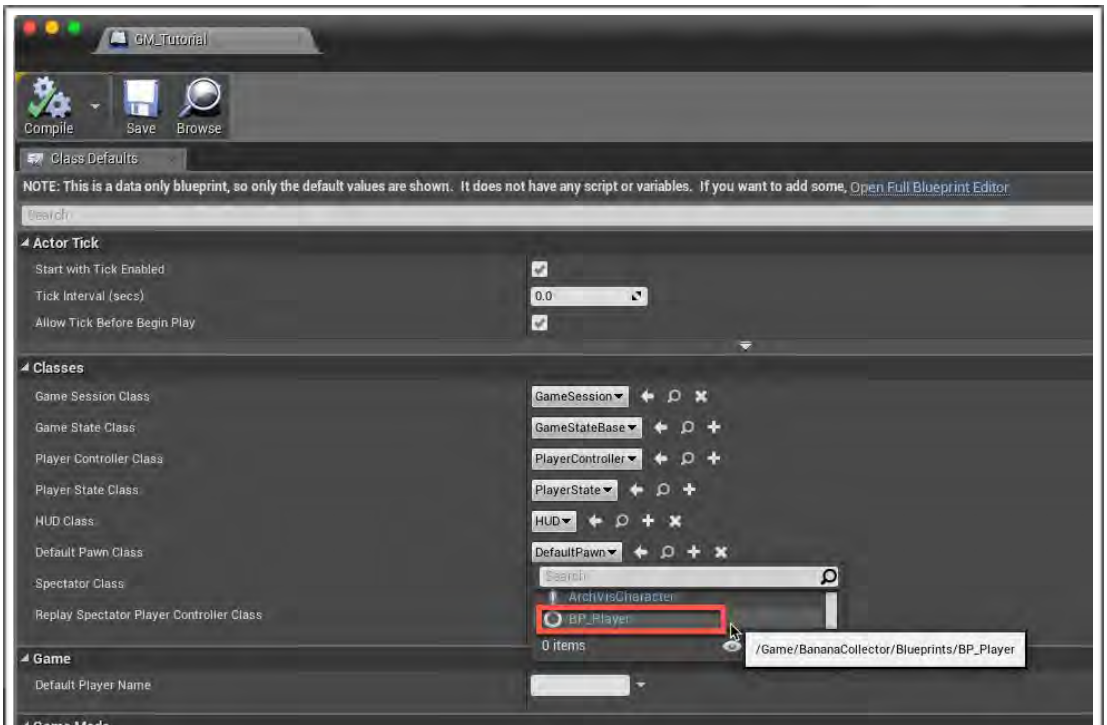
在 Content Browser 中打开 Blueprints 文件夹。点击 Add New 按钮，选择 Blueprint Class。

从弹出的界面中，选择 Game Mode Base，并将其更名为 GM_Tutorial。



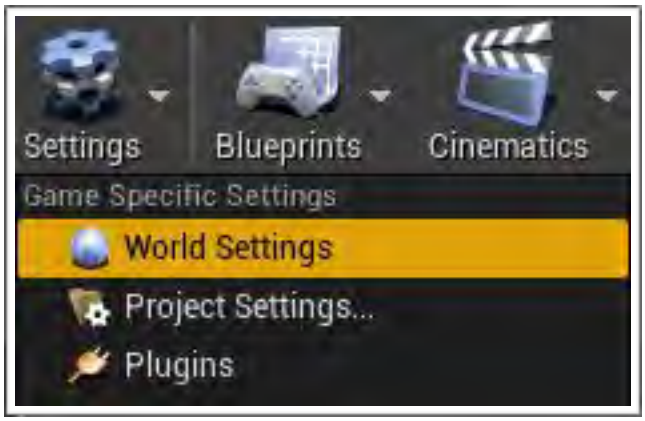
接下来我们需要指定默认的 *Pawn* 类。双击 *GM_Tutorial* 文件将其打开。

在新的界面中，找到 *Classes* 部分，然后在 *Default Pawn Class* 右边的下拉列表中选择

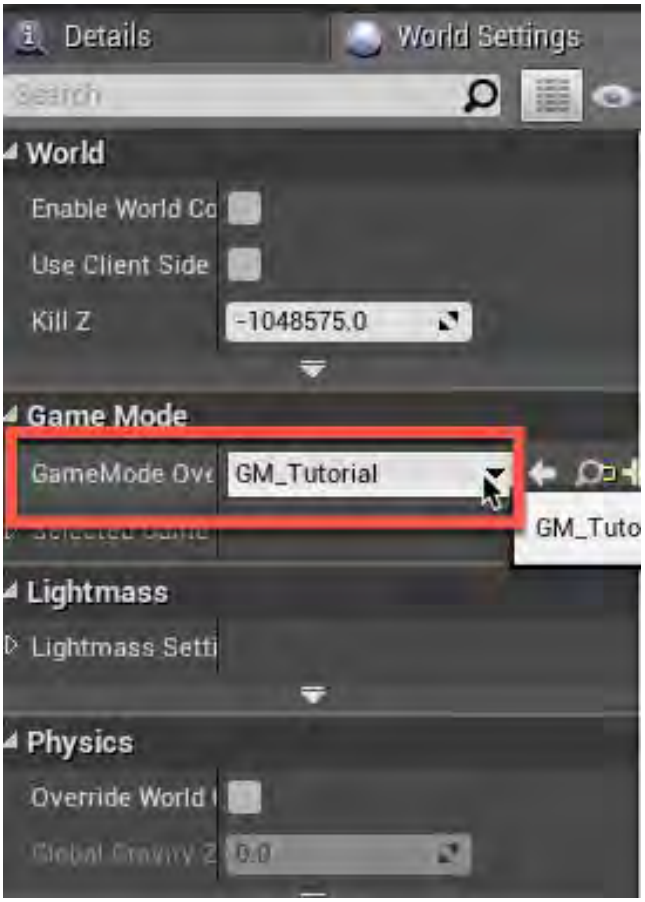


BP_Player。

在我们使用新的 Game Mode 前，需要让当前游戏场景知道所使用的 Game Mode 是哪个。关于这一点，我们可以在 World Settings 中进行设置。在此之前，点击工具栏上的 Compile 按钮，并关闭蓝图编辑器。



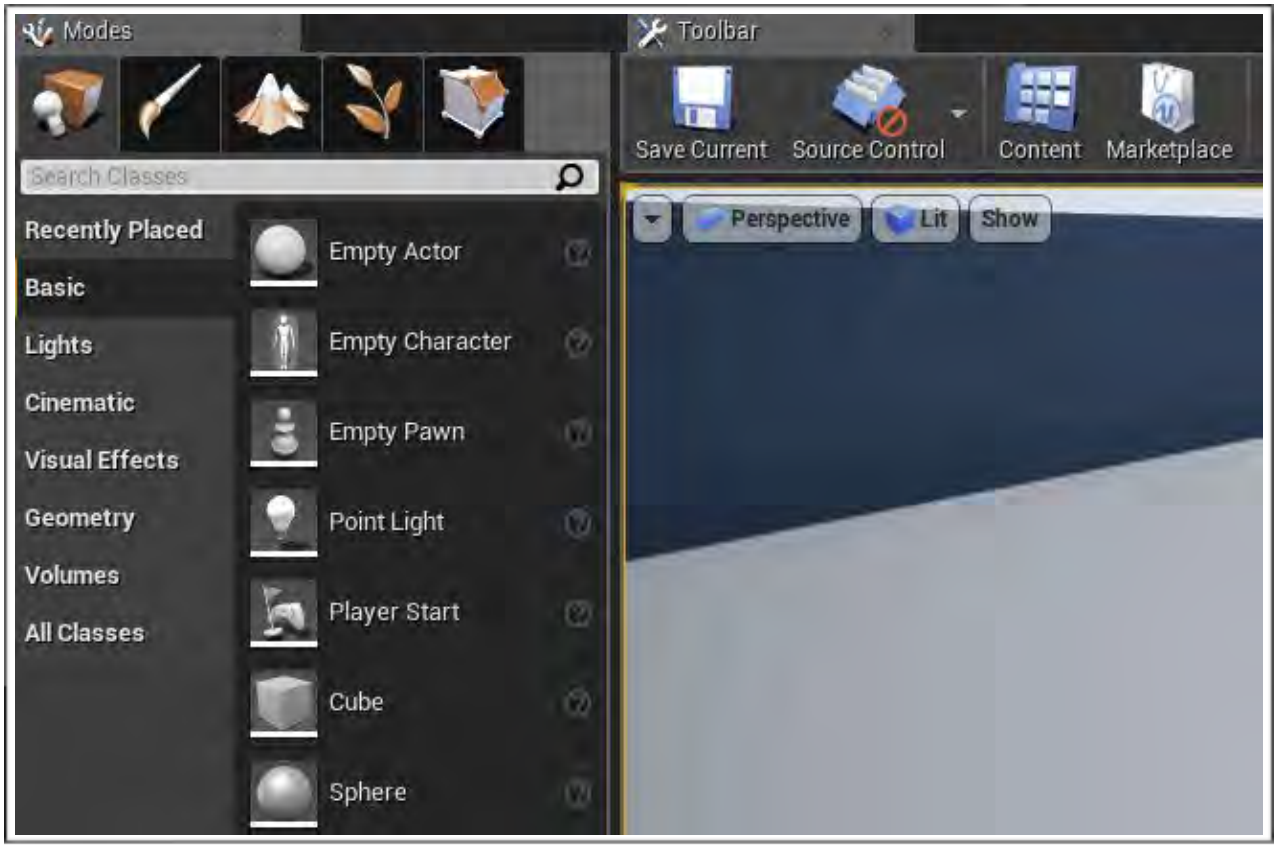
每个 Level（关卡）都有自己的相关设置。我们可以在菜单栏中选择 Windows-World Settings，也可以在工具栏中选择 Settings-World Settings。



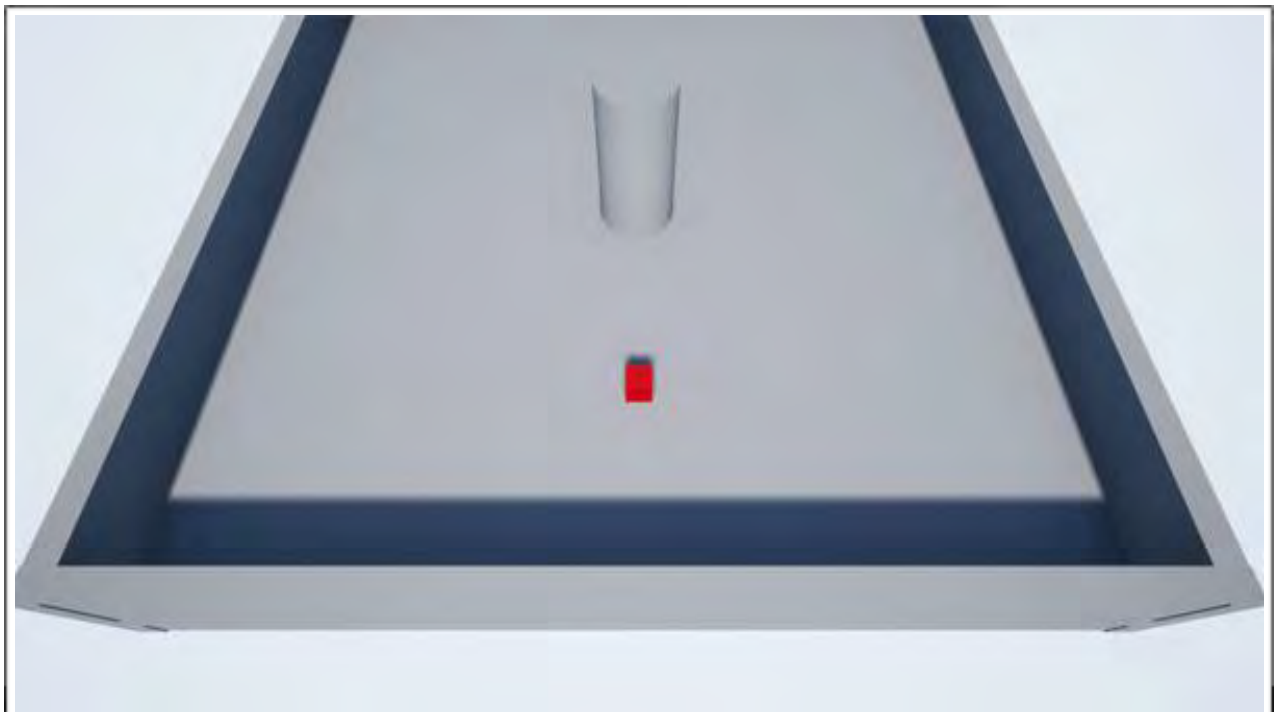
此时在编辑器的右侧会出现一个新的 World Setting 选项卡。
在这里，从 GameMode Override 的下拉列表中选择 GM_Tutorial。
最后，我们还需要指定玩家的出现位置。为此，我们需要在关卡中放置一个 Player Start 的 actor。
好了，本课的内容就先到这里，在下一课的内容中，我们将继续学习如何放置 Player Start。

欢迎继续我们的学习。

为了放置 *Player Start*（玩家起始），我们需要在虚幻 4 的主编辑器中找到 *Modes* 面板，然后搜索 *Player Start*。左键单击选中，并将其拖到 *Viewport* 视口中。



我们可以把它放置在关卡中的任何位置。完成后，可以在工具栏上点击 *Play* 预览游戏效果。玩



家角色将在我们放置 *Player Start* 的地方出现。

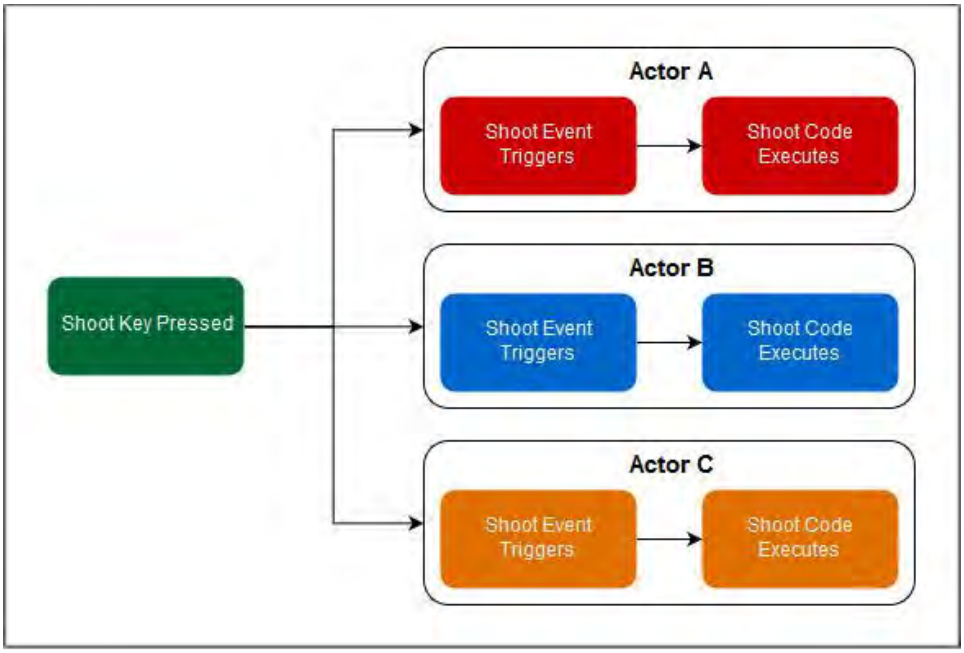
如果想要离开游戏，点击工具栏上的 *Stop* 按钮，或是按下 *ESC* 键即可。如果你看不到光标，那么可以按下 *Shift+F1*.

你可能已经感觉到了，这个游戏里面你连动都不动不了，这也算游戏？所以接下来的事情就是要进行输入设置。

输入设置

为某个动作设置键盘指令又被称之为 *key binding*（键盘绑定或关联）。

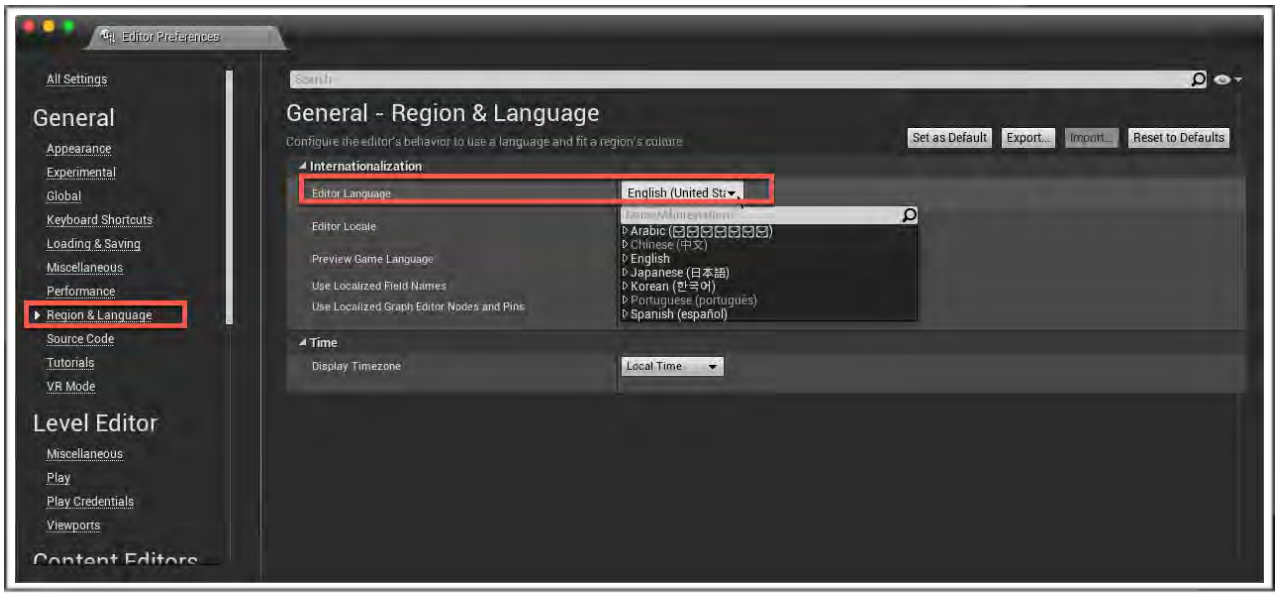
在虚幻 *4* 引擎中，我们可以通过设置 *key binding*，从而实现按下某个按键时触发一个事件。这些事件也是节点，当特定的动作发生时（比如当我们按下某个按键时），就会执行这些节点。当某个事件被触发时，跟该事件所关联的节点都会被执行。



这种键盘绑定的方式是有用的，因为它意味着我们无需对键盘上的按键进行硬编码。
例如，我们将鼠标左键单击设置为 *Shoot*,那么所有可以射击的角色都可以使用 *Shoot* 事件来了解玩家何时按下了鼠标左键。如果我们希望更改按键设置，那么只需要在输入设置中更改即可。而如果我们使用的是硬编码，那么就需要遍历每一个角色，并分别更改按键。

这里顺便提一下，如果你对自己的英文极其没有信息，也可以在初学阶段把编辑器的界面设置为中文的。

在主编辑器中点击 *Unreal Editor-Preferences*，然后在 *Editor Language* 中选择中文即可。



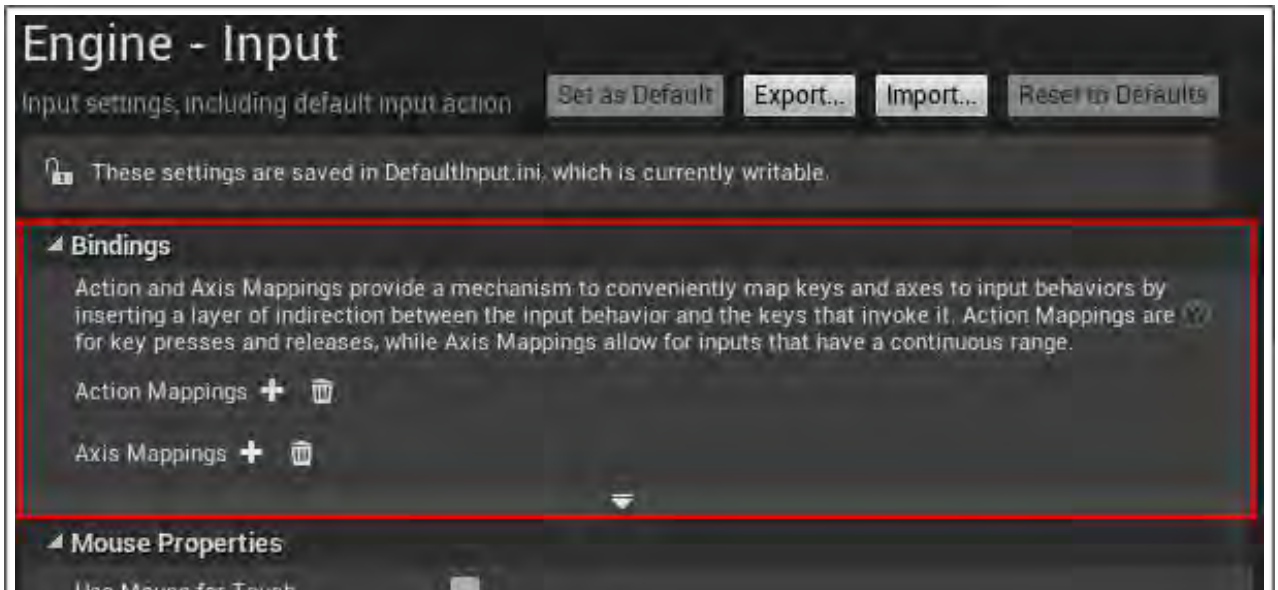
此时你会看到整个编辑器的界面都已经变成中文的了。

当然，在我们的学习过程中，仍然还是会使用英文界面。因为大量的官方和第三方优秀范例和教程都是英文的，要养成这个习惯。

Axis 和 *Action Mappings* 设置

为了查看输入设置，我们需要进入 *Edit-Project Settings*，在左侧选择 *Input*。

此时在界面右侧的 *Binding* 部分，可以设置游戏中的输入。



在虚幻 4 中提供了两种创建键盘绑定的方式：

1.Action Mapping：

对于这种绑定方式，只提供了两种状态，按下或没有按下。仅当我们按下或释放按键的时候才会触发动作事件。这类的键盘绑定方式适用于没有中间状态的事件，比如开火。

2.Axis Mapping:

这种绑定方式会提供一个名为 axis value 的数值，关于这一点，后面还会有更详细的解释。Axis 事件会在每一帧触发。这类的键盘绑定方式适用于需要手柄或鼠标的事件，比如持续性的运动。

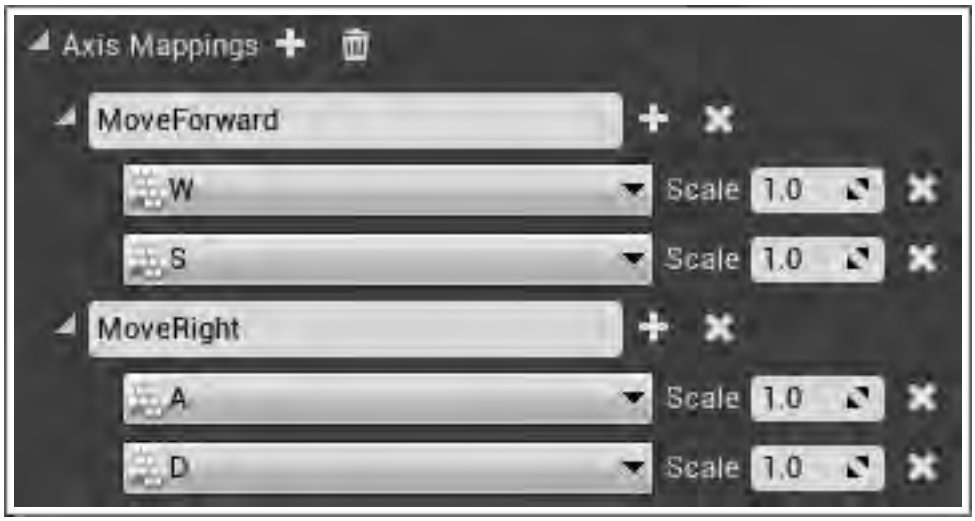
创建 Movement Mapping

对于我们这里来说，需要用到 axis mapping，因此首先需要创建两个 axis mapping group，通过使用群组，可以在单一事件上绑定多个按键。

创建新的 axis mapping group 很简单，只需要点击 Axis Mappings 右侧的加号。接下来创建两个组，并分别命名为 MoveForward 和 MoveRight。
注意这里一定不要选错了类型。

MoveForward 将用来处理向前和向后的运动，而 MoveRight 则用来处理向左和向右的运动。

接下来我们需要将运动分别映射到四个按钮上：W,A,S 和 D。在刚刚创建的 axis mapping group 中，只有两个空槽，只需要在每个 group 的旁边点击+标志，就可以添加新的 axis



mapping。
接下来添加具体的映射。

点击每个 axis mapping 右边的下拉列表，从中选择所需的按键，为 MoveForward 选择 W 和 S 键，为 MoveRight 选择 A 和 D 键。
特别说明的是，因为虚幻 4 引擎的运行非常耗费资源，因此如果你的电脑性能一般，那么在执行虚幻 4 引擎的某些操作时，会发现没有立即发生变化，而是要稍等一会儿。这是正常现象，要解决这一现象的唯一方式就是更换性能配置更高的电脑。

接下来我们需要设置 Scale 的值。

Axis Value 和 Input Scale

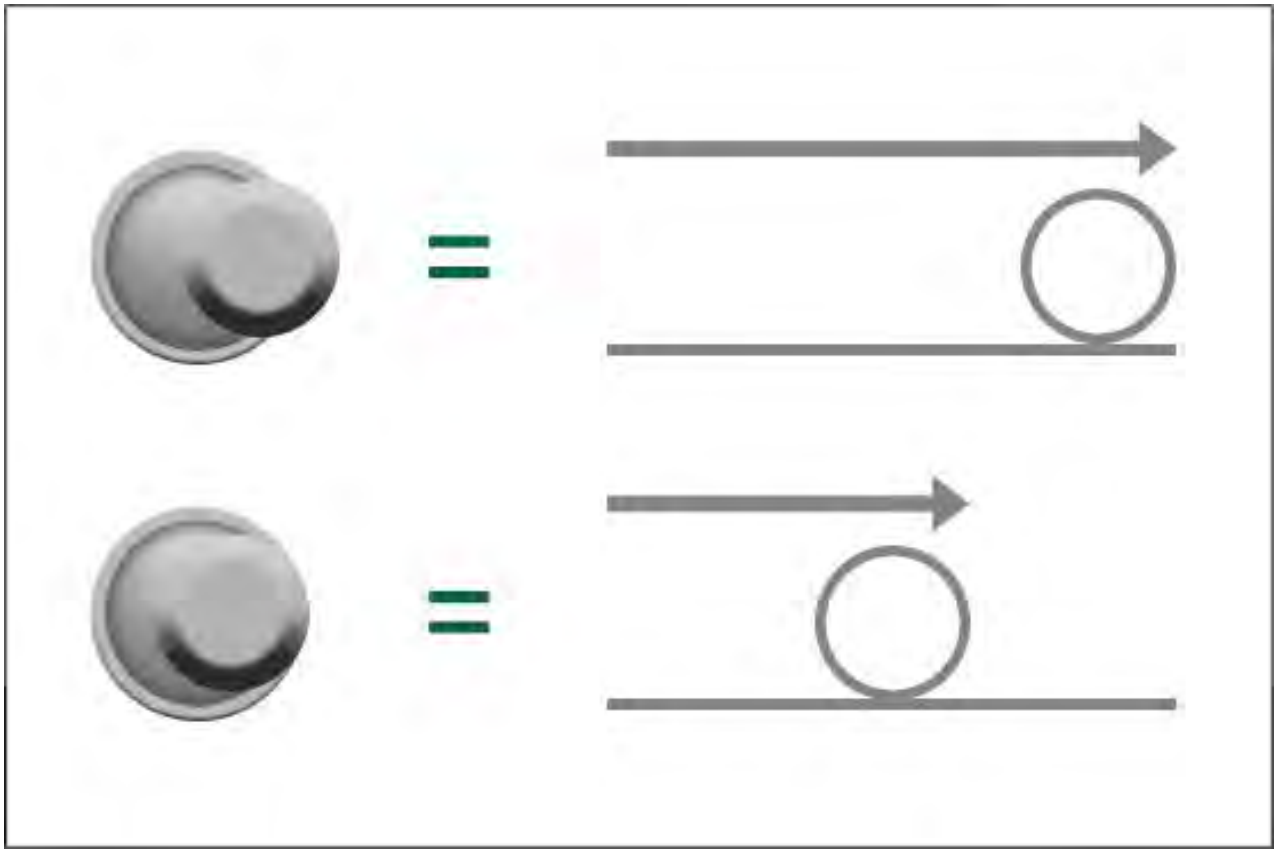
在设置 Scale 字段的具体数值之前，我们需要了解 axis value 的使用方法。

Axis value 是一个数值，由我们所设置的输入类型及输入方式来决定。当按下按钮或按键时，会输出 1，而手柄/操纵杆的输出值在 -1 和 1 之间，具体的数值取决于按下的方向和力度。

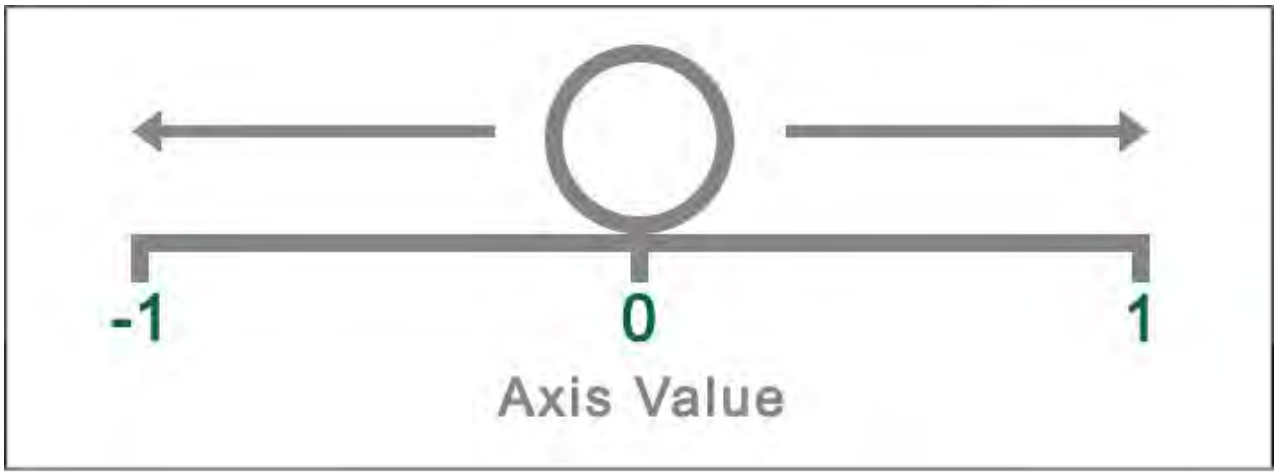
我们可以使用 axis value 来控制 Pawn 的运动速度。例如，当我们把手柄推向最边缘时，axis value 的数值为 1，而如果推到一半的位置，那么数值为 0.5。

通过将 *axis value* 乘以一个速度变量，就可以使用手柄来控制角色的速度了。

我们还可以使用 *axis value* 来指定沿着某个轴的运动方向。如果我们将某个 Pawn 的速度乘以



正的 *axis value*，那么就会得到一个正向的位移，反之会得到一个负向的位移。将这个位移添加到 Pawn 的位置上，就可以最终确定它的运动方向。
由于键盘上的按键只能输出 1 或者 0，我们需要使用 Scale 值将其转换成负值。将 *axis value* 的值乘以 Scale 的数值就可以得到最终的值。



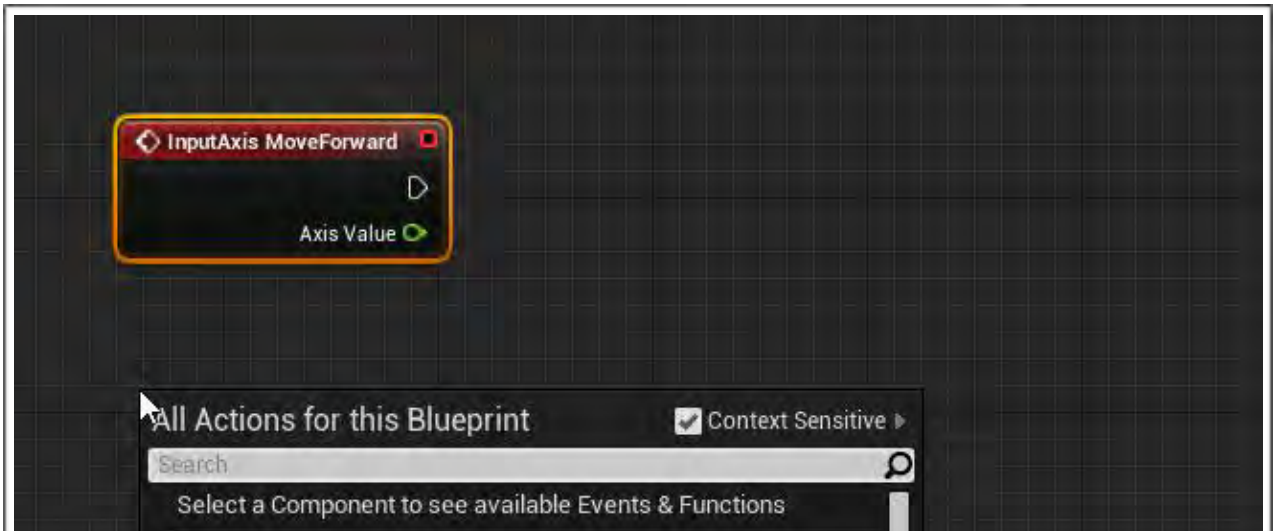
当我们使用正的 axis value 乘以负的 scale 值时，就会得到负的最终值。

在这里，让我们将 S 和 A 对应的 Scale 值设置为-1。

接下来的操作就有意思了！我们将让 Pawn 角色在游戏场景中真正动起来！关闭 Project Settings，然后在主编辑器中找到并打开 BP_Player 这个蓝图文件。

让玩家动起来

首先我们需要获取运动映射的事件。在 Event Graph 视图中右键单击空白区域，从弹出的菜单中搜索 MoveForward，注意这里要添加的是 Axis Events 下面的 MoveForward 节点，而不是 Axis Values 下面的 MoveForward。



重复以上操作添加 *MoveRight* 节点。

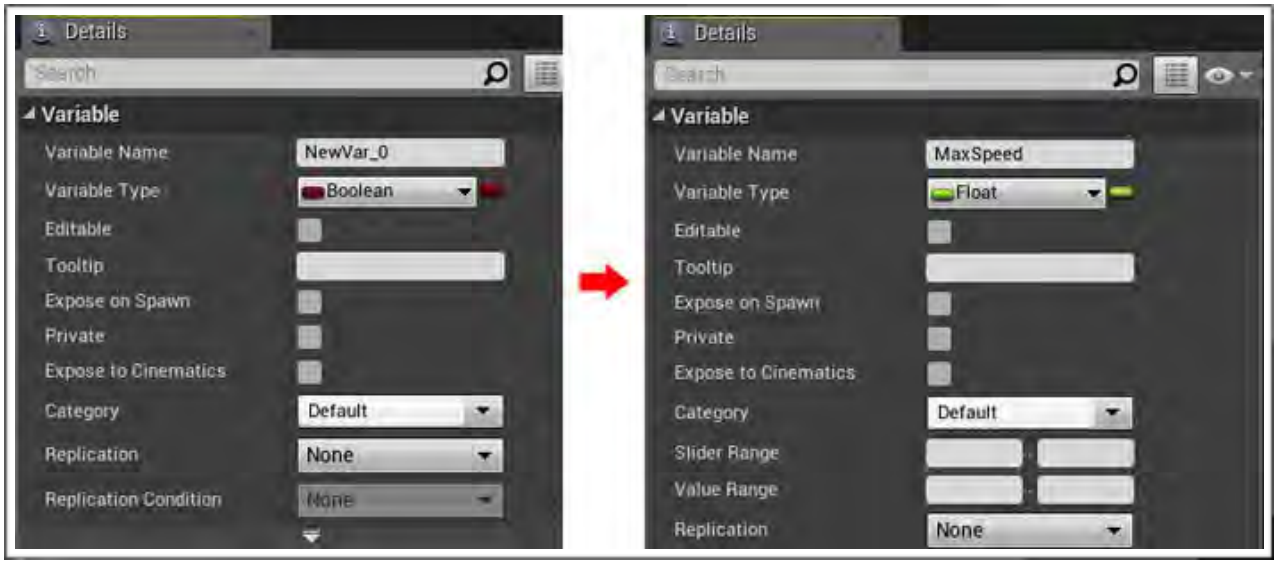
接下来我们将设置 *MoveForward* 节点。

使用变量

为了让玩家动起来，我们需要指定 *Pawn* 的移动速度。一个简单的方式是使用变量来存储移动速度。

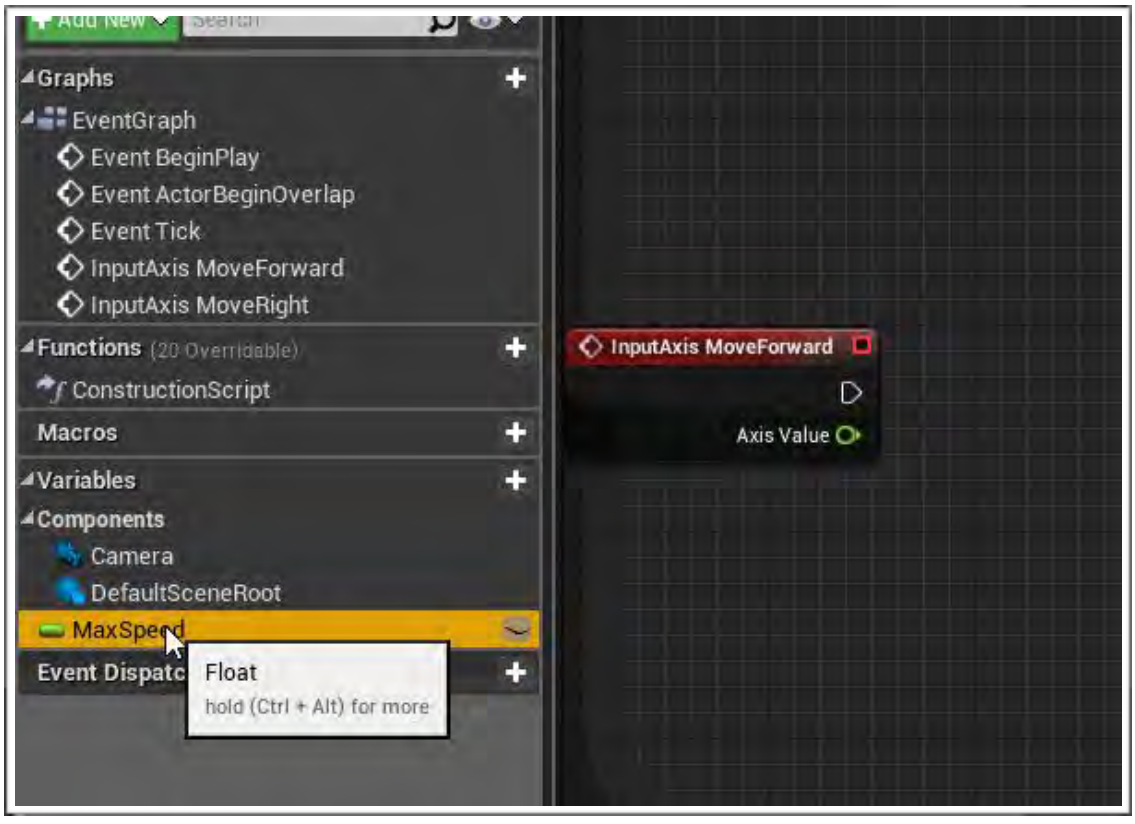
将新创建的变量命名为 *MaxSpeed*。然后在 *Details* 视图中设置变量类型为 *Float*。为此，我们需要在 *Variable Type* 旁边的下拉列表中选择 *Float*。

接下来我们需要设置该变量的默认数值。在此之前，需要点击工具栏上的 *Compile* 按钮。

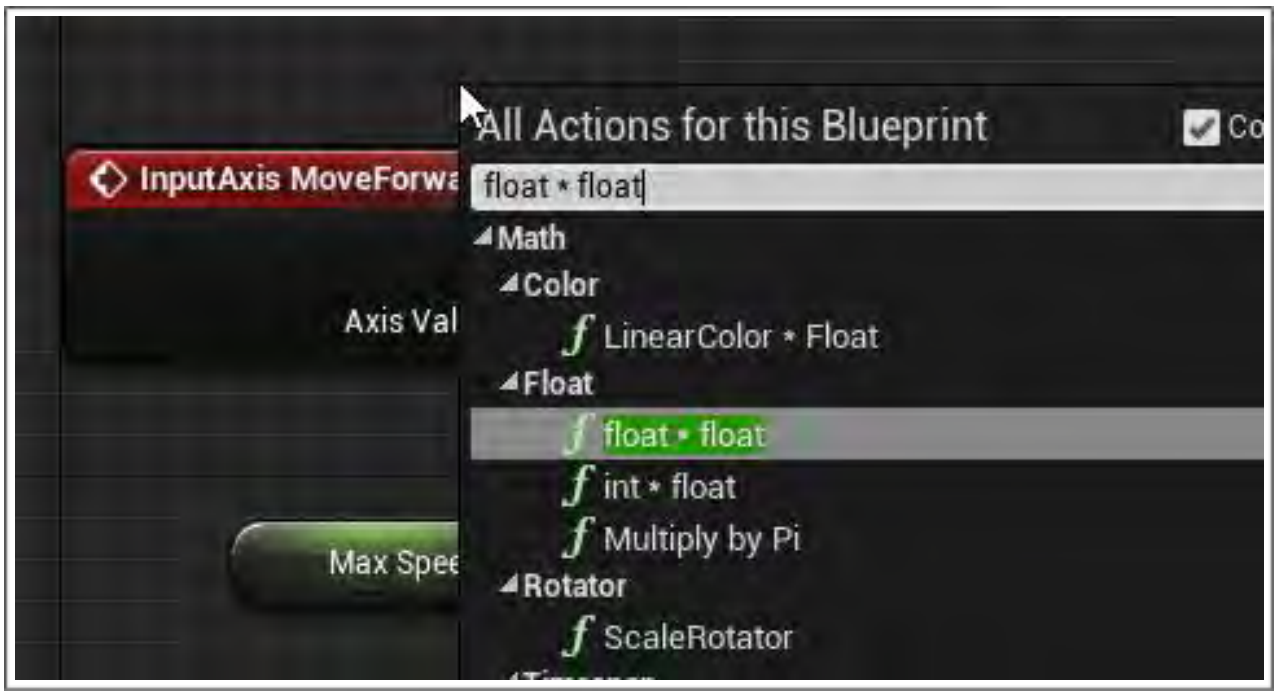


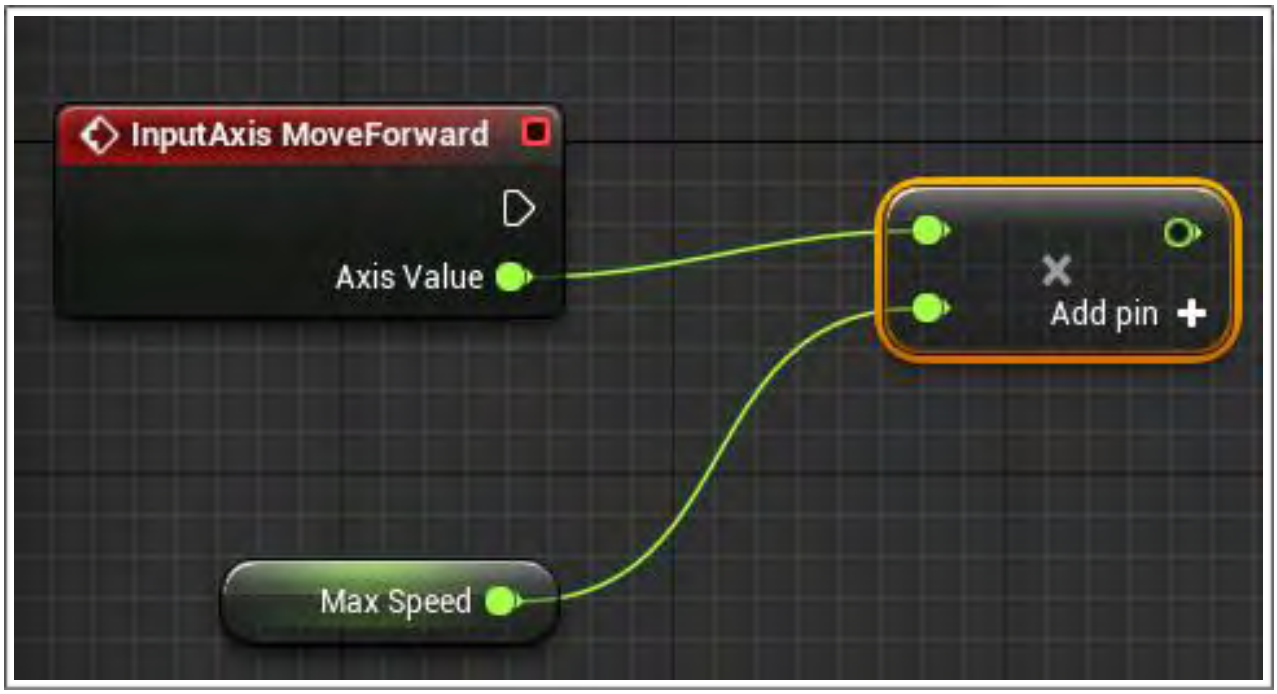
保持选中所创建的 MaxSpeed 变量，在 Details 视图中找到 Default Value 部分，并将 Max Speed 的默认数值更改为 10.

然后使用鼠标将 MaxSpeed 变量从 My Blueprint 视图拖动到 Event Graph 视图中，选择弹出菜单中的 Get MaxSpeed。



现在我们需要将 MaxSpeed 和 axis value 相乘，以获得最终的速度和方向。在 Event Graph 中添加一个 float * float 节点，然后将 Axis Value 和 MaxSpeed 都连接到上面。

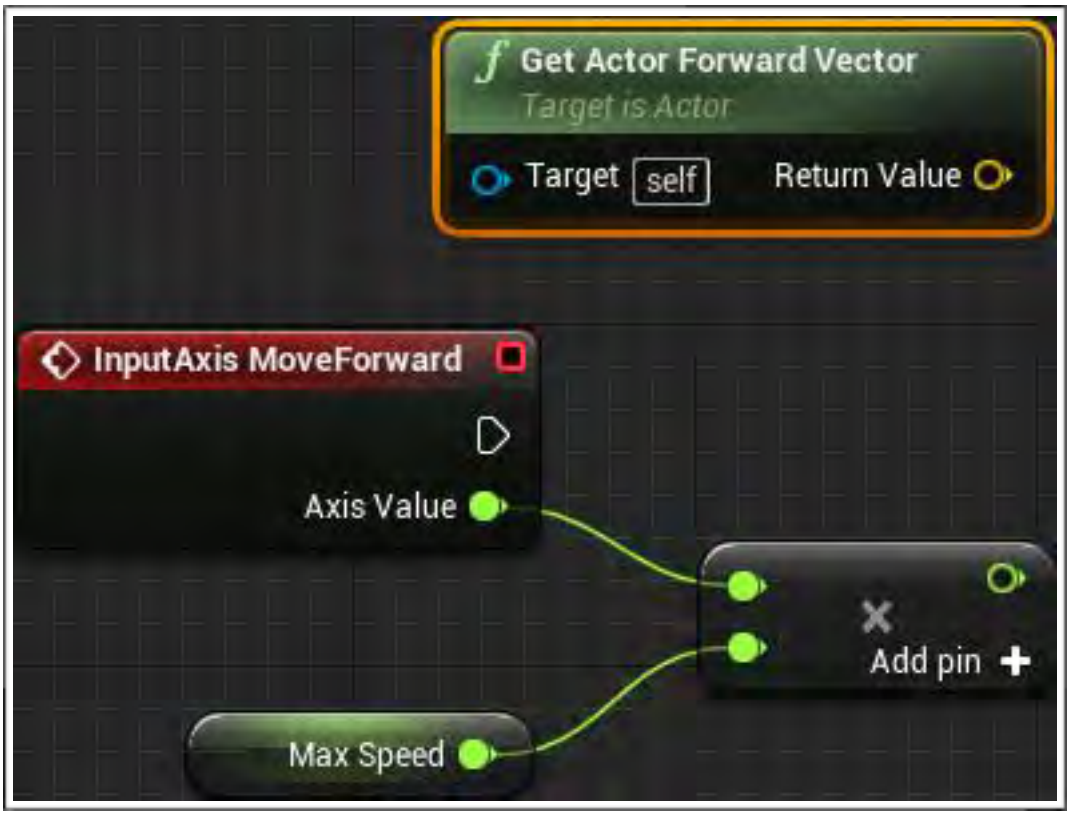




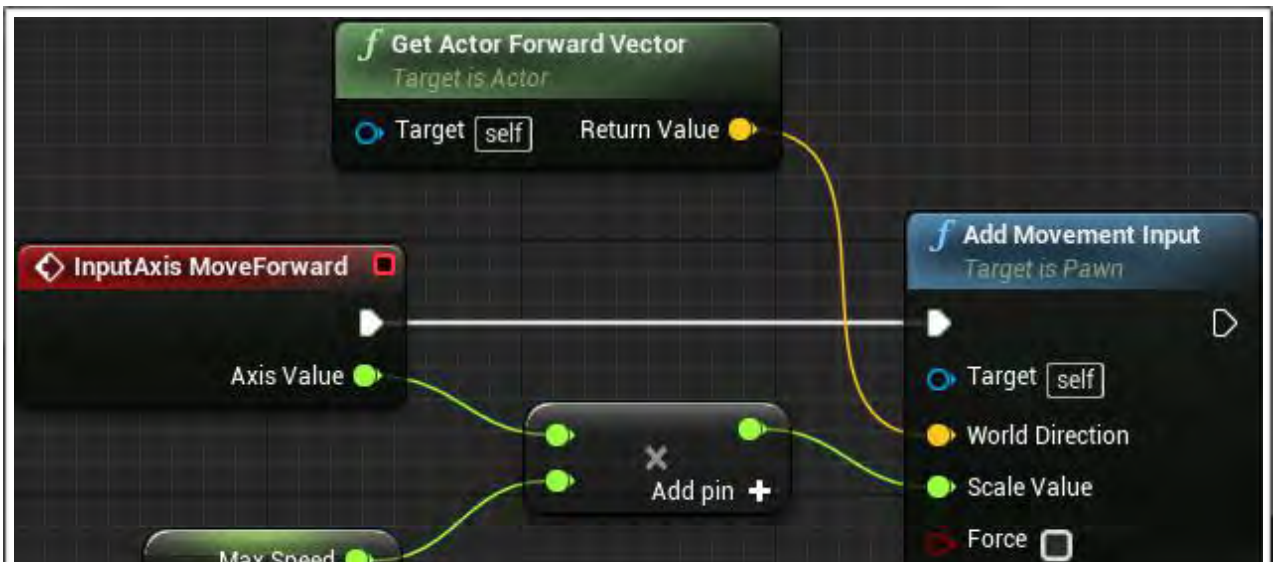
获取玩家的运动方向

为了让玩家向前运动，除了设置玩家的运动速度，还需要了解 Pawn 的朝向。幸运的是，虚幻 4 为我们提供了一个节点可以解决此问题。

在 Event Graph 中添加一个 Get Actor Forward Vector 节点。



接下来让我们添加一个 Add Movement Input 节点。



该节点将获取运动方向和数值，并将其转换成一个偏移量，按照下图的方式来连接节点。
当你看到这么复杂的连接线，可能有点不知所措。其实不用担心，白色的连接线代表执行链，换句话说，当玩家触发了 MoveForward 的输入事件后，就会执行 InputAxis MoveForward 节点，而白线则代表着一旦执行完该节点，紧接着会执行 Add Movement Input 节点。

此外，有个小小的技巧。上一节点的黄色输出接口必须连接到下一节点的黄色输入接口，绿色输出接口必须连接到下一节点的绿色输入接口，其它颜色的输出和输入接口也必须遵循这样的原则，否则就会出现错误。

在 Add Movement Input 节点中有五个输入接口，这里分别来解释一下：

1.白色的右三角
代表执行链，也是游戏的执行逻辑顺序
2.蓝色的 Target 输入接口

代表执行对象，这里设置为 self,也就是玩家自身（红色的方块）

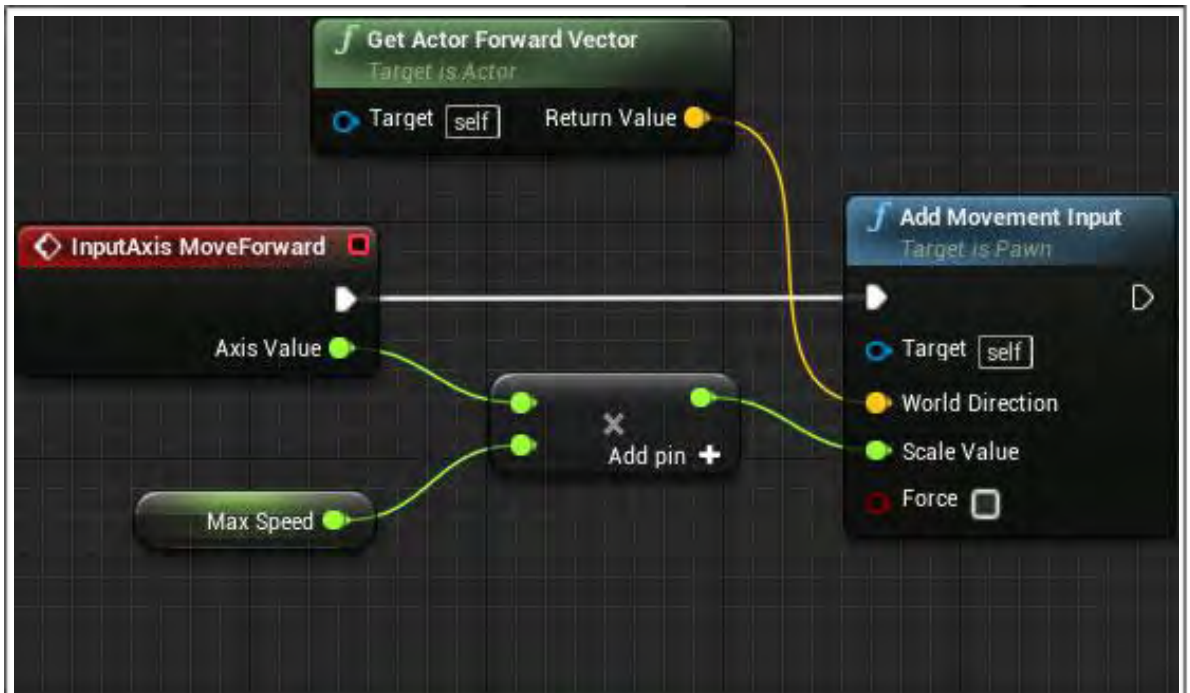
3.黄色的 World Direction 接口

代表目标的移动方向，这里指的是玩家的朝向

4.绿色的 Scale Value 接口

代表目标的移动速度，这里是使用 MaxSpeed 乘以 axis value（范围在-1 到 1 之间）

5.红色的 Force 接口
代表受力的方向，这里没用到，暂且不管

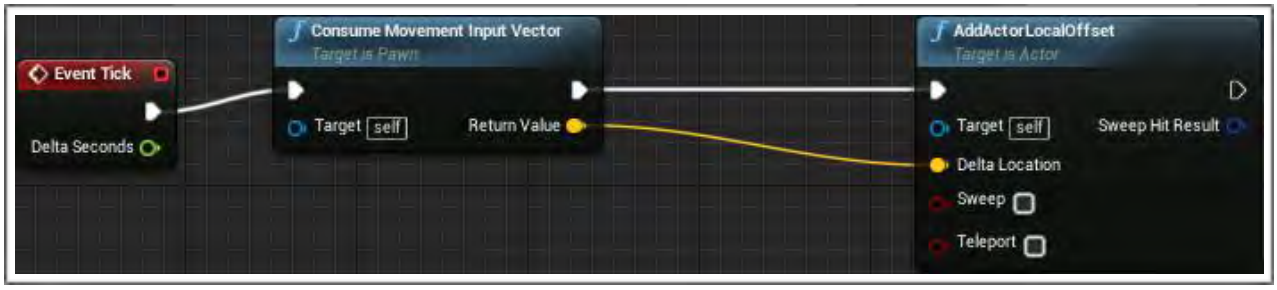


完成以上操作后，为 MoveRight 事件进行类似的设置，只不过需要将 Get Actor Forward Vector 更改为 Get Actor Right Vector。试试看，在不参考上面操作指引的情况下你能完成多少~

添加 Offset

为了让 Pawn 角色真正动起来，我们需要获取刚刚通过 Add Movement Input 节点所计算的偏移量，并将其添加到 Pawn 的位置上。

通常来说，我们希望在游戏的每一帧让玩家角色移动一点距离，而不是来个瞬移或者空间跳跃，



因此我们需要将运动添加到 Event Tick 事件上，因为它会每帧触发。

在 Event Graph 视图中找到 Event Tick 节点，如果没有就添加一个。

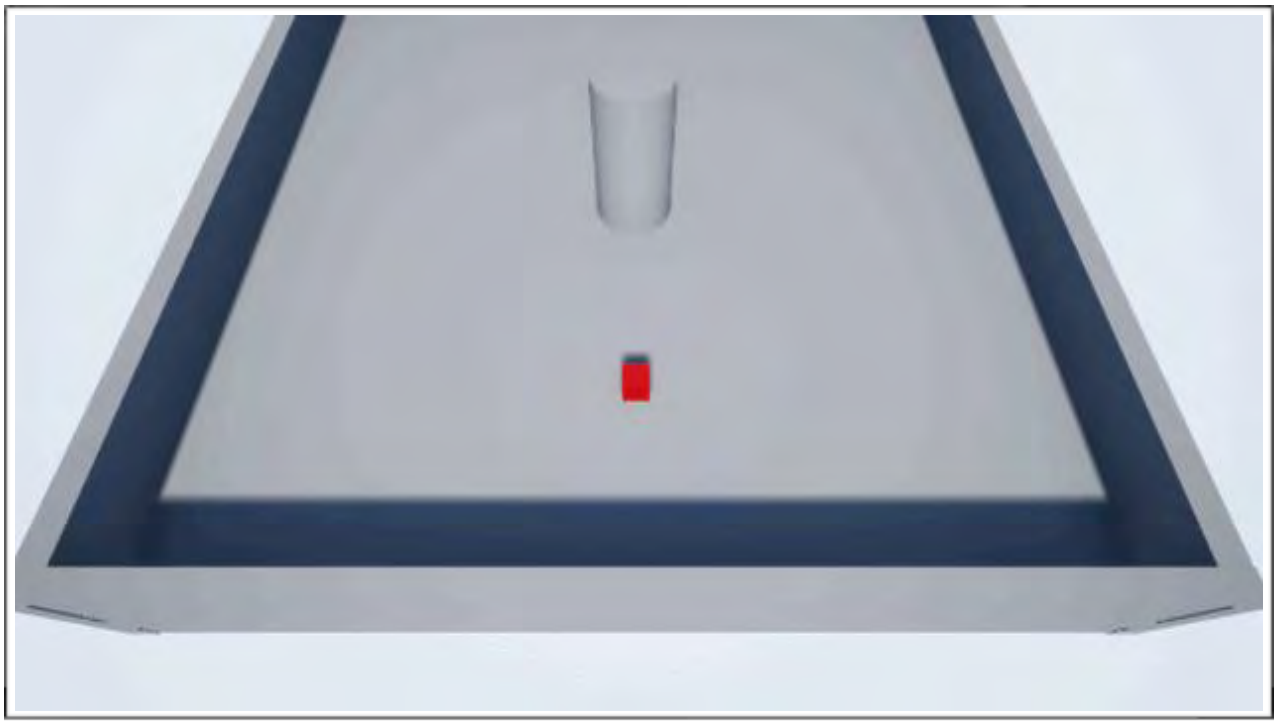
为了获取位置偏移，需要创建一个 Consume Movement Input Vector 节点。同时，为了添加 offset，需要创建一个 AddActorLocalOffset 节点。完成后，使用类似下图的方式来连接这些节点。

通过以上操作，我们可以在游戏的每一帧获取所保存的运动输入信息，然后将其添加到角色的当前位置上。

点击 *Compile* 按钮，然后回到主编辑器，点击 *Play* 预览游戏效果。现在我们可以使用键盘上的四个键来自由移动了！

不过这里有一个小小的问题。配置性能更高的电脑会有更高的游戏帧速。而因为 *Event Tick* 会在每帧调用，因此运动节点会被更频繁的执行。因此在更高端的电脑上，*Pawn* 的运动速度会更高，反之亦然。

为了修复这个问题，我们需要让角色的运动做到 *frame rate independent*（和游戏帧速无关）。



Frame Rate Independence

如果我们能实现独立于游戏帧速，那么不管游戏的实际运行帧速是多少，都会有相同的结果。幸运的是，在虚幻 4 中要实现独立于游戏帧速并不是那么困难。

退出当前的游戏，然后在主编辑器中打开 BP_Player 蓝图文件。接下来找到 Event Tick 节点，然后会看到一个 Delta Seconds。

Delta Seconds 指的是从上一次 Event Tick 到现在的时间间隔。通过将位置偏移乘以 Delta Seconds，我们就可以做到运动和帧速无关。

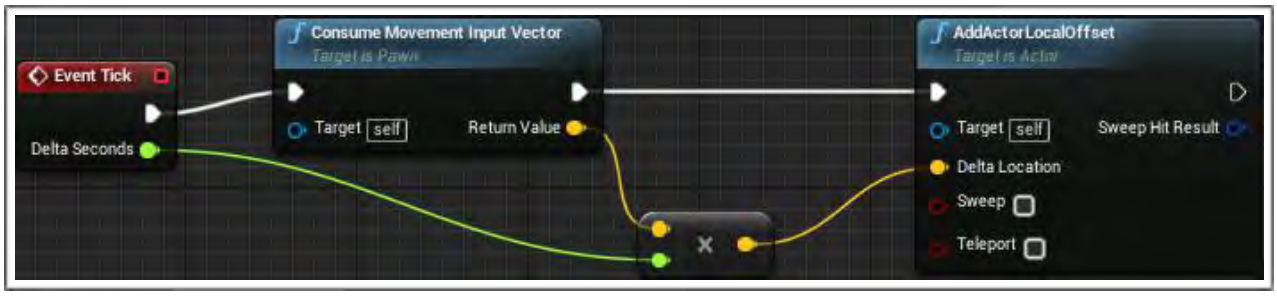
假定 Pawn 的最大速度是 100，如果从上一次 Event Tick 开始过去了一秒，那么 Pawn 应该移动了 100 个单位。如果过去了半秒，那么它应该移动了 50 个单位。

如果角色的运动是独立于帧速的，那么 Pawn 角色就会每帧运动 100 个单位，而和每两帧之间

的 实际时间间隔无关。

为此，我们需要添加一个 vector * float 节点，然后使用下图的方式来连接节点。

因为每两帧之间的时间间隔(Delta Seconds）非常小，所以这样的话 Pawn 的运动速度会超级



慢。为此，我们需要将 *MaxSpeed* 的默认数值设置为 *600*.

恭喜你，至此我们就成功的实现了和帧速无关！

好了，本课的内容到此结束。

在上一课的内容中，我们成功的实现了让玩家角色可以在游戏场景中自由移动。但是如果你仔细观察，会发现代表玩家角色的方块可以穿越所遇到的一切障碍，显然这是不科学的~

为此，我们需要了解 Collision（碰撞）的概念。

实现碰撞检测

可能你还在奇怪为什么方块没有发生碰撞，因为它似乎有一个 collision mesh。需要注意的是，在虚幻 4 中，只有根组件才会进行碰撞检测。而我们的 Pawn 的根组件中不包含任何碰撞检测，因此可以穿越所碰到的一切障碍。

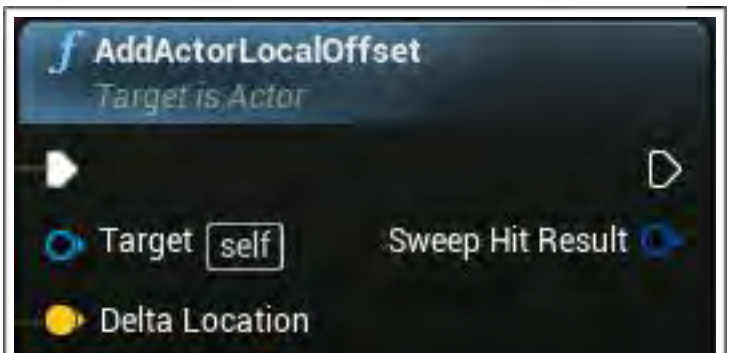
注意：一个在根组件中没有碰撞检测的角色仍然可能阻挡其它角色。但是仅限于原地，一旦我们让这个 actor 动起来，它不会和任何物体发生碰撞。

因此，为了使用 collision mesh，必须让 StaticMesh 成为根节点。

打开 Epic Games Launcher，打开 BananaCollector 项目，从 Content Browser 中找到 Blueprints 目录中的 BP_Player，双击在蓝图编辑器中将其打开。

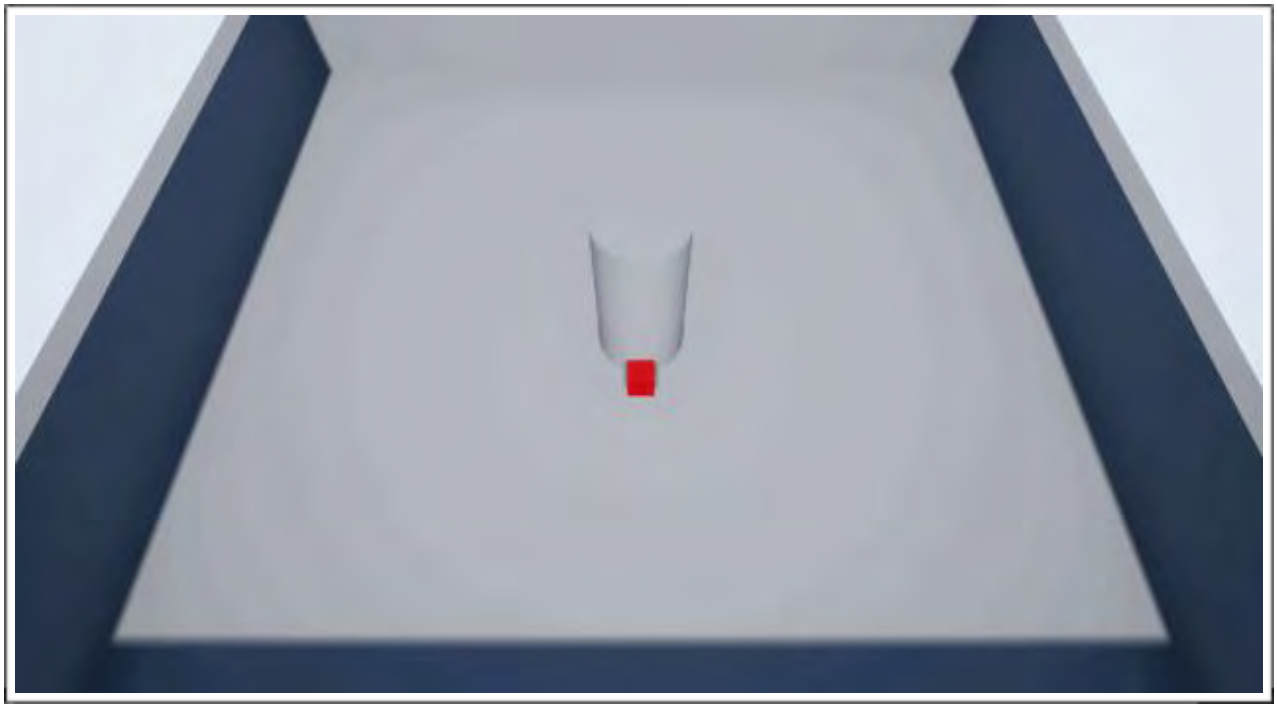
在 Components 面板中，将原来作为 DefaultSceneRoot 子组件的 StaticMesh 拖曳到 DefaultSceneRoot 的位置，使其成为新的根组件。

当然，为了让碰撞生效，还有一件事要做。



切换到蓝图编辑器中的 Event Graph 视图，找到 AddActorLocalOffset 节点，找到 Sweep 这个输入接口，并勾选右侧的选框，使其值为 true。

AddActorLocalOffset 的作用是让角色传送到新的位置，而 Sweep 选项则确保角色会和新旧位置之间的任何物体发生碰撞。



点击 Compile 保存蓝图的修改。

一切就绪，让我们返回虚幻 4 的主编辑器，点击工具栏上的 Play 按钮预览游戏效果，现在方块就会跟关卡中的物体发生碰撞了！

创建可拾取的物品

理论上来说，只要是玩家可以收集的东西都可以算作游戏物品。因此，我们可以使用 BP_Banana 来作为游戏中的物品。

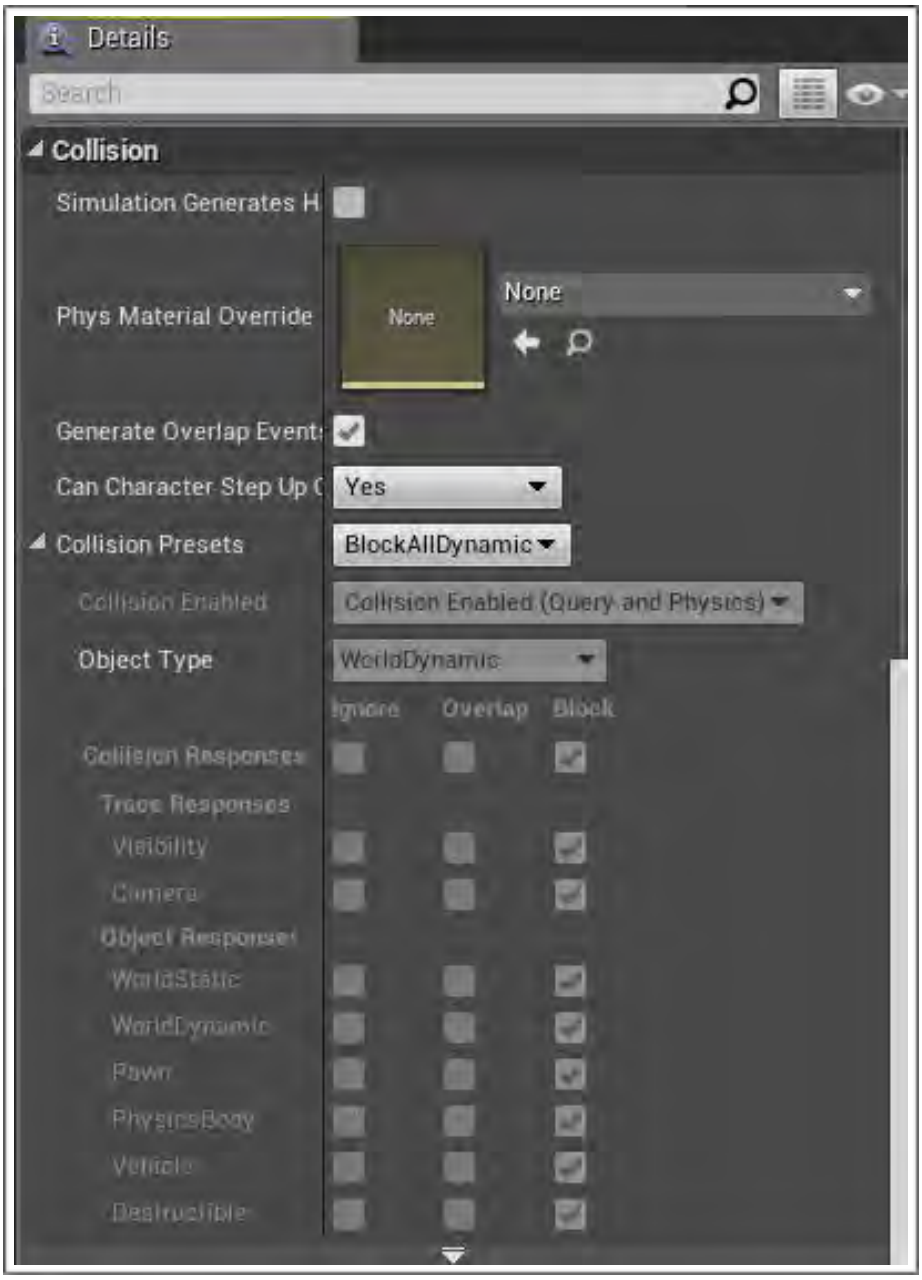
为了检测代表玩家角色的方块是否碰到了物品，我们需要一个事件节点，当发生碰撞时触发该节

Actor A	Actor B	Result
Ignore	Any response	Actors will pass through each other. No events will trigger.
Overlap	Overlap or Block	Actors will pass through each other. An overlap event will trigger for both actors.
Block	Block	Actors will block each other. A hit event will trigger for both actors.

点。我们可以使用 collision response（碰撞响应）来生成此类事件。
使用 collision response 还可以判断角色在碰到其它角色时如何响应。虚幻 4 中提供了三种类型的 collision response，分别是 Ignore,Overlap 和 Block。
下图显示了三种 collision response 之间的交互作用：
虽然我们既可以使用 Overlap，也可以使用 Block，但本教程中主要使用的是 Overlap。

设置 Collision Response

从 Content Browser 中找到并打开 BP_Banana 这个蓝图文件。选择 StaticMesh 组件，然后查看 Details 面板，我们可以在 Collision 部分设置 collision response。
如你所见，现在大部分的设置都是灰色的。为了让这些属性可编辑，首先我们要从 Collision Presets 右侧的下拉列表中选择 Custom。



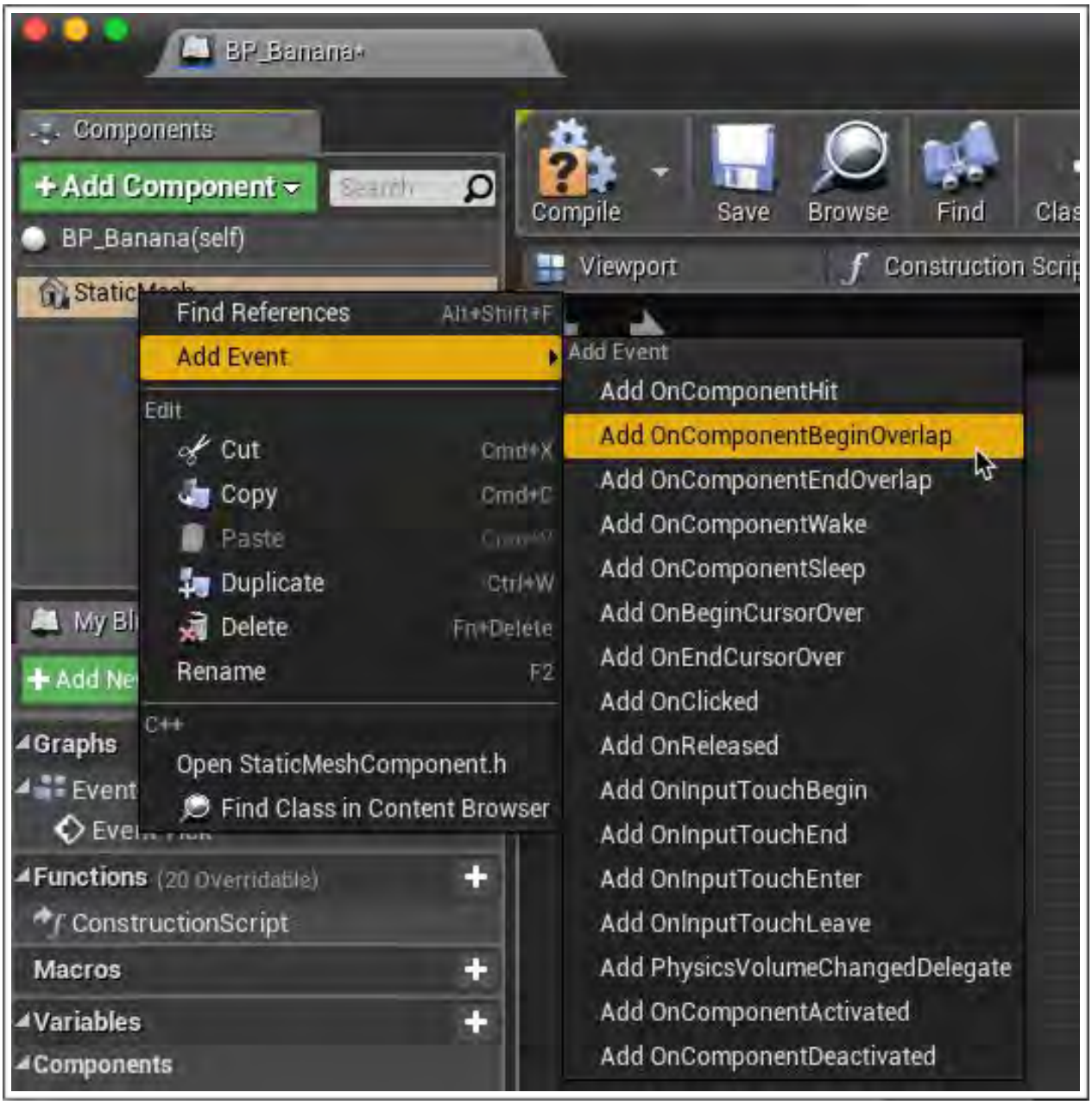
现在我们就可以指定游戏物品和玩家角色之间的 collision response 了。

可以看到，这里有一个 *Object Type* 属性，它只是为了方便把相似的角色分组在一起。关于 *object type*，可以查看官方文档中更详细的解释（ *https://docs.unrealengine.com/latest/INT/Engine/Physics/Collision/Reference/index.html* ）。

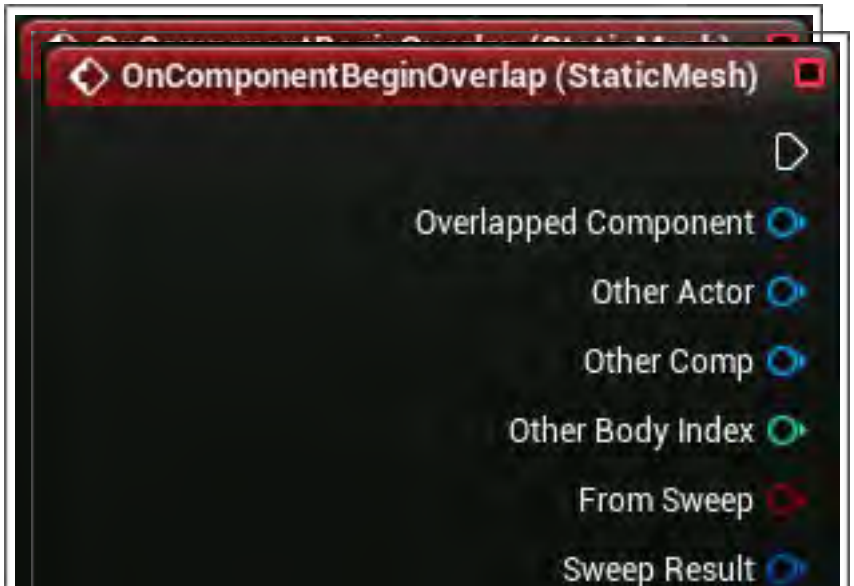
因为代表玩家角色的类型是 *WorldDynamic*，我们需要把游戏物品的 *collision response* 更改为同样的类型。
在 *Object Responses* 部分将 *WorldDynamic* 的 *collision response* 类型设置为 *Overlap*，如下图所示。

处理碰撞事件

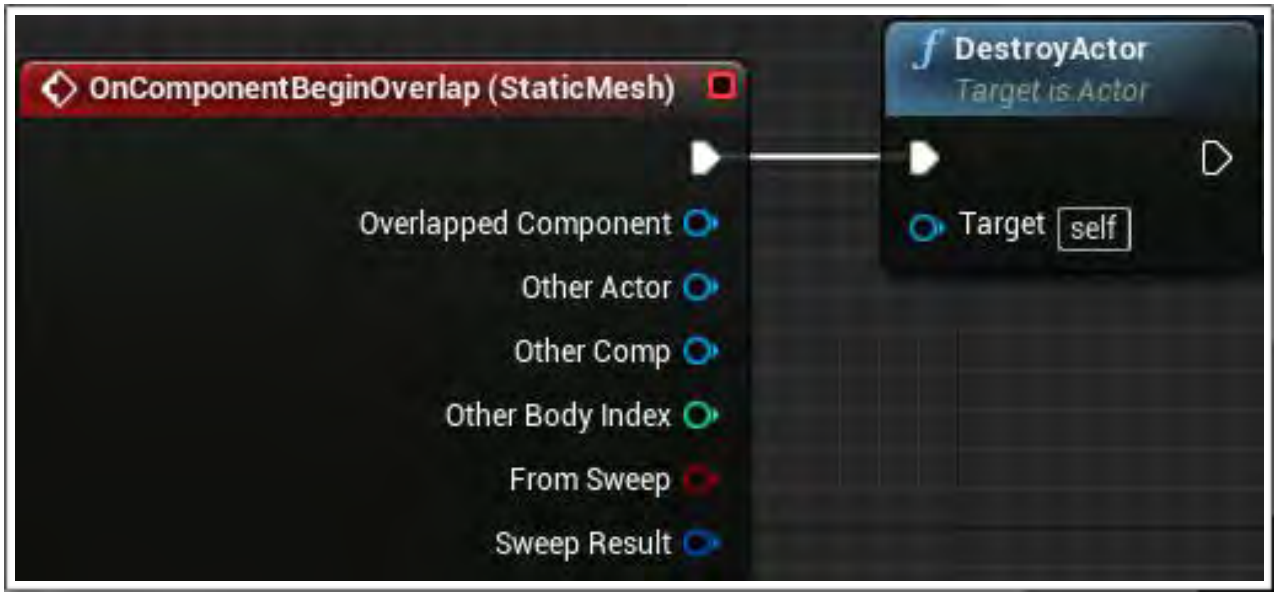
为了处理碰撞事件，我们需要使用 *overlap* 事件。在蓝图编辑器的 *Components* 面板中右键单击 *StaticMesh*。从弹出的菜单中选择 *Add Event-Add OnComponentBeginOverlap*。



在完成此步操作后，在 *Event Graph* 视图中会添加一个新的节点，如下图所示。



最后，在 Event Graph 视图中创建一个 DestroyActor 节点，并将其连接到刚刚创建的 OnComponentBeginOverlap(StaticMesh)节点上。从这个节点的名字可以看到，它的作用就是将目标角色从游戏中移除。不过因为这里没有设置其它 target，所以它销毁的对象就是自己。

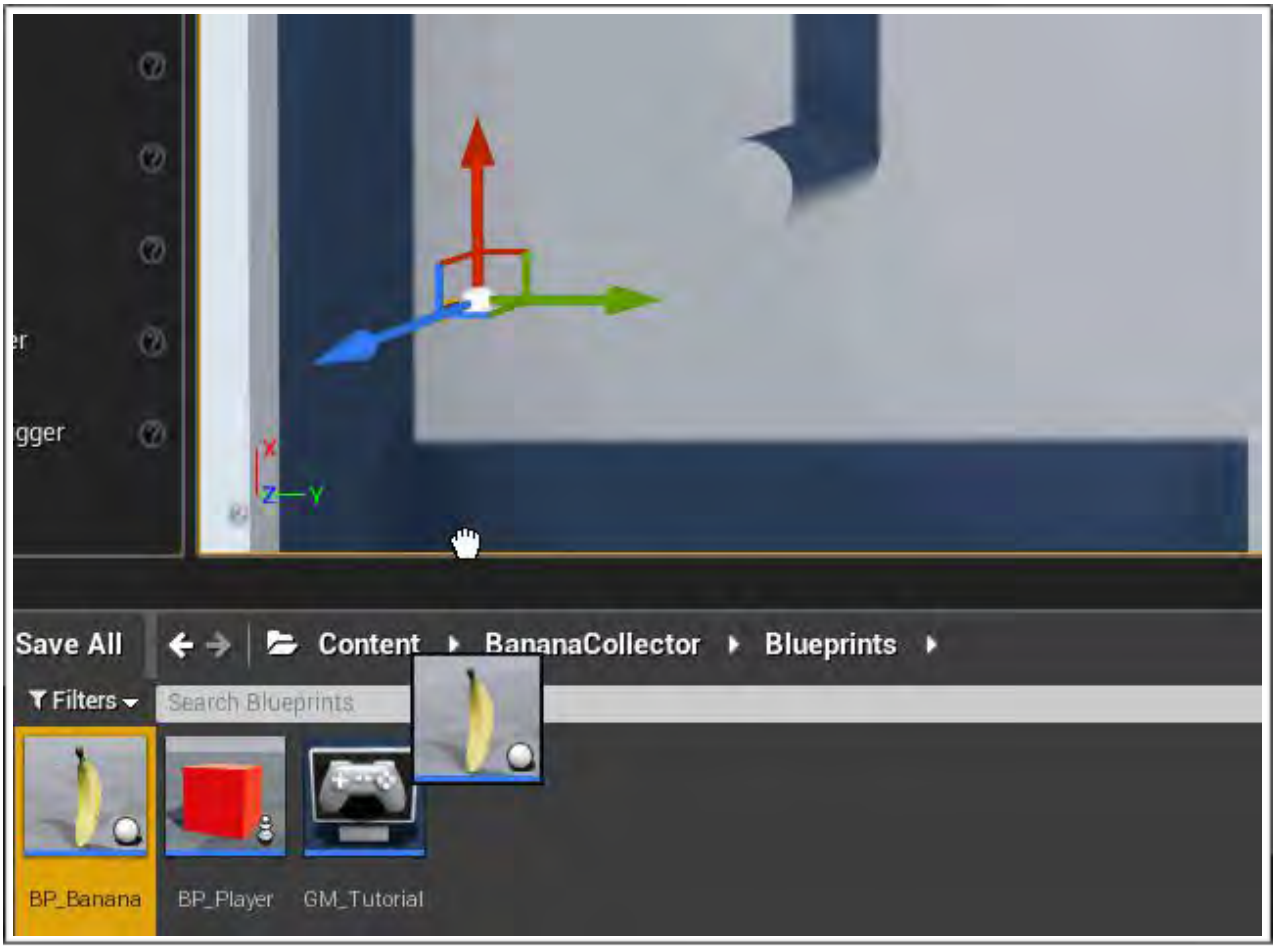


完成会点击蓝图编辑器工具栏上的 Compile 按钮保存更改，然后关闭当前的蓝图编辑器。

放置游戏物品

回到虚幻 4 的主编辑器，从 Content Browser 中找到 BP_Banana，并将其拖动到 Viewport 视口中。

根据自己的喜好，你可以在不同的位置放置多个香蕉 。



好了，现在点击工具栏上的 *Play* 按钮，就可以开始收集　了！

好了，本部分的内容到此结束，完整项目可以在这里下载。

在下一部分的内容中，我们将深入学习虚幻 4 引擎中的材质系统。

和真实世界类似，游戏中包含了很多物体，其中每一个都有自己的独特外观。在虚幻 4 引擎中，材质决定了物体的外观。一个物体是什么颜色？表面是否发光？看上去是否透明？这些都可以在材质中进行设置。

在虚幻 4 中，几乎任何一个视觉元素都可能会用到材质。我们可以将材质应用到不同的事物上，比如纹理、粒子系统和 UI 元素等。

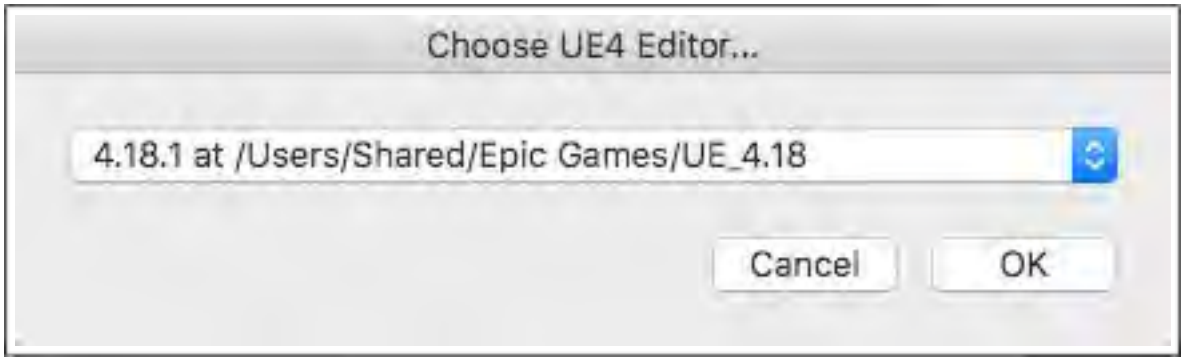
在这部分的教程中，我们将学习以下内容：

1.如何调整纹理，以更改亮度和色彩
2.使用 material instance 快速创建不同的变化
3.当玩家收集物品时，使用动态 material instance 来更改 avatar 的色彩。

此外，我们还会进一步熟悉材质编辑器和蓝图编辑器。如果你对这两部分内容一无所知，建议先阅读之前的教程内容。

开始前的准备

首先我们需要下载起始项目的相关文件，https://koenig-media.raywenderlich.com/uploads/2017/07/BananaCollectorStarter.zip

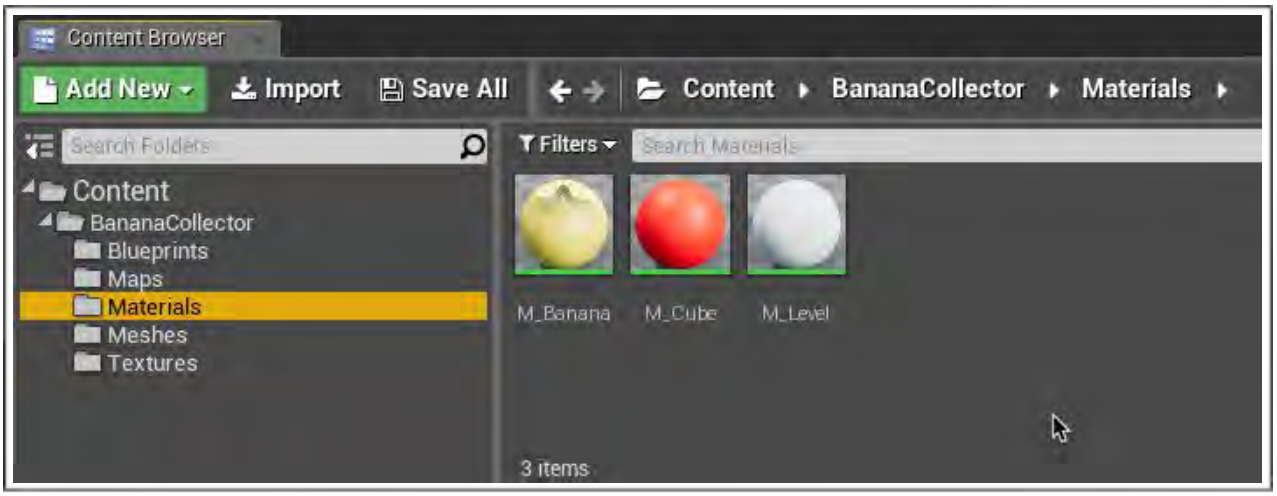


为了打开项目，跳转到项目目录，并打开 BananaCollector.uproject。

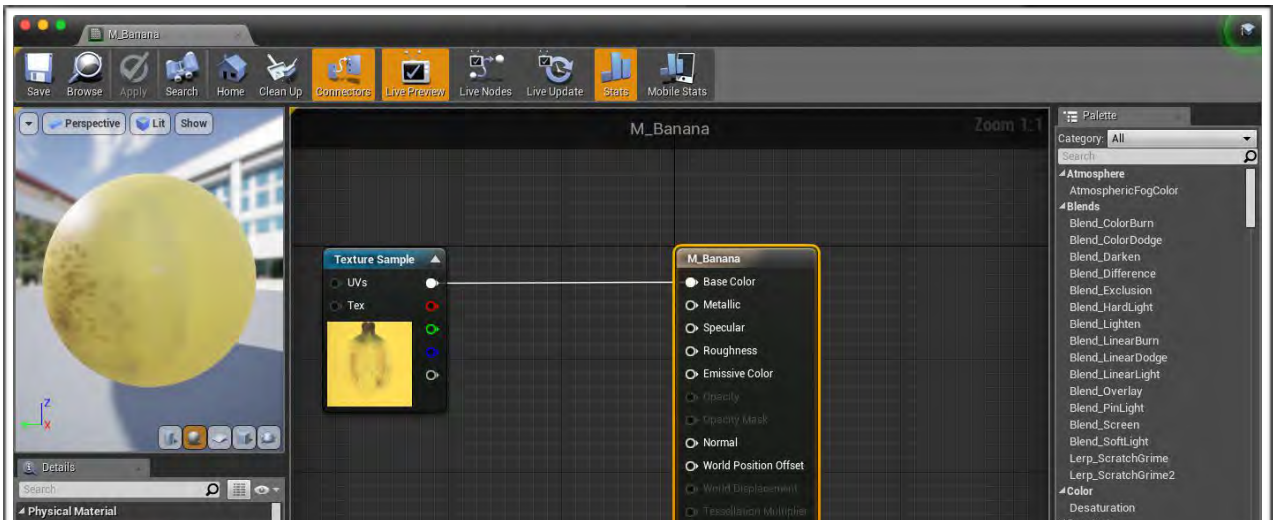
和之前类似，还是可能出现对话框，让你选择编辑器的版本。

在 Viewport 中可以看到一个小小的区域里面塞满了　。点击工具栏上的 Play 按钮，可以使用 W,A,S 和 D 键来控制红色的方块。我们可以移动方块靠近香蕉，从而收集它们。

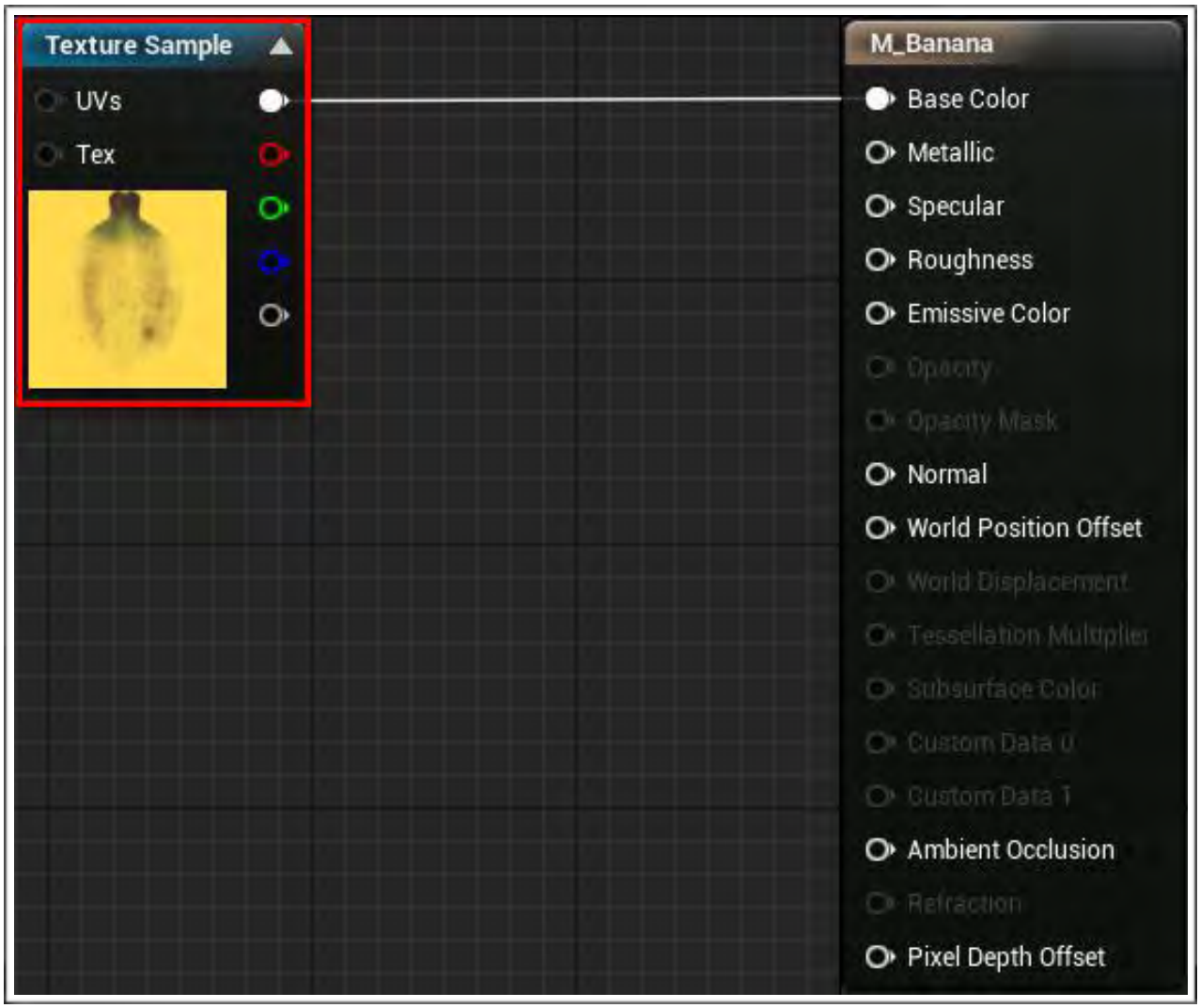
接下来，我们将首先更改香蕉的材质，以调整其亮度。在 *Content Browser* 中找到 *Materials* 文件夹，然后双击 *M_Banana*，在材质编辑器中将其打开。



此时你会看到类似下面的画面。



为了调整香蕉的亮度，我们需要对其纹理进行设置。



设置纹理

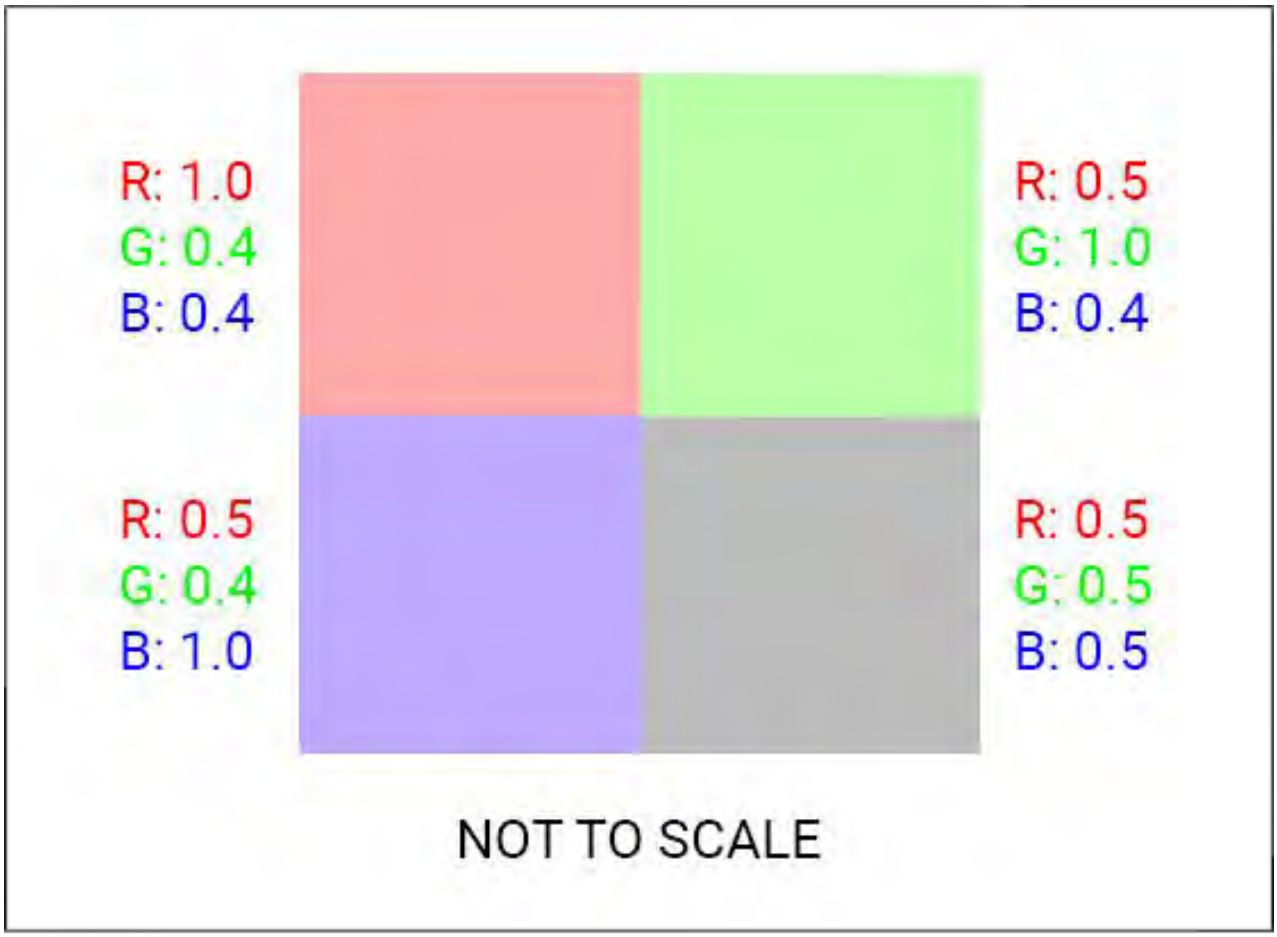
纹理就其本质来说不过是一张图片，而图片则是一堆像素的集合。在一张色彩图片中，像素的颜色由其 *R*（红色），*G*（绿色）和 *B*（蓝色）通道来共同决定。

下面是一个 2*2 像素的图片的示例，其中每个像素的 RGB 值都使用数字标出了。

注意，在虚幻 4 引擎中，RGB 通道的数值范围是 0.0 到 1.0。然而在大多数的其它应用中，RGB 通道值的范围是 0 到 255。这只不过是信息显示方式的不同，不要因此就认为虚幻 4 中的色彩范围更小。

要想设置纹理，就需要对纹理图片中的每个像素进行操作。而这种操作可能是类似往某个色彩通道添加数值这么简单。

在下面这个例子中，我们把每个通道的数值都限定在 0.4 到 1.0 之间，通过这样可以提升每个

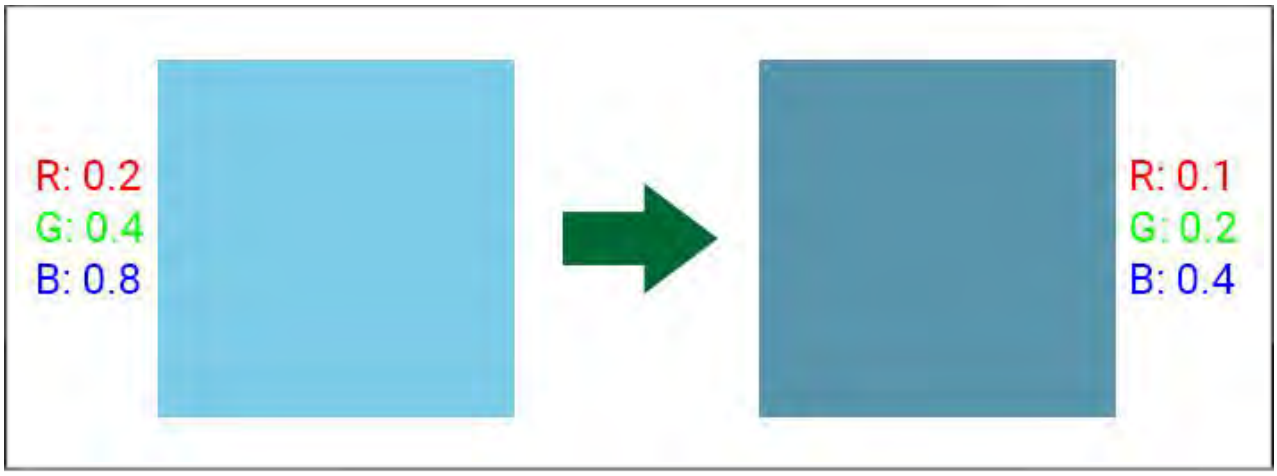


通道的最小数值，从而让每种色彩都显得更亮。

在材质编辑器中，我们将使用以下方式来实现上面的操作：

接下来，我们将使用 *Multiply* 节点来调整纹理的亮度。

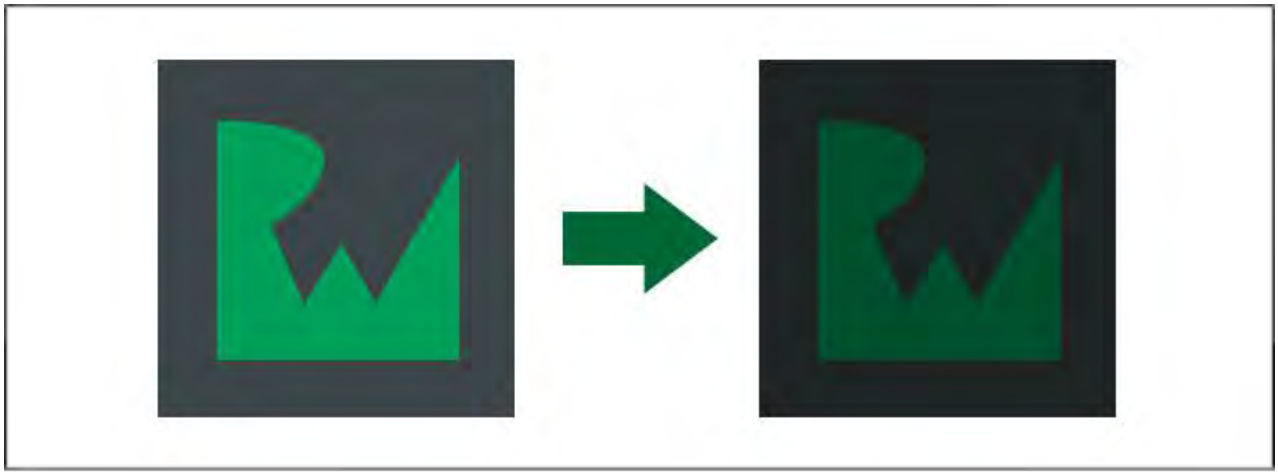
Multiply 节点



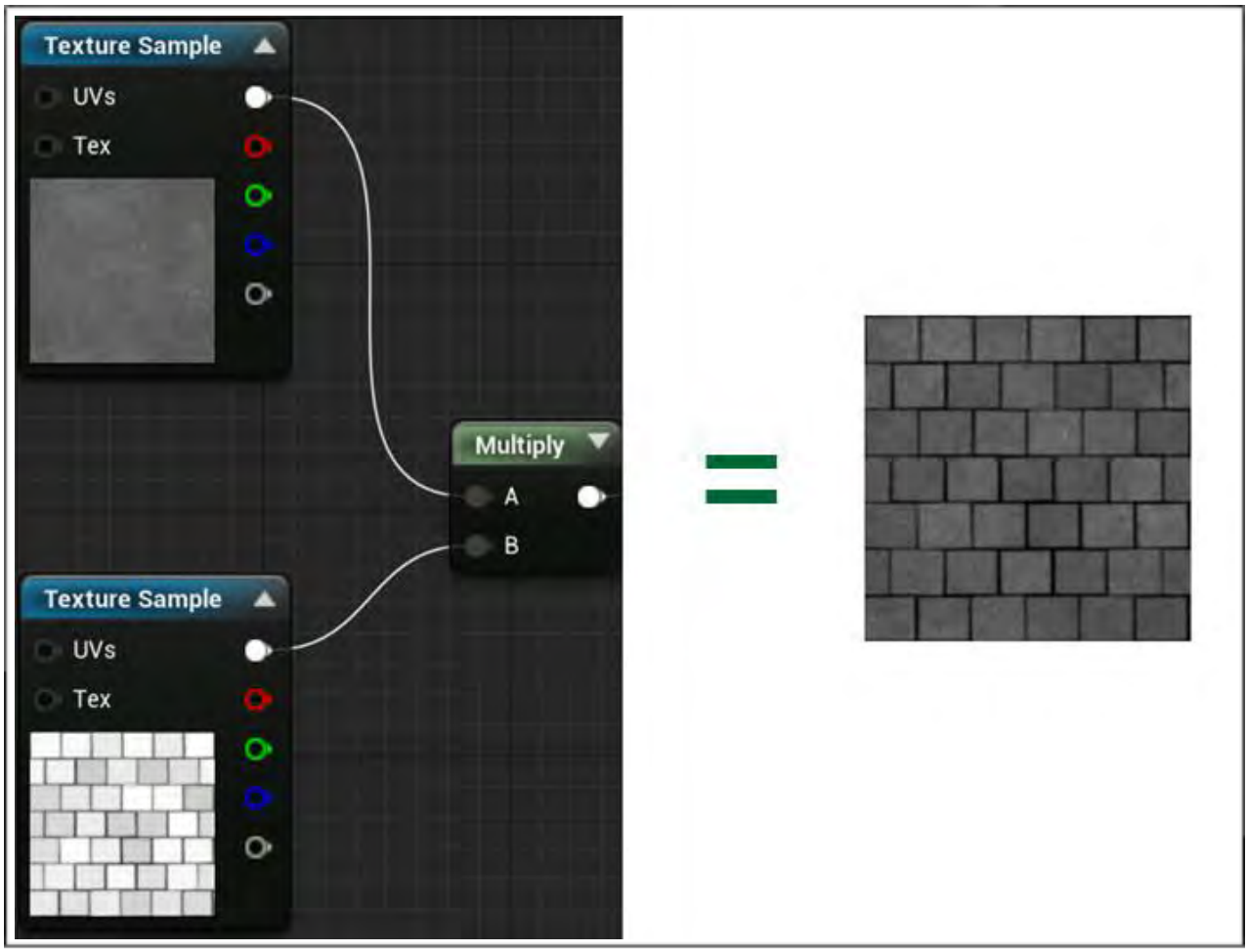
正如 *Multiply*（相乘）的字面意思，*Multiply* 节点的作用是将某个输入乘以另外一个输入。

使用乘法，我们可以在不影响像素的色彩和饱和度的前提下更改某个像素的亮度。在下面这个实力中，我们将每个色彩通道的数值乘以 *0.5*，从而将像素的亮度降为一半。

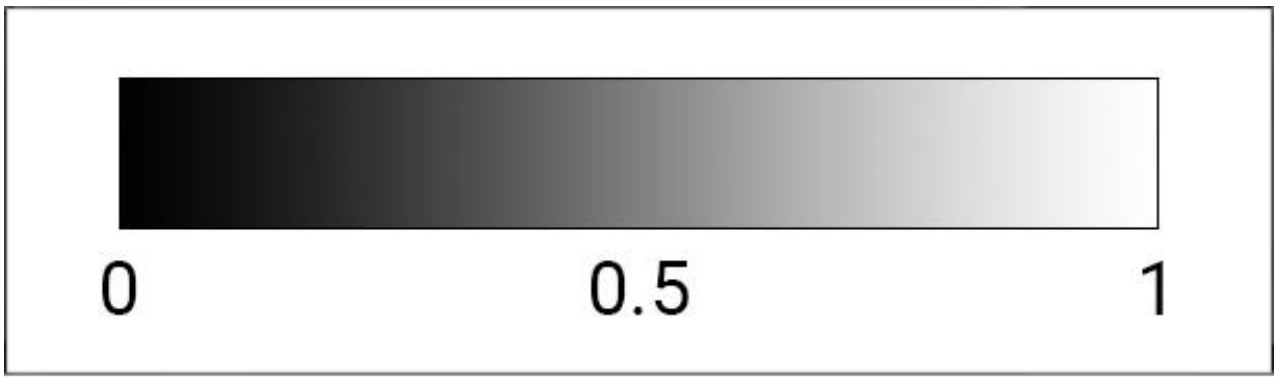
通过对每个像素进行这种运算，我们可以调整整个纹理图片的亮度。



此外，虽然本课内容中没有涉及到，但我们可以将 *Multiply* 节点和 *mask texture* 配合使用。通过使用 *mask* 遮罩，我们可以让纹理图片的某些特定区域变得更暗。下面这个例子展示了如何使用 *tiles* 纹理对石头纹理进行遮罩，从而让其变暗。



可以按照这里的遮罩起作用了，因为灰阶图代表着从 *0*（黑色）到 *1*（白色）的范围。

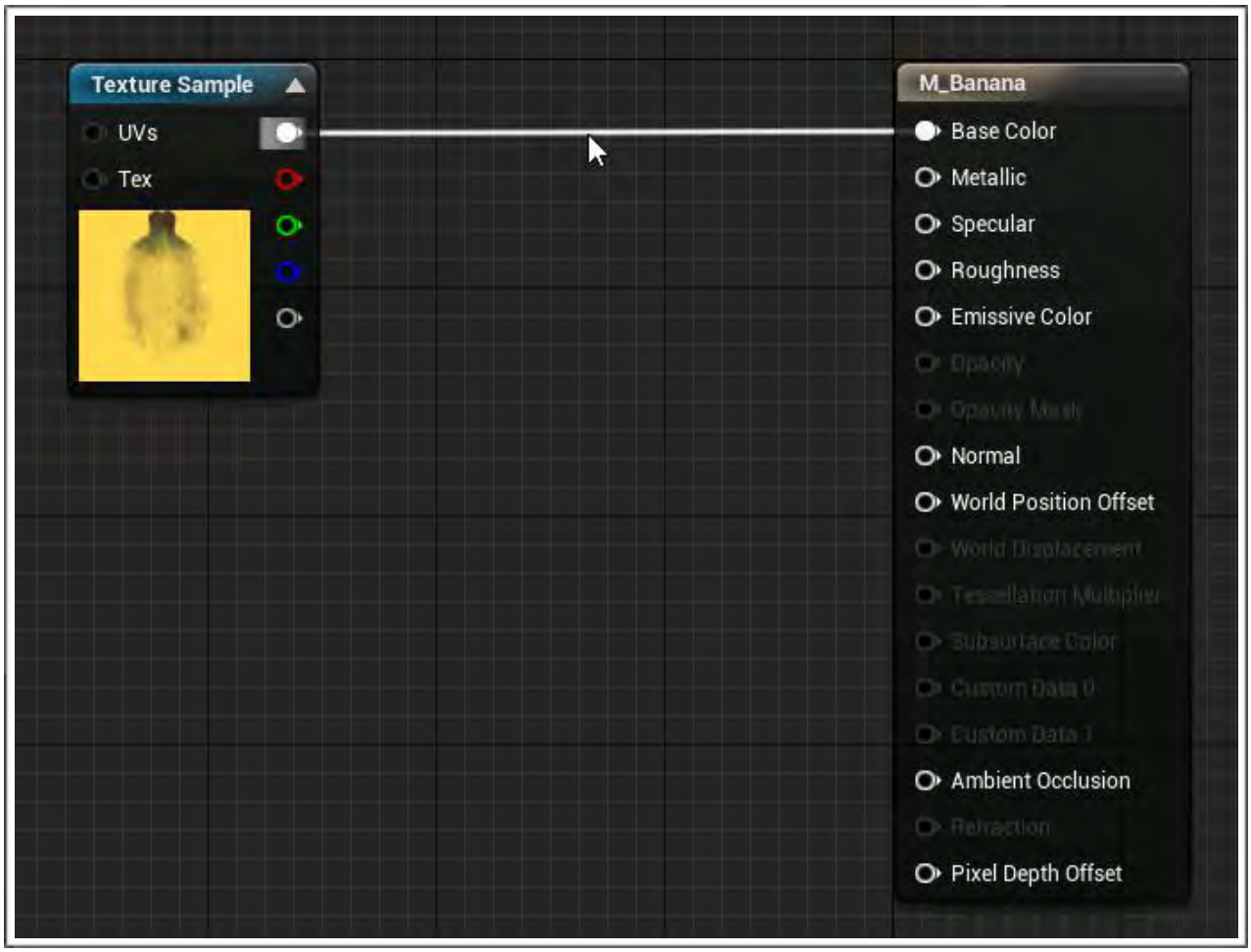


白色的区域亮度最大，因为对应的通道使用 *1* 进行数乘。灰色的区域显得更暗，因为对应的通道使用小于 *1* 的数值进行数乘。黑色的区域则没有亮度，因为对应的通道使用 *0* 进行数乘。

好了，关于理论的知识就到此结束。
在下一课的内容中，我们将实际使用 *Multiply* 节点。

在本课的内容中，我们将学习如何设置材质参数。
打开 Epic Game Launcher，从 Library 中找到并打开我们的项目。
从 Content Browser 中找到 M_Banana 材质球，并双击在材质编辑器中将其打开。

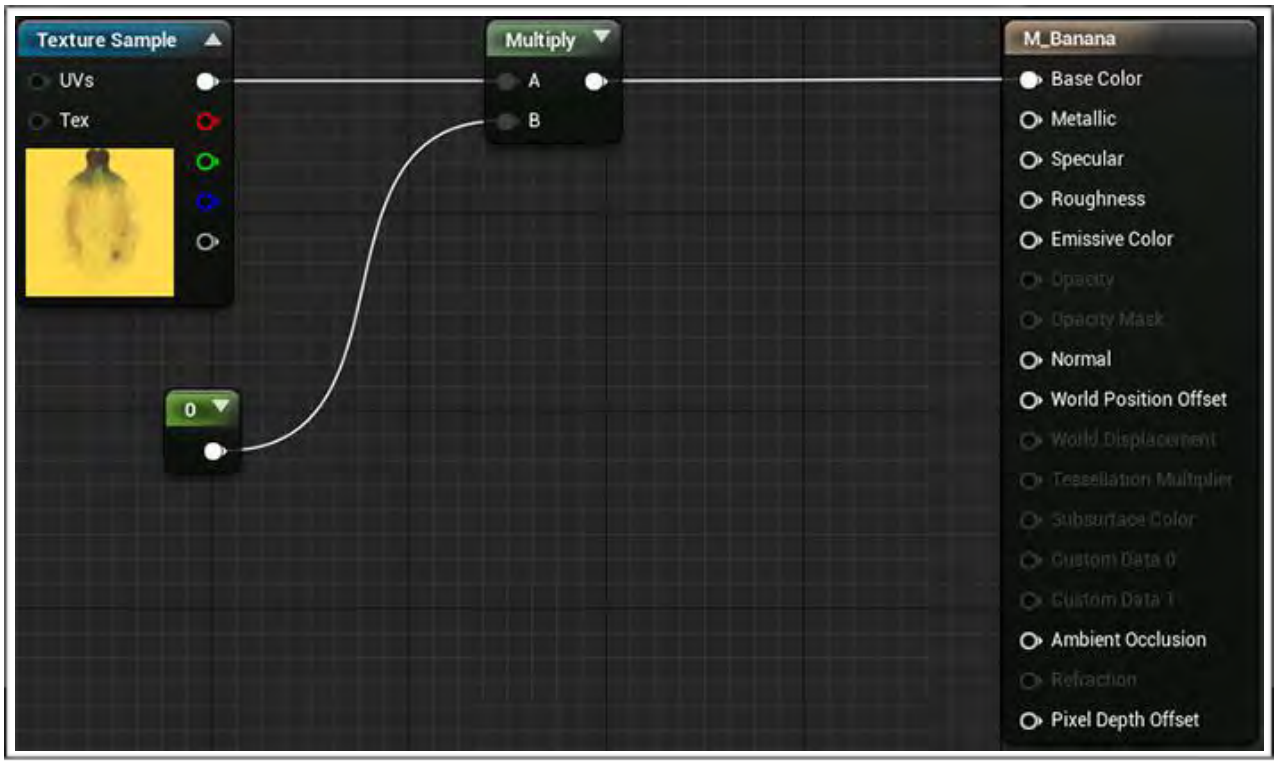
首先要做的就是断开 Texture Sample 到 M_Banana 之间的连接。
要实现这一点，需要右键单击 Texture Sample 的白色输出节点，或是 M_Banana 的 Base



Color 输入节点，然后选择 Break Links。当然，更快捷的方法是按住 Alt 键，然后左键单击连线。

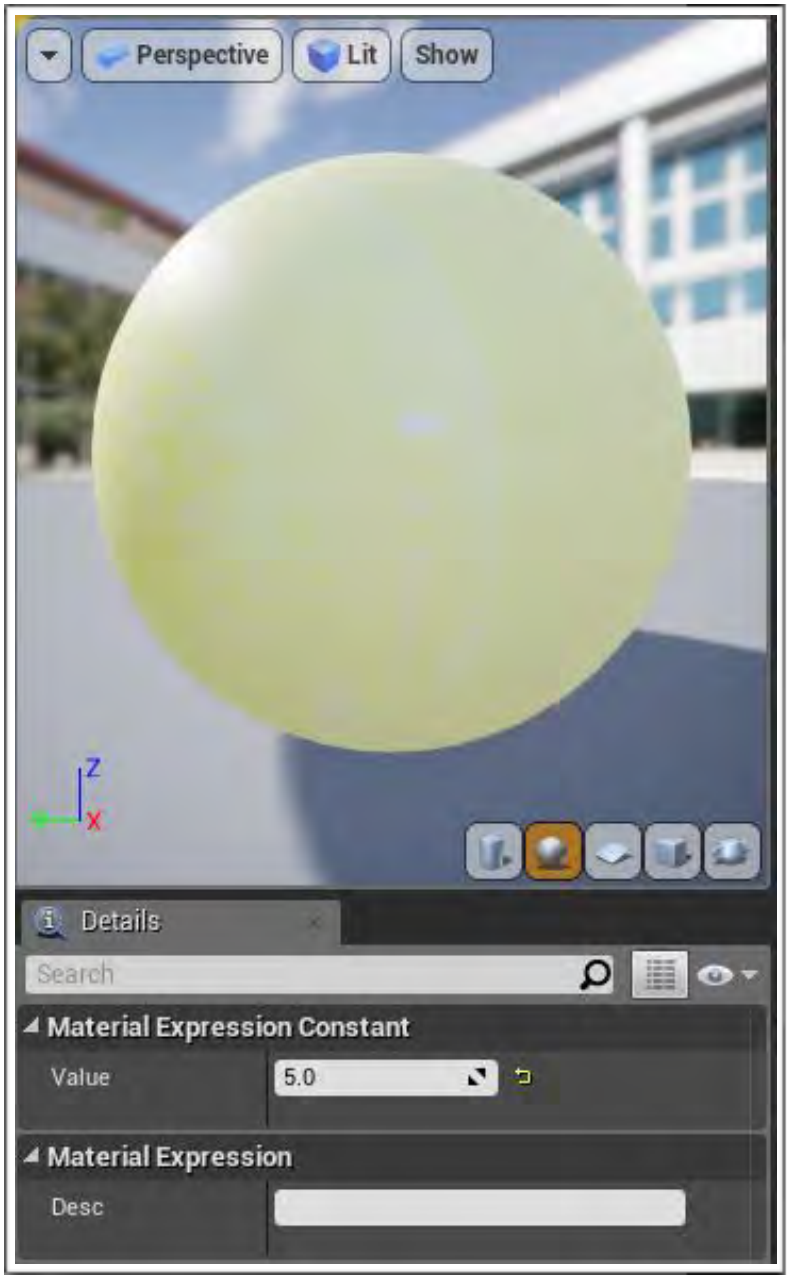
接下来需要创建一个 Multiply 节点和一个 Constant 节点。要创建这两个节点相对比较简单，只需要按下 M 键不放，然后左键单击 Event Graph 中的空白区域，就可以创建 Multiply 节点。类似的，按下数字 1 键不放，然后左键单击 Event Graph 中的空白区域，就可以创建 Constant 节点。

最后，按照下图的方式创建连线：



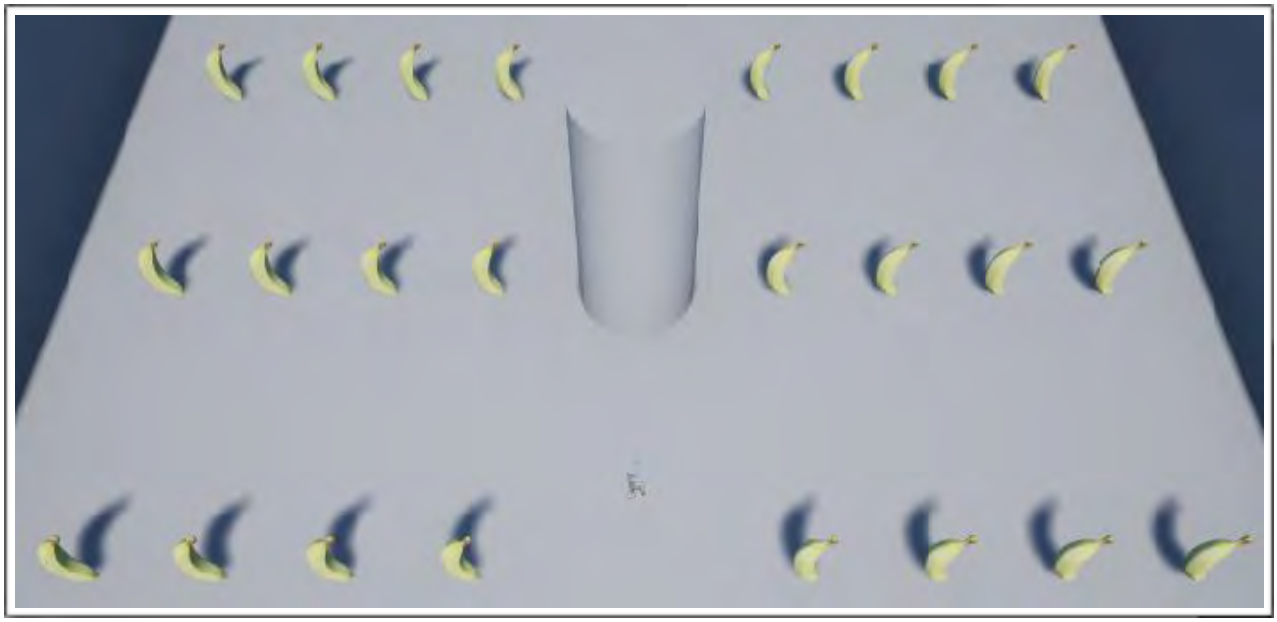
通过以上设置，可以让遍历纹理中的每个像素，使其每个通道值都乘以 *Constant* 节点的数值。而最后我们会将结果输出到 *Base Color*。

从材质编辑器的预览区可以看到，最后生成的纹理是纯黑色的，因为这里的数乘参数设置为 0，（任何数字乘以 0 的结果显然都是 0）。为此，我们需要选中 Constant 节点，然后在 Details 米板中将其 Value 设置为 5.



可以看到，预览区的纹理亮度一下子就提高了，如图所示。

点击材质编辑器工具栏上的 Apply 按钮，然后可以在主编辑器中看到对应的效果：



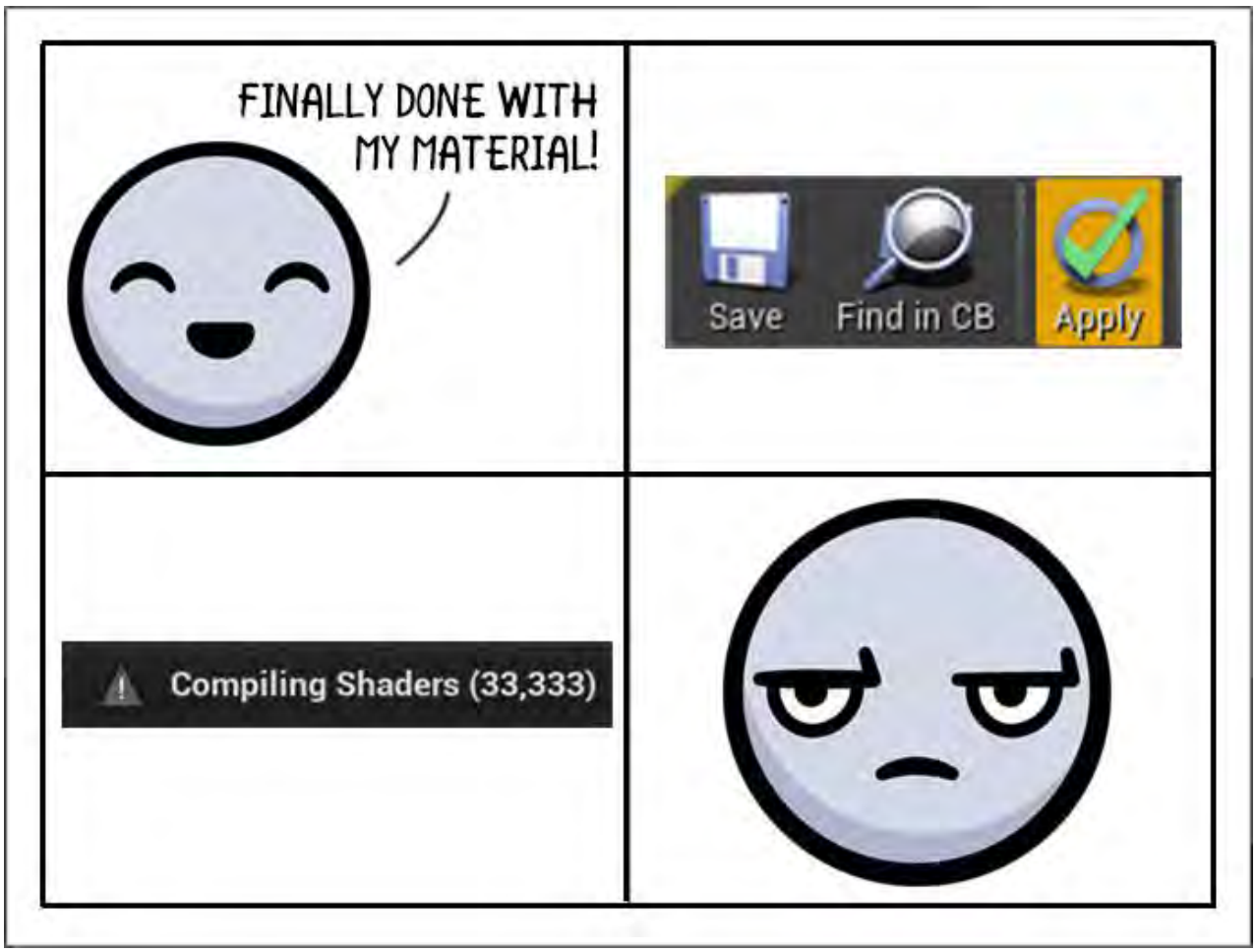
可以明显的看到所有的香蕉亮度都提高了。

为了让场景更加丰富多彩，我们可以考虑添加不同色彩的香蕉。尽管我们可以为每种色彩设置一个新的材质，不过更简单的方法是创建一个 *material instance*。

关于 *Material instance*（材质实例）

一个材质实例是材质的一份拷贝。因此，对基础材质的任何更改都会立即应用到 *material instance* 上面。

Material instance 的好处在于，我们可以在无需重新编译的情况下直接修改它们。



当我们点击材质的 *Apply* 按钮时，可以看到一个通知，告知对应的 *shader* 正在编译。
对于简单的材质来说，这个过程只要几秒钟就完成了。但是对于比较复杂的材质，编译时间可能会飞涨。

当我们遇到以下情况时建议使用 *material instance*:

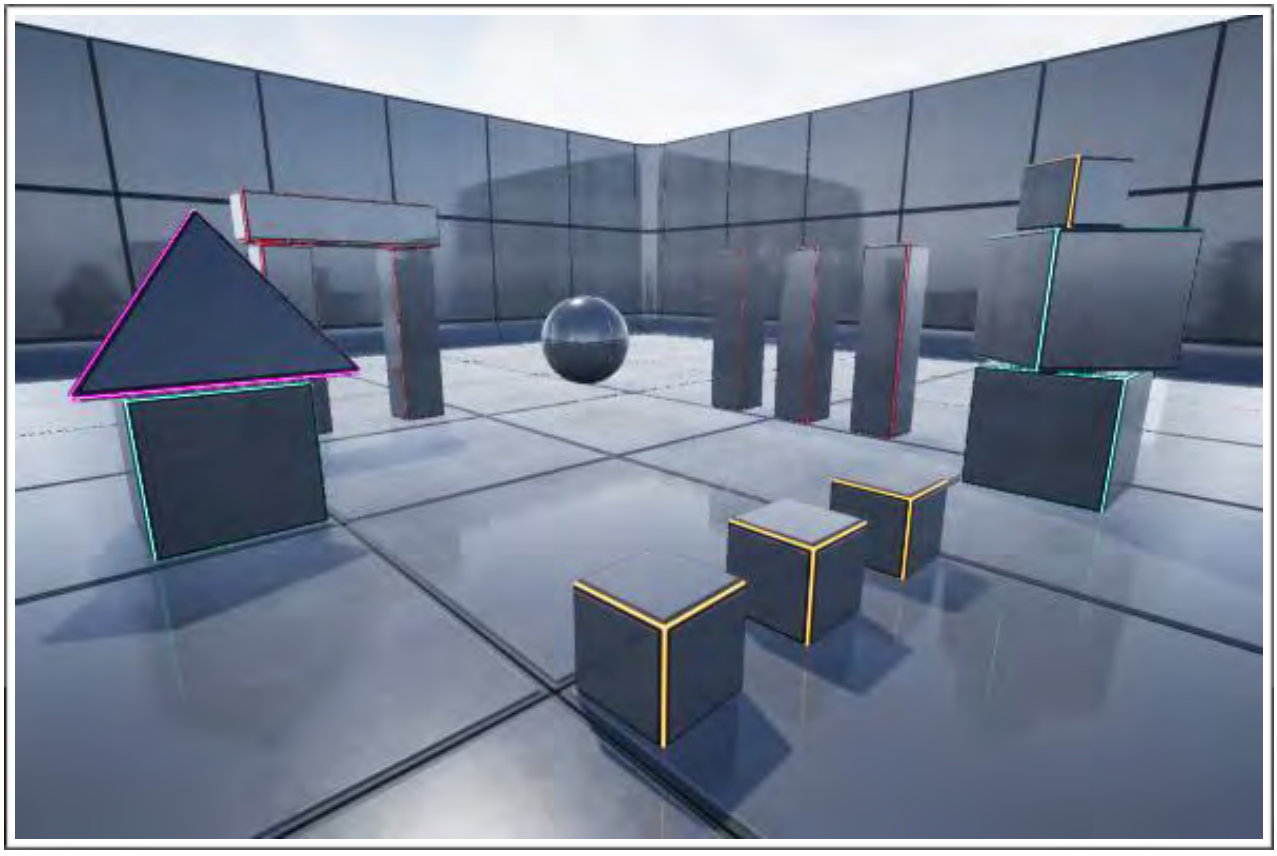
1.材质本身非常复杂，而且希望可以快速做出调整

2.希望创建基本材质的各种变化，比如更改色彩，亮度，甚至是纹理。

下图的场景中使用了 material instance 来创建不同色彩变化，所有的材质实例都共享一个基础材质。

需要注意的是，在创建材质实例之前，我们需要为基础材质创建 parameters。这些参数将显示在材质实例中，并允许开发者调整材质的各种属性。

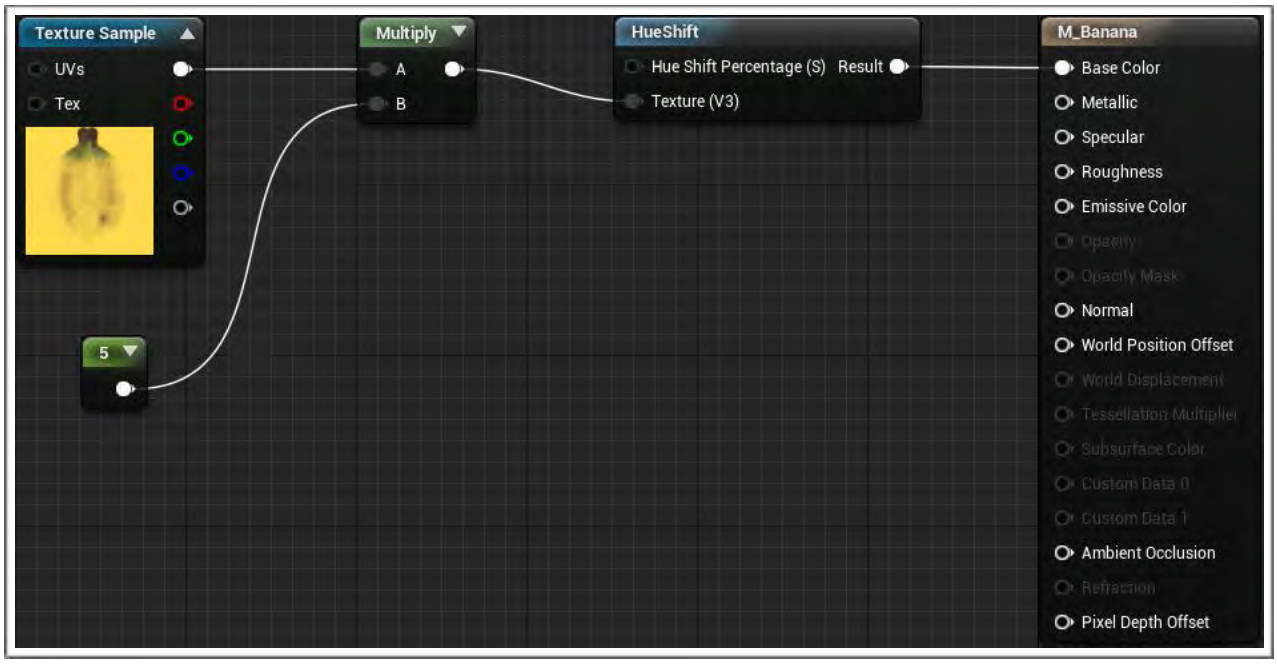
创建材质参数



现在返回之前打开的材质编辑器，确保我们仍然在 M_Banana 这个材质中。

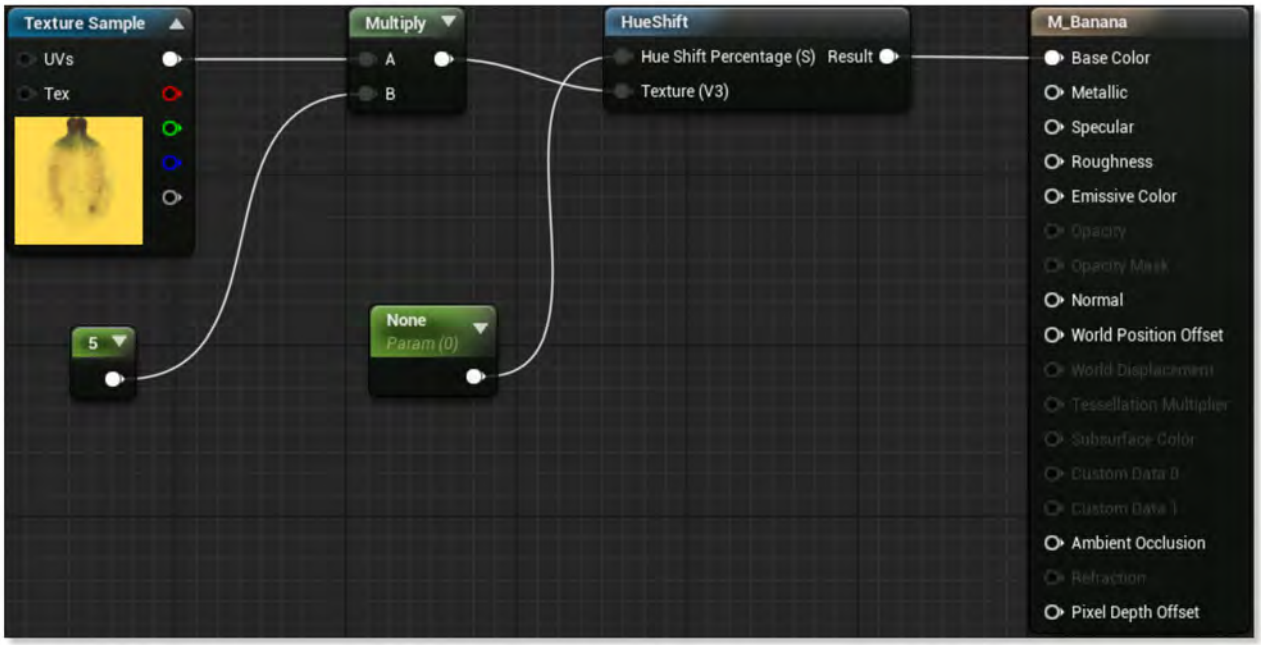
首先我们需要一个节点用于更改纹理的色调。为此，我们将使用 HueShift 节点。在 Event Graph 中添加一个 HueShift 节点，并使用下图的方式来创建连接。

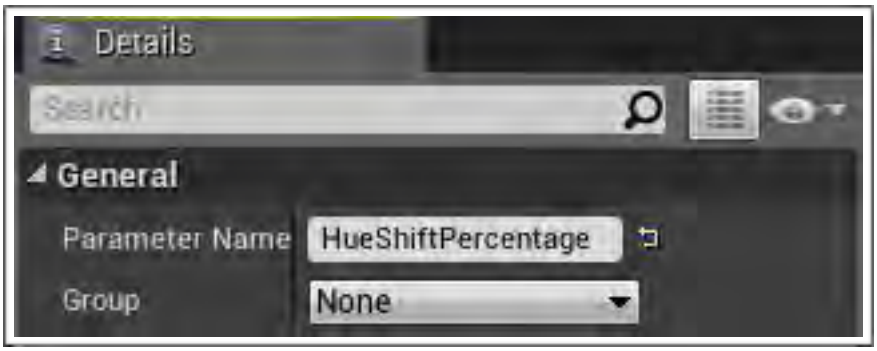
对于添加的具体步骤，这里简单描述下，如果你已经会了可以无视。
1.按住 Alt 键，然后左键单击连线以断开 Multiply 节点和 M_Banana 节点的关联。
2.右键点击 Event Graph 视图中的空白区域，然后搜索并选择 HueShift 节点。
3.按照上图的方式来创建连线。



接下来我们需要创建一个 Scalar Parameter 节点，该节点中将保存一个数值，并可以在材质实例中进行编辑。我们可以按下键盘上的 S 键，然后左键单击 Event Graph 视图的空白区域。接下来将其连线到 HueShift 节点的 Hue Shift Percentage(S）接口上。

为了便于项目管理，我们需要对材质参数进行命名。选中 Scalar Parameter 节点，然后在 Details 面板中将 Parameter Name 更改为 HueShiftPercentage。





此外，我们还可以将 Constant 节点转换为 Scalar Parameters。右键单击之前添加的 Constant 节点，右键单击，选中 Convert to Parameter。接下来，将其 parameter name

设置为 Brightness。

好了，现在我们已经实现了基础材质的参数化。点击材质编辑器上的 Apply 按钮，然后关闭材质编辑器。

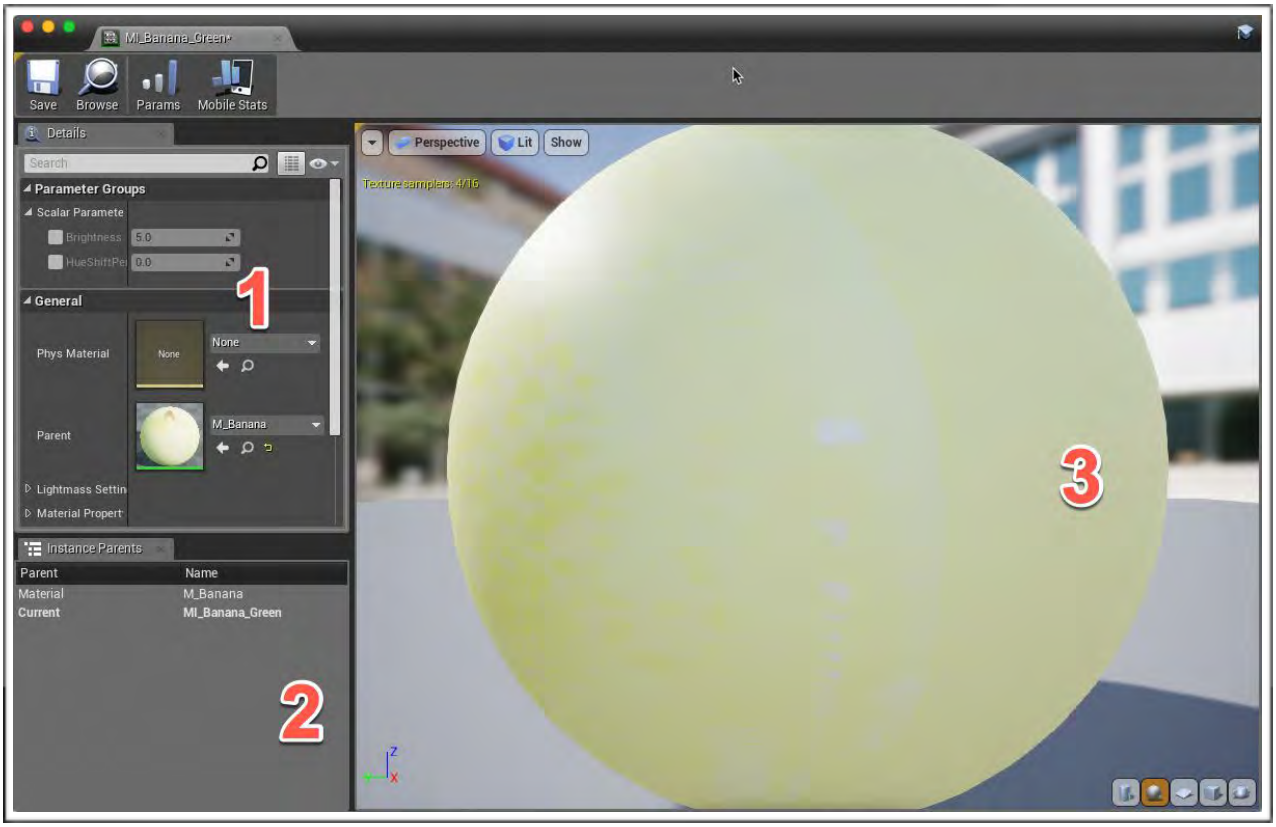
在下一课的内容中，我们将学习如何创建材质实例。

在本课的内容中，我们将学习如何创建材质实例。

创建材质实例

在虚幻 4 主编辑器的 *Content Browser* 中打开 *Materials* 文件夹。然后右键单击 *M_Banana* 蓝图文件，选择 *Create Material Instance*，将新的资源命名为 *MI_Banana_Green*。

双击该文件打开材质实例编辑器，如下图所示。



可以看到，材质实例编辑器由 3 个面板组成：

1.Details：

这里可以查看材质实例的参数和各种常见设置

2.Instance Parents:

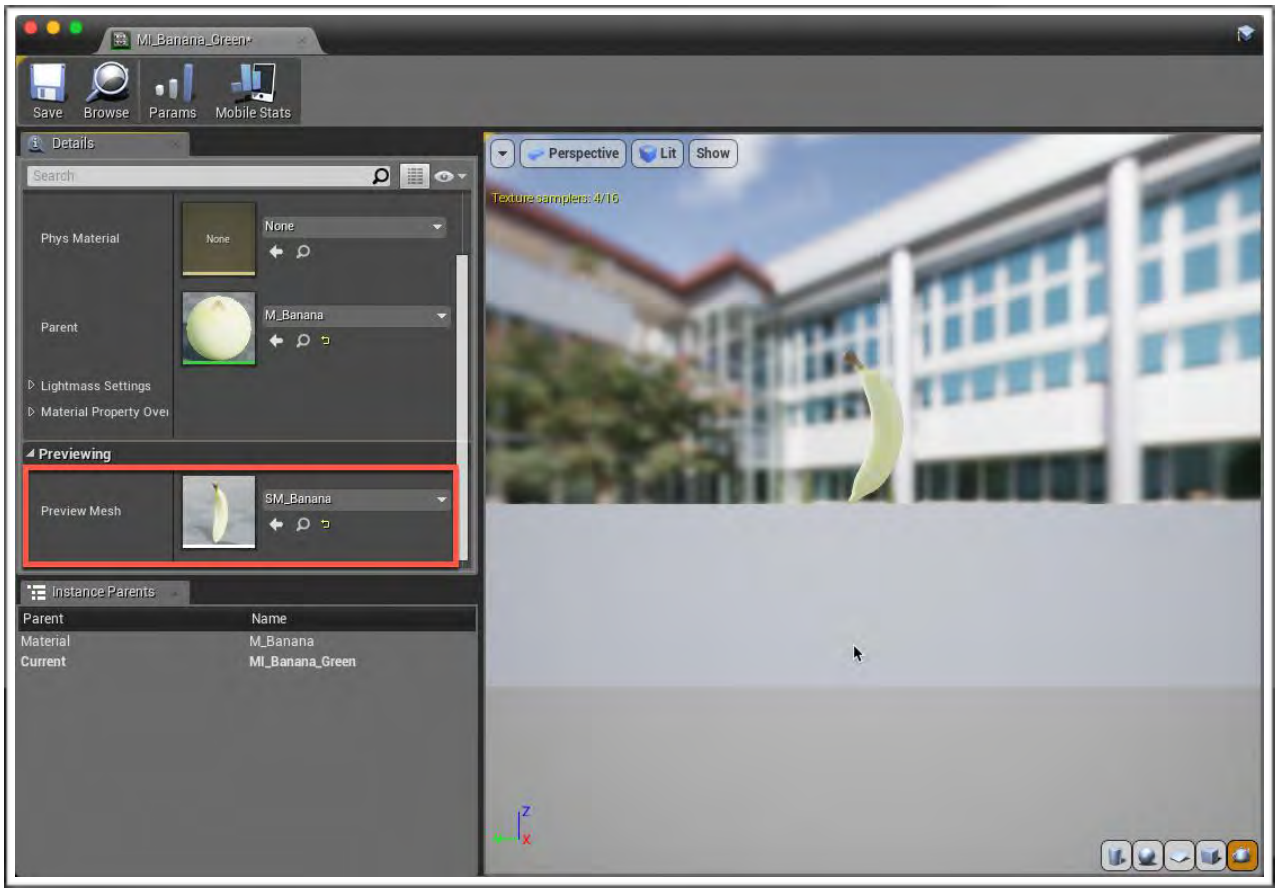
显示了当前材质实例的父材质。在这里当然就是 M_Banana

3.Viewport：

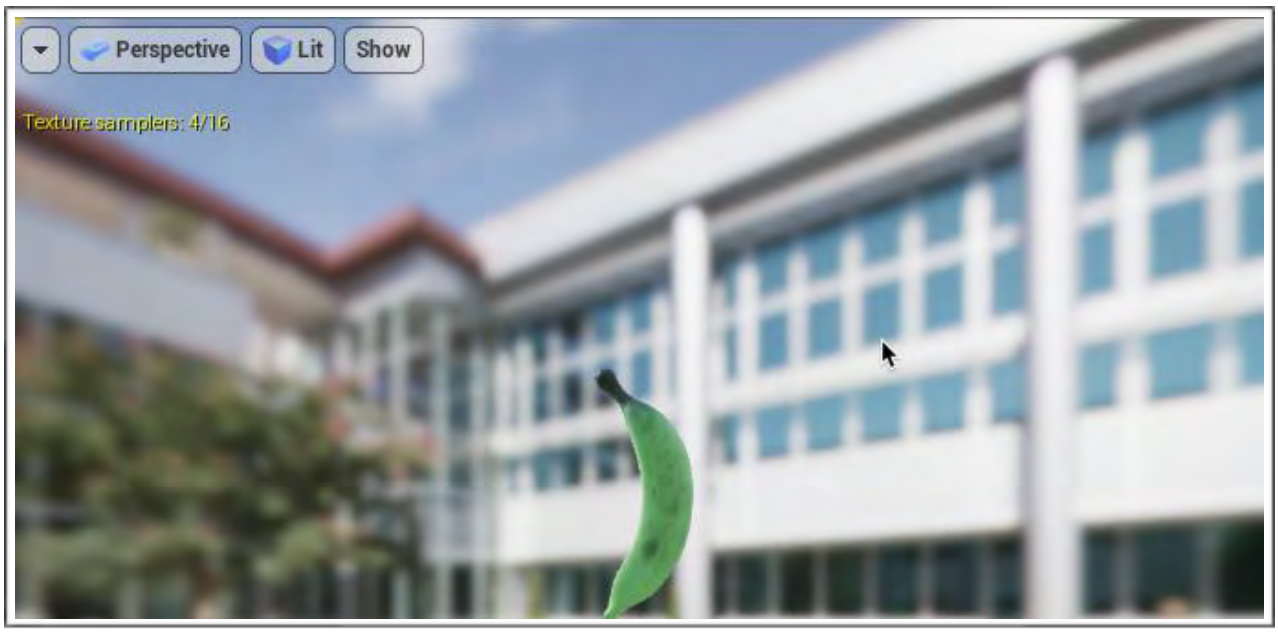
这里包含了一个预览的模型，用来显示我们的材质实例。在这个视口中，可以按住鼠标左键，然后移动鼠标来旋转摄像机。也可以使用鼠标滚轮来缩放视图。

如果我们想查看香蕉模型，那么可以在 Details 面板中找到 Previewing 部分，从 Preview Mesh 右侧的下拉列表中选择 SM_Banana。此时可以在预览区看到香蕉，而非球体。

接下来我们将编辑材质实例的参数，使得香蕉的色彩变为绿色。为了让参数可以编辑，需要左键单击每个参数旁边的选框。



将亮度设置为 0.5,HueShiftPercentage 设置为 0.2，此时可以在预览区看到类似下面的效果。

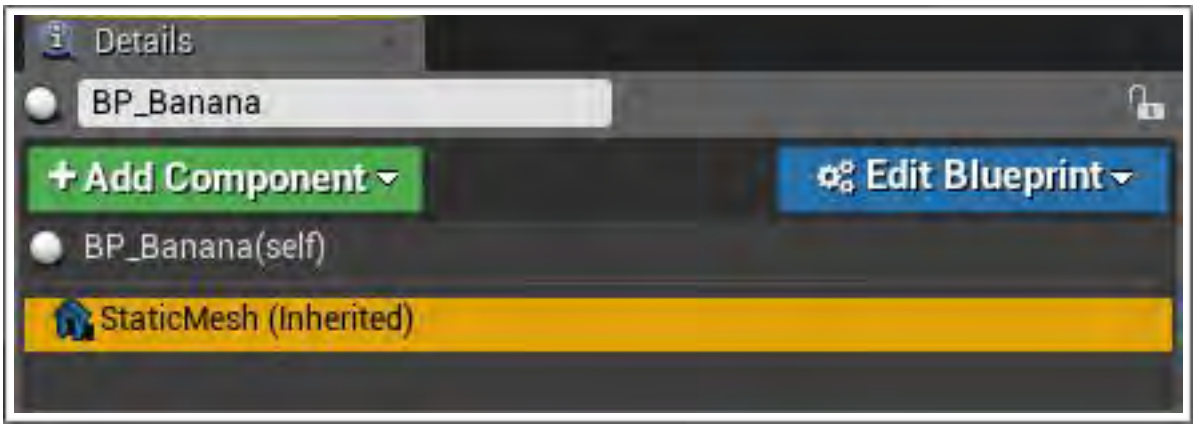


现在我们已经创建了自己的材质实例，接下来就可以把它应用到某些香蕉上了。关闭材质实例编辑器，然后查看主编辑器中的 *Viewport* 视口。

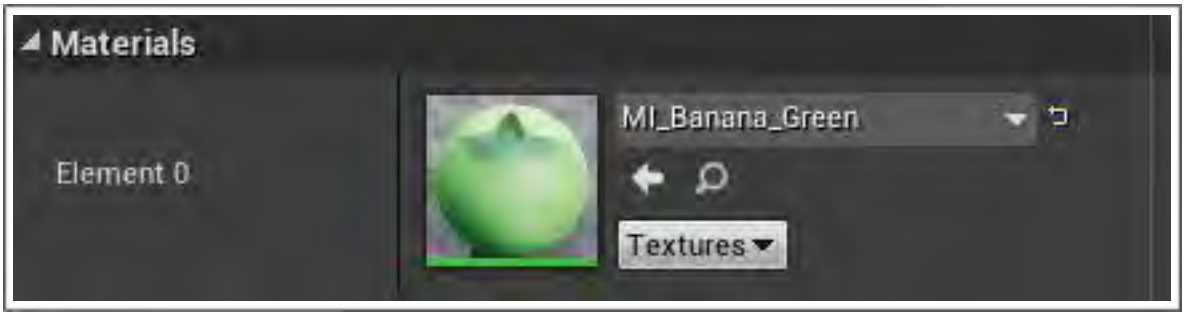
应用材质实例

需要说明的是，场景中的角色是可以被单独编辑的。这就意味着我们可以为每个香蕉分别设置材质，而不会影响到其它的香蕉。我们可以使用这种方式将某些香蕉的色彩设置为绿色。

在场景中选中任何一个香蕉，然后切换到 *Details* 面板，在 *component list* 中，选中 *StaticMesh* 组件。

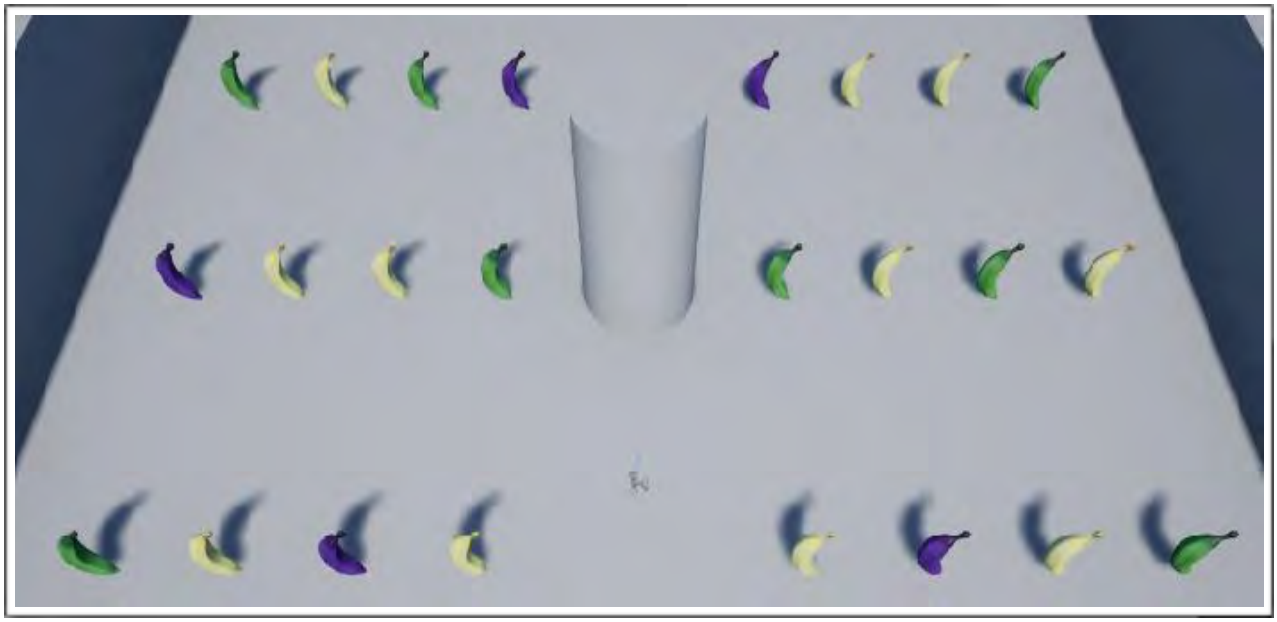


此时在 *Details* 面板中会显示 *StaticMesh* 组件的属性，将其中的材质设置为 *MI_Banana_Green*。



重复这个过程，让场景中的　有不同的色彩。当然，我们还可以使用类似的方式创建另外的材质实例，从而让某些香蕉变成紫色或者其它颜色。

如果这样操作，那么最后你可能会看到类似下面的场景~



好了，本课的内容到此结束。

在下一课的内容中，我们将学习如何让材质可以在游戏中动态变化。

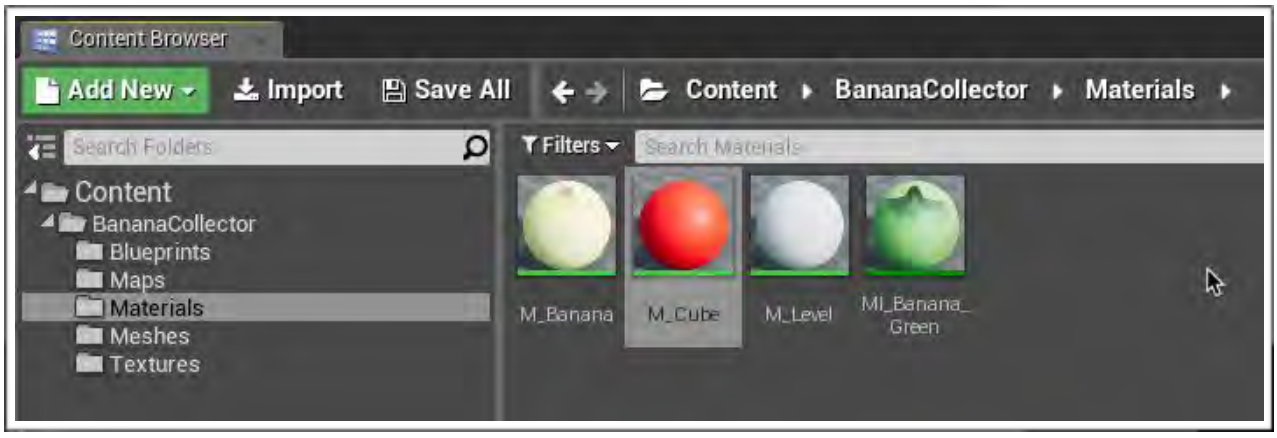
在本课的内容中，我们将学习如何创建可以动态变化的材质。

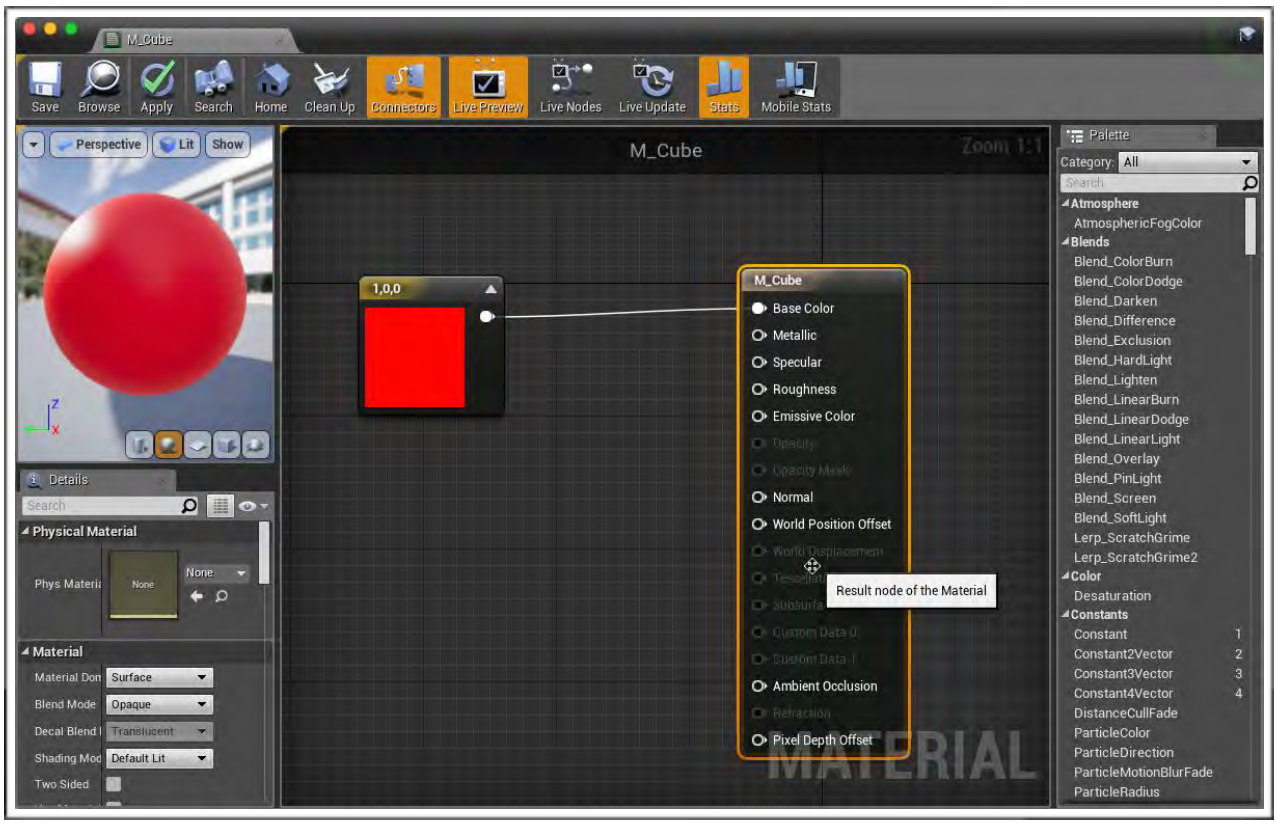
需要提醒大家的是，材质不仅仅可以作为花瓶一样的装饰品，我们也可以使用材质来辅助游戏设计。

接下来，我们将学习如何创建动态变化的材质，从而实现当玩家收集香蕉物品的时候，让方块从白色变成红色。

在创建材质实例之前，我们首先需要设置方块的材质。

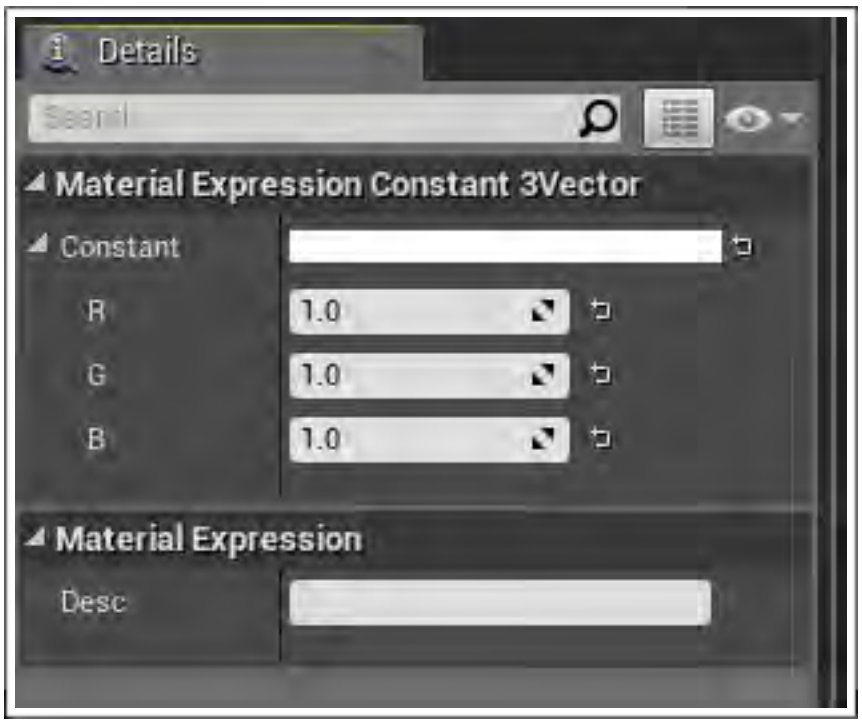
在虚幻 4 中打开我们的项目，然后在 Content Browser 的 Material 文件夹中找到 M_Cube，并双击将其打开。





首先我们需要创建材质的颜色。在材质编辑器中，可以看到有一个 *Constant3Vector* 节点连接到 *Base Color* 节点上。使用该节点来选择颜色是非常合适的，因为我中包含了 *RGB* 三个通道。

现在我们已经有了红色，接下来需要创建白色。添加另一个 *Constant3Vector* 节点，可以按下数字 *3* 键，然后左键单击 *Event Graph* 的空白区即可快速创建。



选中该节点，在 *Details* 面板中将 *Constant* 下面的 *R,G,B* 都设置为 *1.0*。

为了让色彩从白色变成红色，我们需要创建两者之间的平滑过渡。而实现这一点的最简单方式就是使用 *linear interpolation*（线性插值）。

这个词听起来有点唬人，其实也没那么复杂。线性插值的作用就是找到 A 和 B 两个数值之间的一个数值。在下图的例子中我们使用线性插值找到和 100，200 等距的值，也就是 150。
当然，如果你对线性插值的数学概念很感兴趣，可以参考百度百科，或其它的更详细解释：

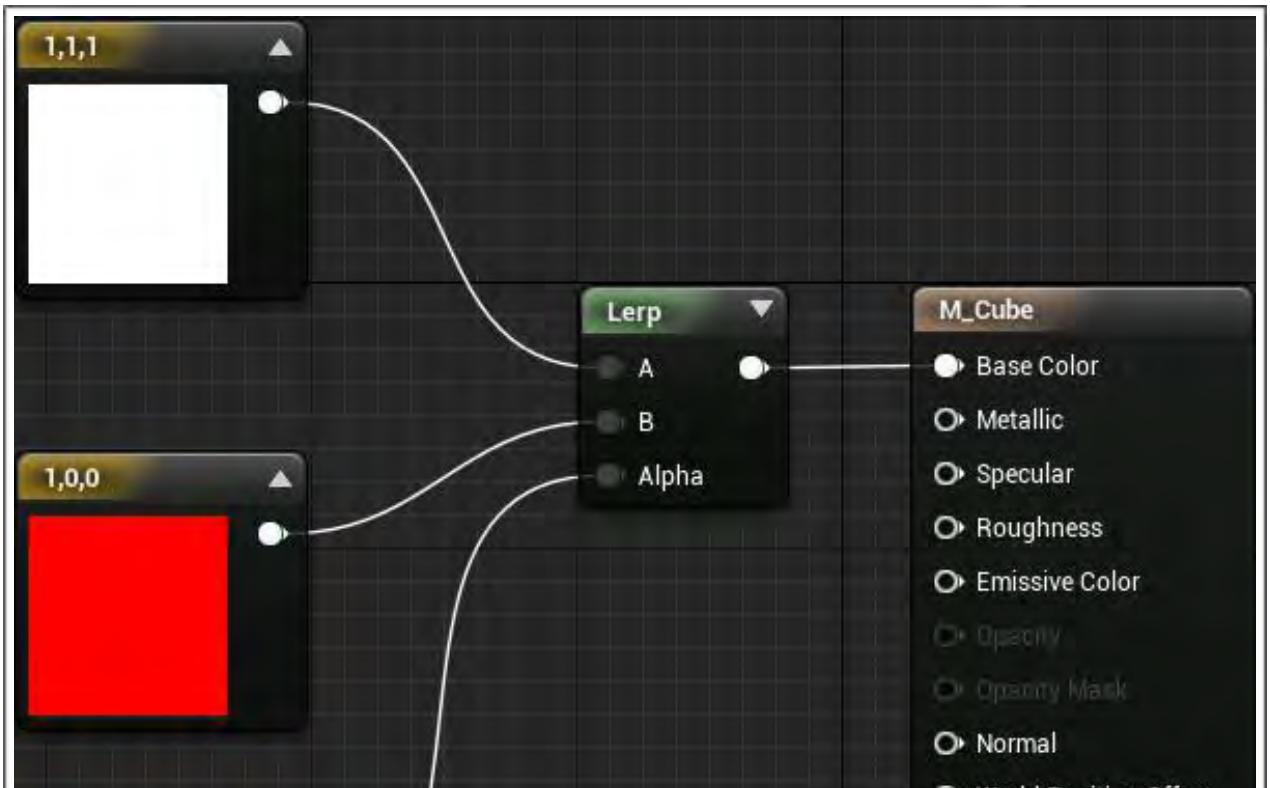
https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E6%8F%92%E5%80%BC

如果我们能够控制 alpha 值，那么线性插值就会更加强大。我们可以认为 alpha 是 A 和 B 之间的百分之，当 alpha 为 0 时，我们得到的数值是 A，而 alpha 为 1 时，数值是 B。

在本课的内容中，我们将使用所收集到的香蕉梳理来控制 alpha 值。

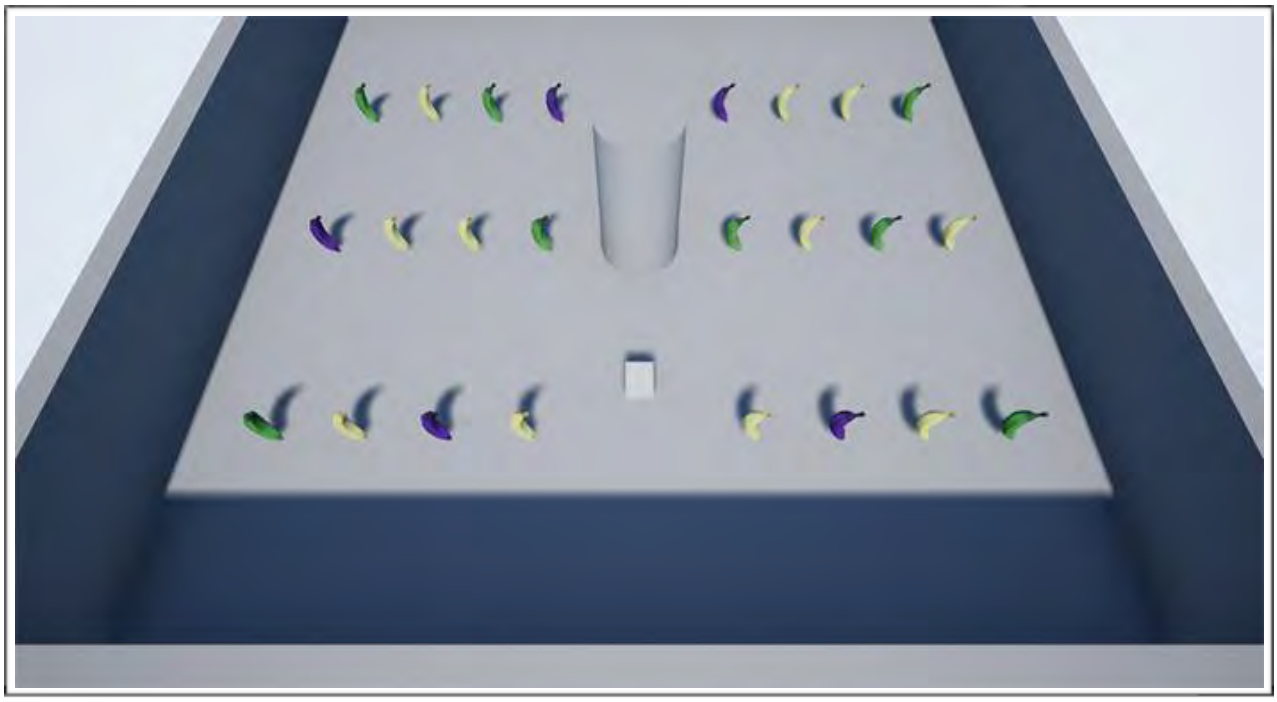
使用 LinearInterpolate 节点

首先我们需要添加一个 LinearInterpolate 节点。按下键盘上的 L 键，然后在 Event Graph 的空白区左键单击。



接下来还需要创建一个 Scalar Parameter 节点，然后将其命名为 ColorAlpha。接着我们按照以下方式来连接节点。

注意：默认情况下 LinearInterpolate 节点输出的是 A，这是因为 alpha 的初始值是 0，当



alpha 值变成 1 的时候，节点输出的就是 B。

好了，现在材质已经准备好了。接下来我们还有一些准备工作，但是先让我们检查一下一切是否正常。点击材质编辑器上的 Apply 按钮，然后关闭材质编辑器。如果我们点击工具栏上的 Play 按钮，就可以看到方块的色彩是白色的了。

为了让方块可以改变颜色，我们需要动态编辑 ColorAlpha 参数。但是现在有一个问题，我们没法在游戏运行的时候动态编辑材质实例的参数。
解决的方法是使用动态材质实例。

关于 Dynamic Material Instances（动态材质实例）

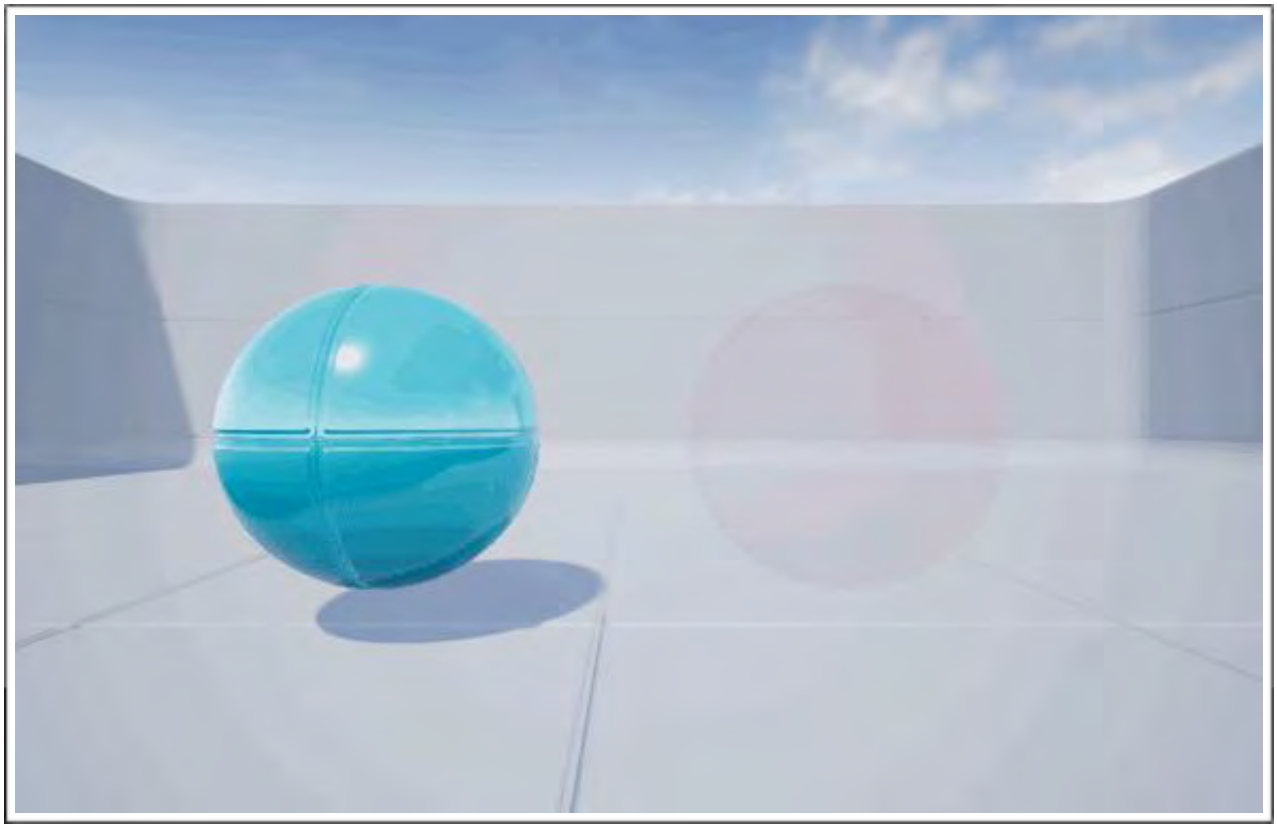
和常规的实例对象不同，我们可以在游戏运行的过程中对动态材质实例进行编辑。而实现这一点既可以使用蓝图系统，也可以直接使用 C++代码。

我们可以对动态材质实例做很多操作，比如更改物体的透明度，让其变得透明，或是增加物体表面的 *specularity*（镜面发射），让其显得更潮湿。

关于动态材质实例，更有意思的一点是我们可以分别对其进行编辑。

创建动态材质实例

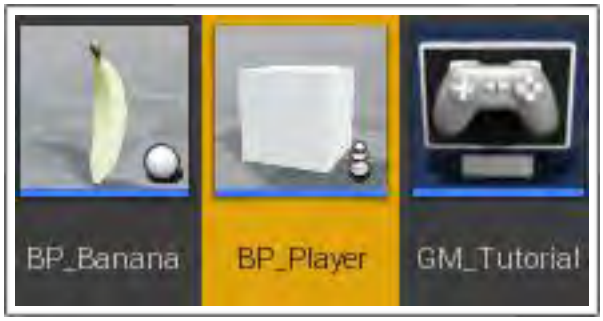
我们只能在游戏运行的过程中创建动态材质实例，可以用蓝图系统或 c++来实现这一点。



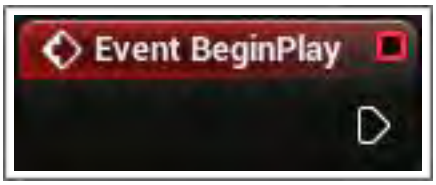
在虚幻 4 主编辑器的 Content Browser 中，找到 Blueprints 文件夹，然后双击打开 BP_Player 蓝图文件。

首先我们要创建一个新的动态材质实例，然后将其应用到方块的模型上。为此，我们可以在虚幻刚刚生成角色时进行这个操作，也就是要使用 Event BeginPlay 节点。

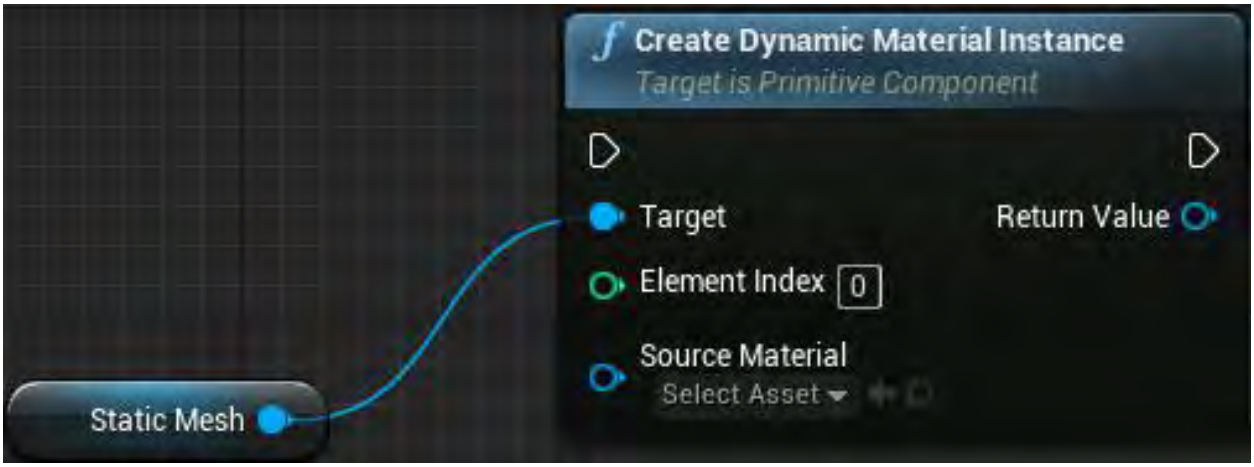
在 Event Graph 视图中找到或添加一个新的 Event BeginPlay 节点。



接下来添加一个 Create Dynamic Material Instance(StaticMesh)节点，该节点将会同时创建并应用一个新的动态材质实例到方块模型上。



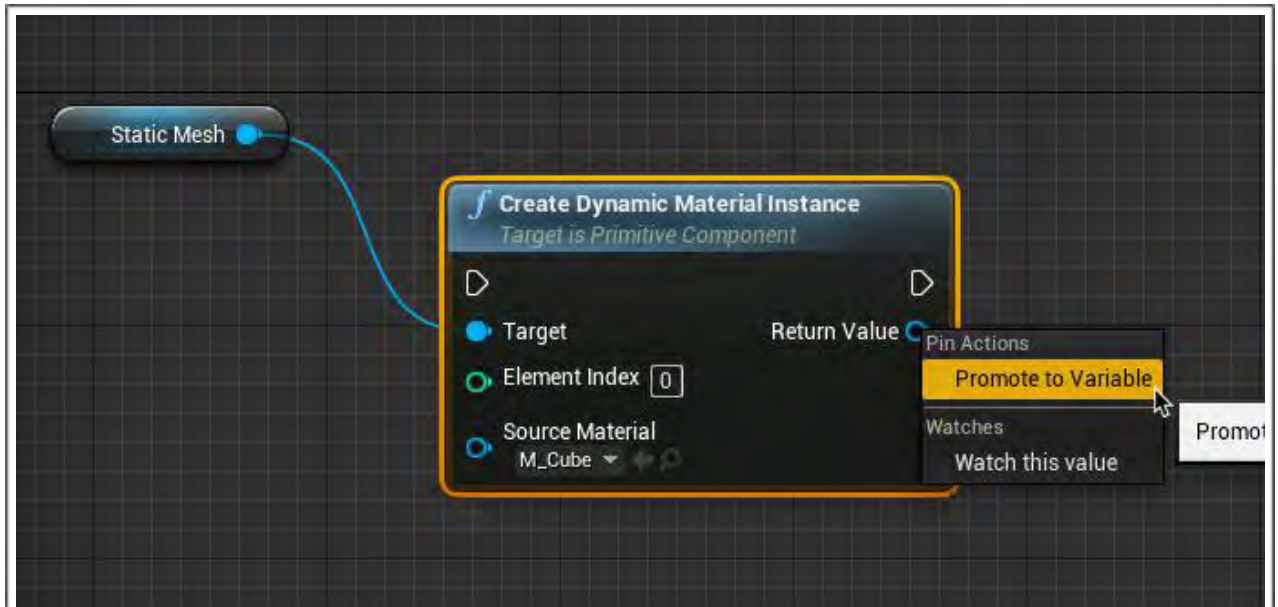
接着我们需要设置方块所使用的材质。在该节点的 *Source Material* 接口从下拉列表中选择



M_Cube。

为了方便稍后引用材质，我们需要将其保存到一个变量中。为此，可以右键单击 *Create Dynamic Material Instance* 节点的 *Return Value* 蓝色接口，然后选择 *Promote to Variable*。

此时在 *My Blueprint* 选项处会看到出现了一个新的变量，将其更名为 *CubeMaterial*。也可以



使用快捷键 *F2* 来完成该操作。

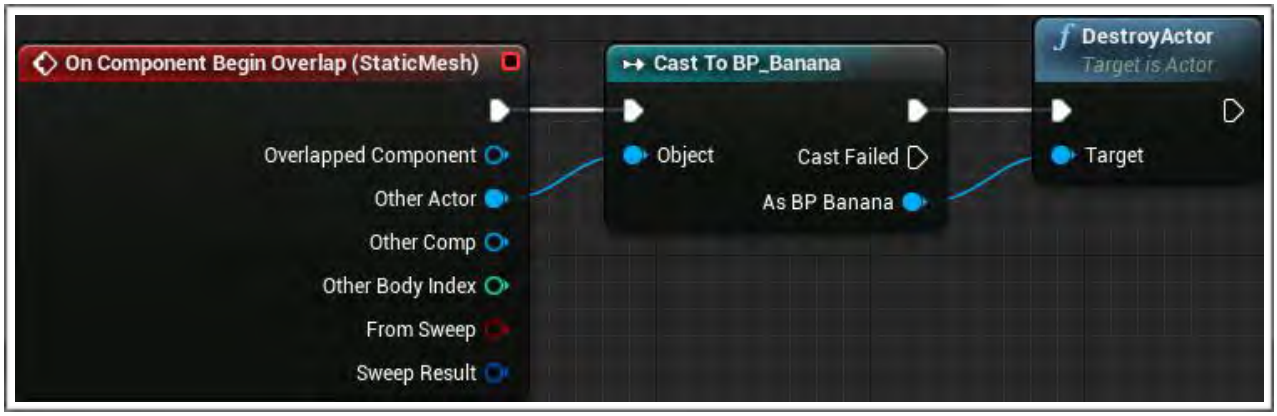
最后让我们从 *Event BeginPlay* 节点到 *Create Dynamic Material Instance* 节点之间的白色接口之间添加连接。

注意：一旦虚幻引擎生成了 BP_Player，就会创建一个新的动态材质实例对象，然后将其应用到 StaicMesh 组件上。然后它会将所创建的材质保存在名为 CubeMaterial 的变量中。

好了，本课的内容到此结束。
在下一课的内容中，我们将创建一个计数器，从而记录所收集的　数量。

在本课的内容中，我们将创建一个计数器，从而记录所收集的　数量。

在虚幻 4 引擎中打开项目，然后在 Content Browser 中找到 Blueprints 文件夹，双击打开 BP_Player 蓝图文件。
在 Event Graph 视图中向下移动，可以看到一个 On Component Begin Overlap（StaticMesh)节点，在这里我们可以更新香蕉计数器和材质。



当代表玩家角色的方块重叠到其它角色上时，就会执行 On Component Begin Overlap 节点。而白色执行线的下一站是 Cast to BP_Banana 节点，在此将检查所重叠的角色是否是香蕉。如果该角色时香蕉，那么就会调用下一个节点 DestroyActor，将其从游戏中销毁。

为了创建香蕉计数器，首先我们需要创建一个变量，来保存所收集的　数量。
之后就简单了，只需要在每次方块重叠到香蕉上的时候将该变量的数值加 1 就好。

在蓝图编辑器的 My Blueprint 面板中年级 Variables 旁边的加号，创建一个新的 Float 类型变量，并将其命名为 BananaCounter。

从 My Blueprint 中将 BananaCounter 拖动到 Event Graph 中，选择 Get BananaCounter。

然后在项目中添加一个 IncrementFloat 节点，并将 BananaCounter 连线过去。

紧接着把 DestroyActor 的白色执行线连接到 IncrementFloat 节点。

现在只要玩家捡起一根 ，BananaCounter 变量就会自动加 1.

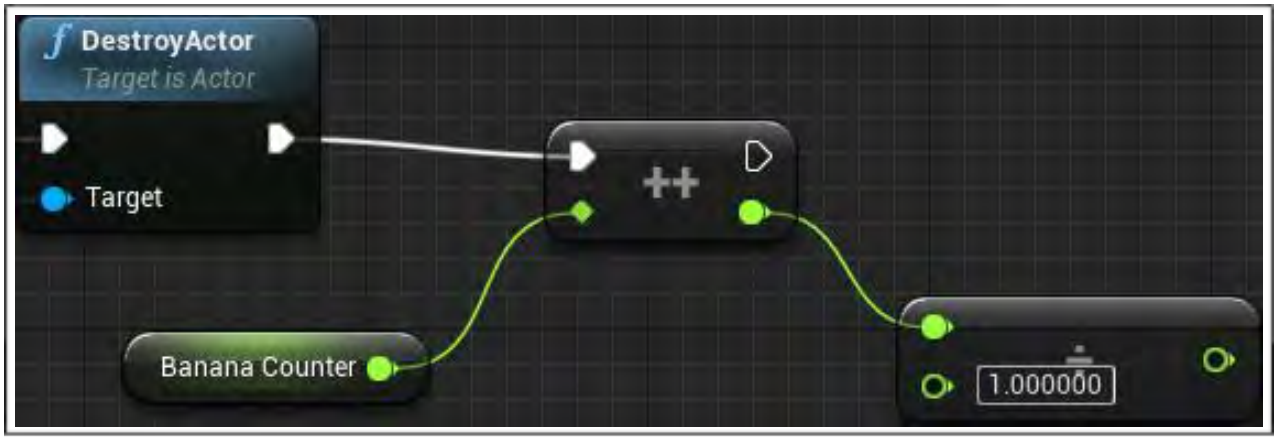
当然，现在如果直接使用 BananaCounter 的话，可能会得到不可思议的结果。

这是因为 LinearInterpolation 节点需要一个从 0 到 1 的数值。因此，我们需要使用标准化将计时器的数值转换成 0 到 1 之间。

标准化的方式很简单，只需要将 BananaCounter 除以一个最大值即可。而该数值的大小可以设置为让玩家角色完全变红所需要收集的香蕉。

在 Event Graph 中添加一个 float/float 节点，然后将其顶部的接口连线到 IncrementFloat 节点的另一个端口。

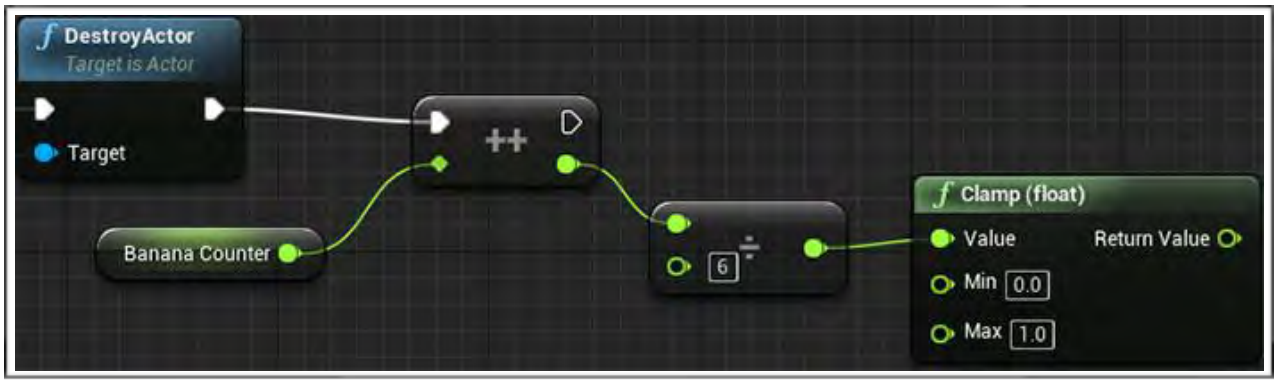
将 float/float 节点底部的输入值设置为 6，这就意味着当方块完全变红时，玩家需要收集 6 根香蕉。



现在还有一个小问题。当玩家收集的香蕉数量少于 6 根时，我们将得到超过 1 的 alpha 值。因此，我们需要使用一个 Clamp(float)节点，让 alpha 的数值始终在 0 到 1 之间。

在 Event graph 中添加一个 Clamp(float)节点，然后将 float/float 节点的输入接口连线到 Clamp(float)的输入接口上。

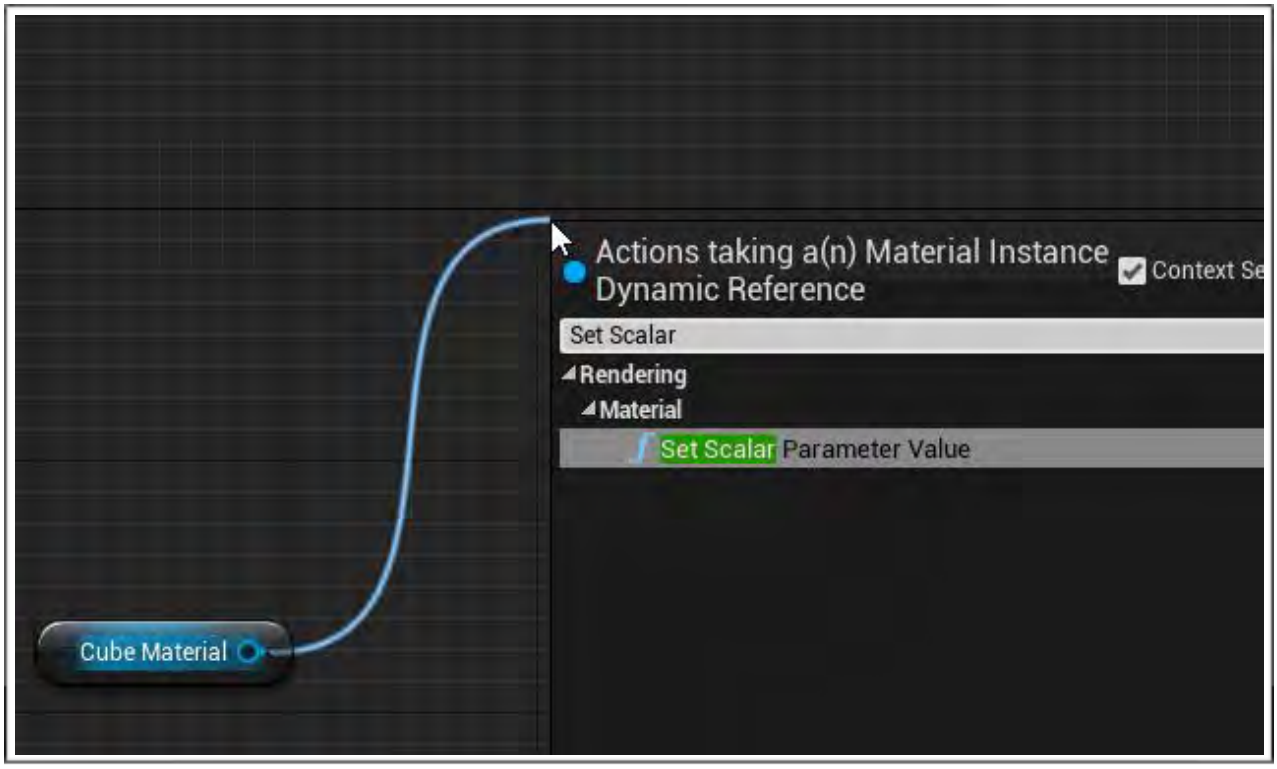
好了，现在我们有了正确的 alpha 值，接下来我们需要将其传递给材质。



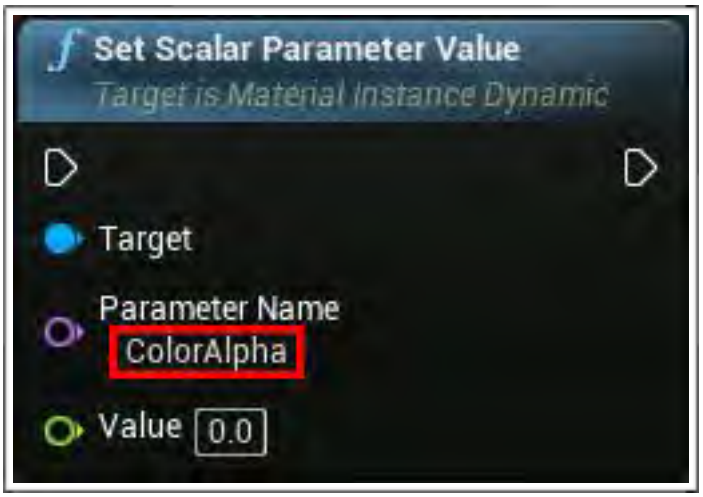
更新材质

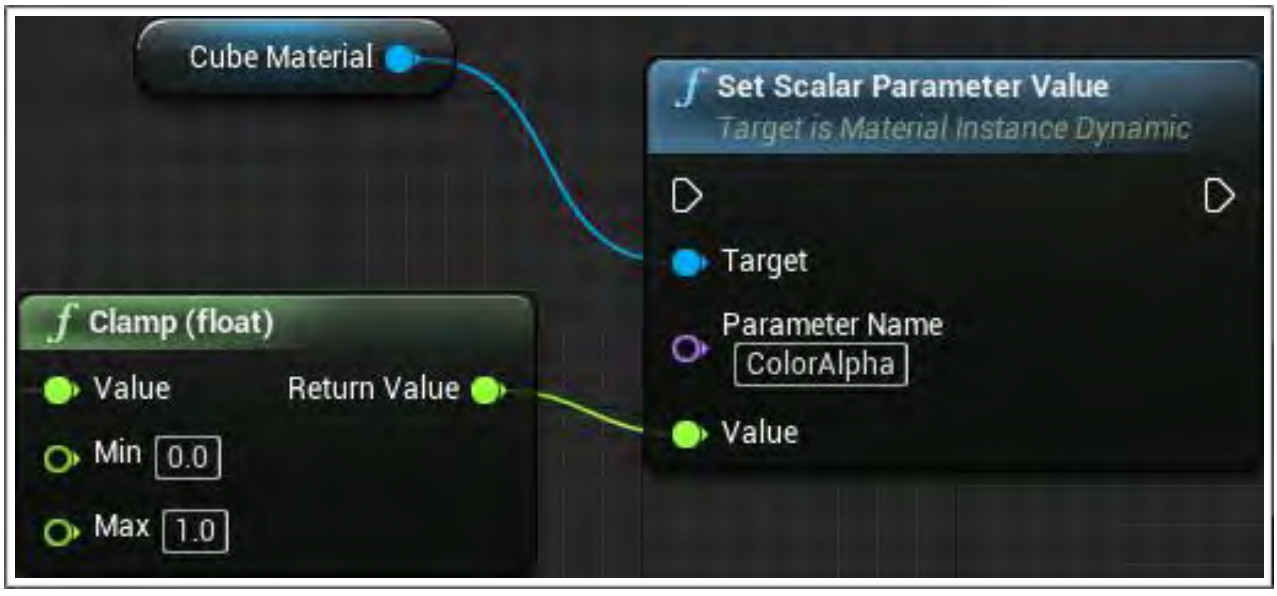
将 CubeMaterial 变量拖曳到 Event Graph 中，然后选择 Get。

接下来从 Cube Material 的右侧输出接口拖一条线到空白区，然后松开鼠标左键，此时我们会看到一个可以使用该类型变量的节点清单。在这里所选中的节点都将自动连接到变量上，这里我们要选择的是 Set Scalar Parameter Value 节点。该节点将为所提供的数值设置一个指定的参数。

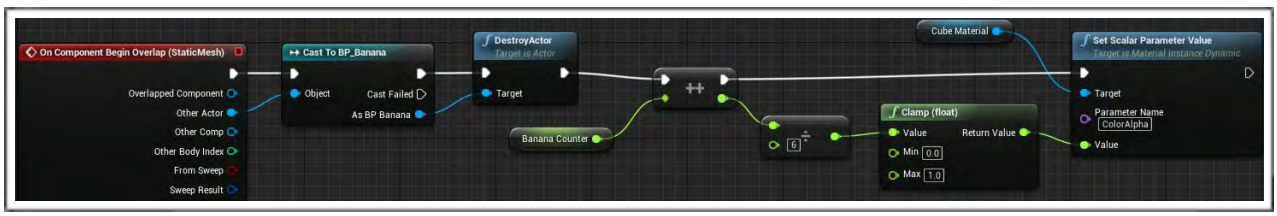


现在我们可以指定要更新的参数。将 Parameter Name 字段设置为 ColorAlpha。该参数也是我们在方块材质中创建的参数。





然后把 *Clamp(float)*节点的结果连接到 *Set Scalar Parameter Value* 节点的绿色输入接口上。



最后，让我们把 IncrementFloat 节点（两个加号的节点）连接到 Set Scalar Parameter Value 节点上。

以下是节点的执行顺序：
1.On Component Begin Overlap(StaticMesh): 当代表玩家角色的方块模型重叠到另外一个角色上时就会执行。

2.Cast to BP_Banana:检查所重叠的角色对象是否是香蕉

3.DestroyActor:如果所重叠的角色对象是香蕉，那么就将其销毁，这样香蕉就会消失。

4.IncrementFloat:让 BananaCounter 变量加 1

5.float/float:让计数器除以一个指定的数字，从而使其标准化

6.Clamp(float):让除法计算的结果限制在 0 到 1 之间

7.Set Scalar Parameter Value:设置方块材质的 ColorAlpha 参数到所提供的数值。而这里，该数值就是之前所计算处的标准化且在 0 到 1 之间的 BananaCounter

好了，一切就绪，点击蓝图编辑器上的 Compile 按钮，然后关闭蓝图编辑器。

回到虚幻 4 的主编辑器，当我们点击 Play 按钮时，使用键盘上的 WSAD 来操控代表玩家角色的方块。可以看到，随着方块碰到的香蕉越来越多，它的颜色变得越来越红。当收集的香蕉数量达到 6 个时，就会完全变红。

好了，本课的内容到此结束。
而本部分关于材质的相关内容也随之结束了。

如果你想了解关于虚幻 4 中材质的更多知识，可以阅读官方的相关文档。
https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/MaterialInputs/index.html

从下一课开始，我们将学习如何在虚幻 4 中添加类似标签或按钮的 UI 元素。

从本课开始，我们将花上几课的时间来学习虚幻 4 的 UI 界面系统。

众所周知，在视频游戏中，开发者使用图形和文字向玩家显示一些重要的相关信息，比如生命值，或是玩家的得分等。而这就是所谓的用户界面（UI）。

在虚幻 4 中，我们通常使用 Unreal Motion Graphics(简称为 UMG)来创建 UI 界面。通过使用 UMG，我们可以轻松的使用可视化的方式来拖曳和放置 UI 元素，如按钮、文本标签等。

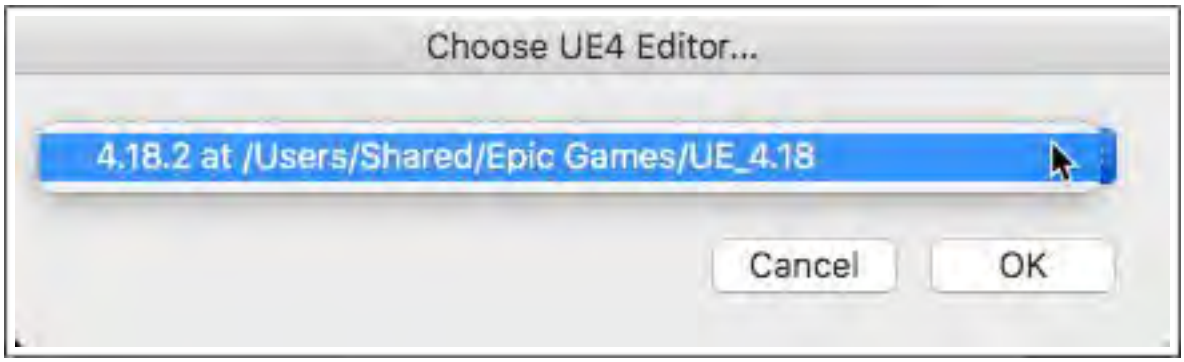
在本教程中，我们将学习以下内容：

1.如何创建一个 HUD（heads-up display），哦用于显示计数器和计时器。
2.如何显示 HUD
3.如何实时更新计数器和计时器中的内容，用于显示变量数值

开始前的准备

在开始学习之前，首先从这里下载项目相关的文件。
链接:https://pan.baidu.com/s/1jHYNcOm 密码:gym2

解压缩后打开 GeometryCatcher.uproject。



提示：
根据你所安装的虚幻 4 引擎的版本，很可能看到下面的提示：

打开项目后，在虚幻 4 的主编辑器中点击 Play 按钮预览游戏效果，可以移动鼠标来控制一个白色的方块左右移动，以便接住落下的形状。而 10 秒钟之后，这些形状就会停止掉落。

在接下来的学习中，我们首先要做的就是创建一个 HUD，用于显示下面的内容：

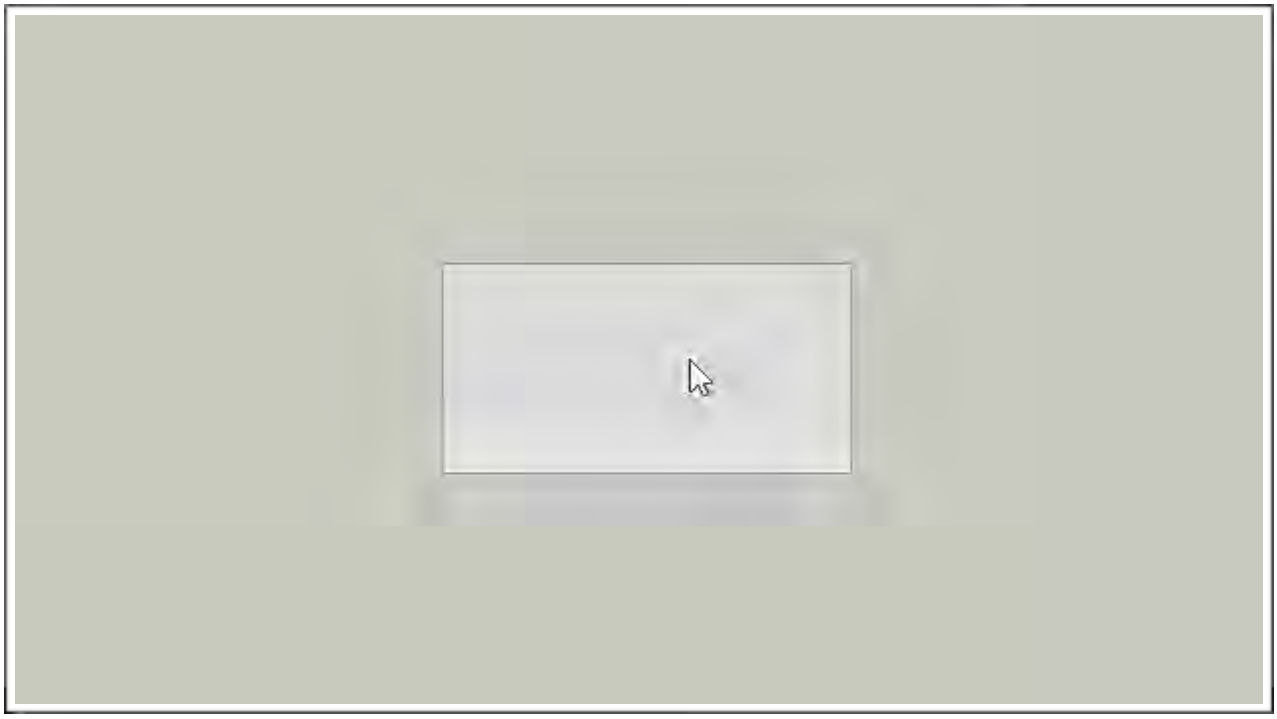
1.一个计数器，用于记录玩家所收集的形状数量
2.一个计时器，用于显示到形状停止掉落之前还剩余的时间。

为了创建以上内容，我们需要使用 widgets。

关于 Widgets

Widget 是一种 UI 元素，可以在 UI 中提供某种视觉功能呢。例如，一个按钮 widget 提供了用户可以查看和点击的对象。
widget 不一定要是可视的。例如，一个 Grid Panel widget 可以让其内部空间均匀分布。玩家不会看到 Grid Panel，但是可以看到它的效果。

Widget 中还可以包含其它的 widget。下图是一个定制化 widget 的例子，其中包含了一个文本 widget（也就是 Name 标签），和一个文本框 widget。



我们甚至可以创建一个 widget 填满整个用户界面，比如菜单界面。下面就是一个 widget 的例子，它看起来就像一个游戏标题界面。所有的 UI 元素都是 widget，同时被包含在游戏界面这个

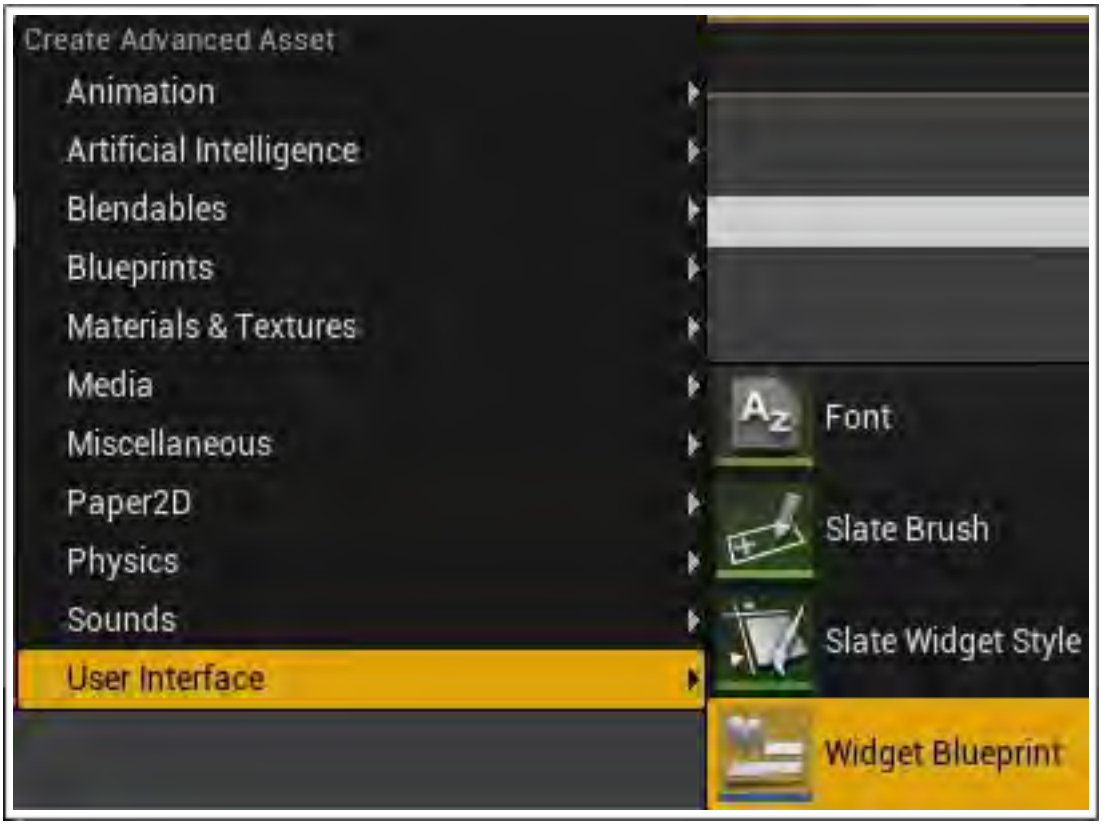
widget 中。
好了，基本的理论知识就到这里了。

从下一课开始，我们将真正创建 HUD。

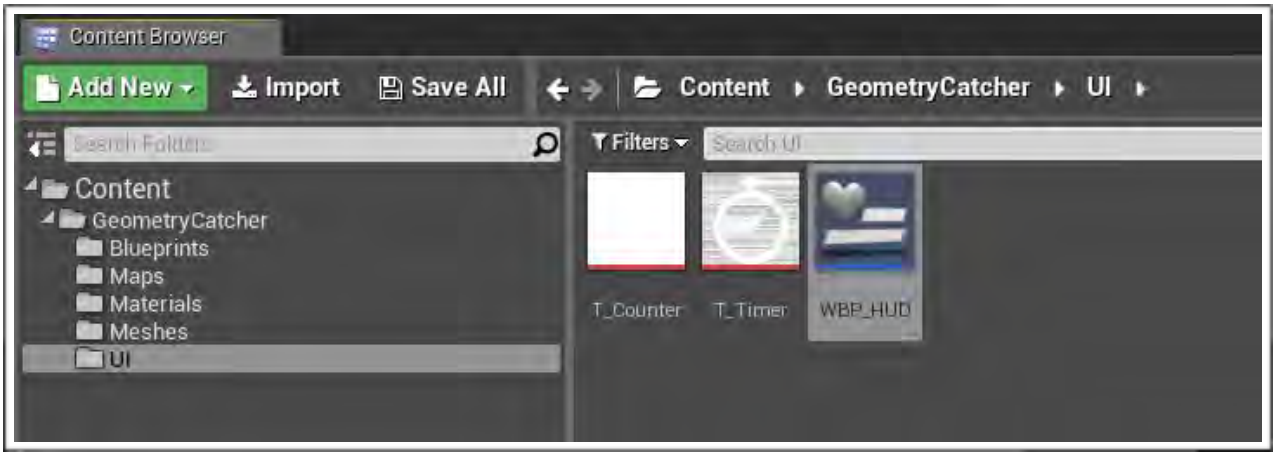
在上一课的内容中我们介绍了关于虚幻 4 中 Widget 的基本知识，从这一课开始我们将正式学习如何创建各种 Widget。

开始创建 Widget

在虚幻 4 中打开上一课的项目，关于如何打开项目的操作后面就不再赘述了。
在主编辑器的 Content Browser 中选择 UI 文件夹。点击 Add New 按钮，然后选择 User

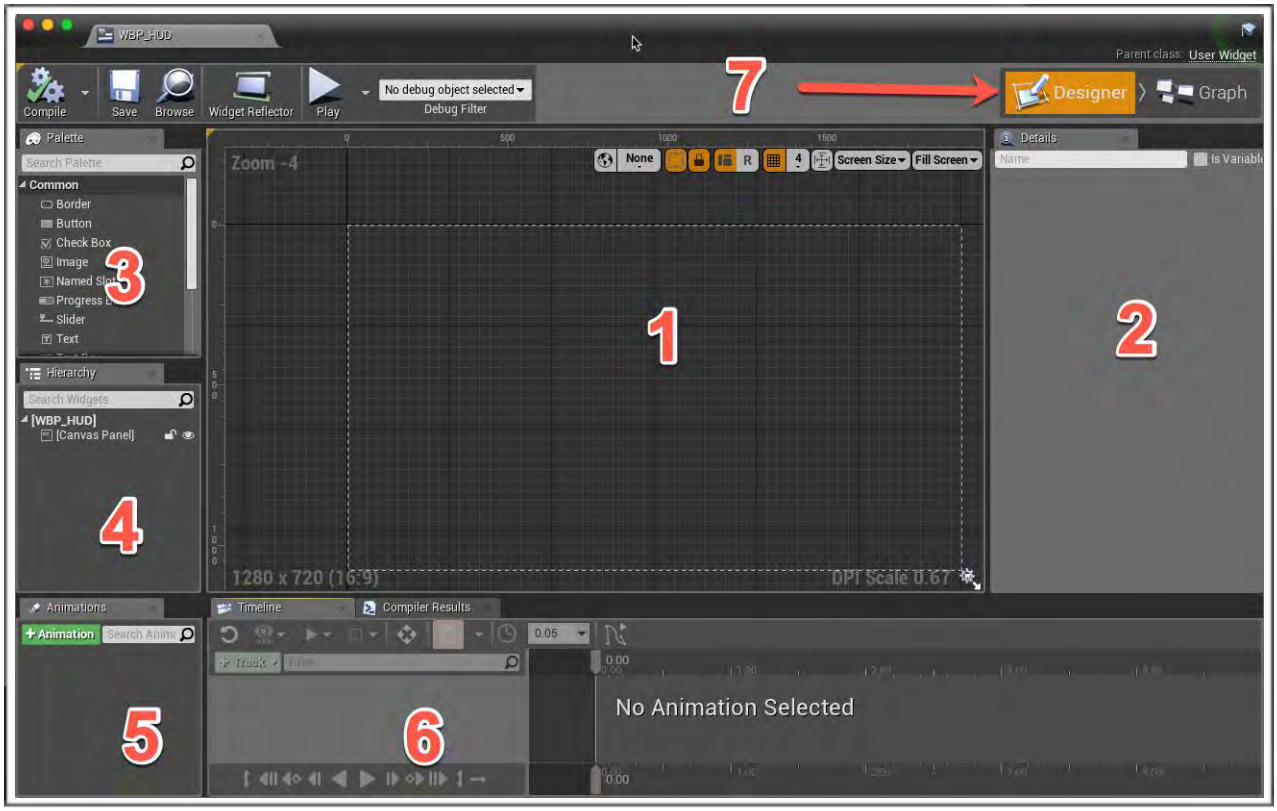


Interface\Widget Blueprint，将新创建的蓝图文件命名为 WBP_HUD。



双击在 UMG UI 设计器中打开 WBP_HUD。

关于 UMG UI Designer（UMG UI 设计器）



UMG UI 设计器由 7 个主要区域组成：

1.Designer（设计区）：

该区域将显示所创建的 widget 的视觉元素。我们可以按住鼠标右键在区域中平移，使用鼠标滚轮来缩放该区域。

2.Details（详情区）：

当选中某个 widget 后，可以在该区域看到相关的属性。

3.Palette（面板）：

在这里可以看到我们可以使用的所有 widget。当然，开发者自己创建的 widget 也会在这里显示。

4.Hierarchy:
这里会看到当前正在使用的所有 widget。

5.Timeline:
当我们选中某个动画后，该面板将显示动画的相关属性和关键帧。

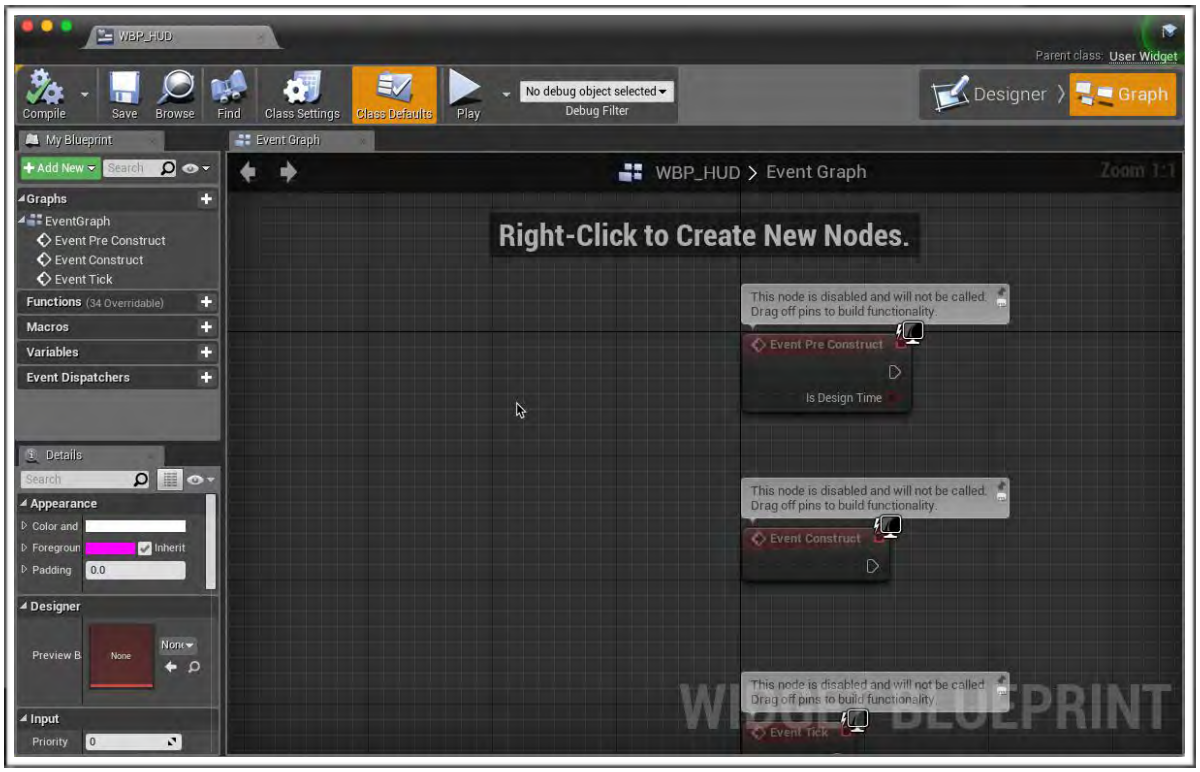
7.Editor Mode(编辑器模式）

可以在这里从 Designer 模式切换到 Graph 模式,Graph 模式和一般蓝图的 Event Graph 没有任何区别。

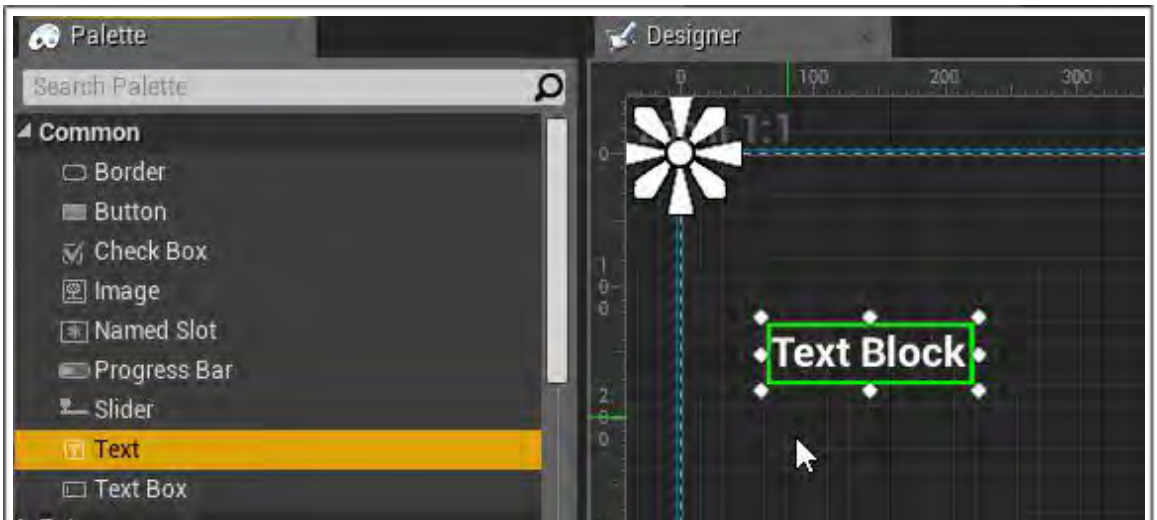
好了，在了解了整个设计器的布局之后，接下来就是具体创建 widget 了。

创建一个文本 widget

对于显示计数器和计时器这样的数字信息，文本 widget 是再合适不过的了。

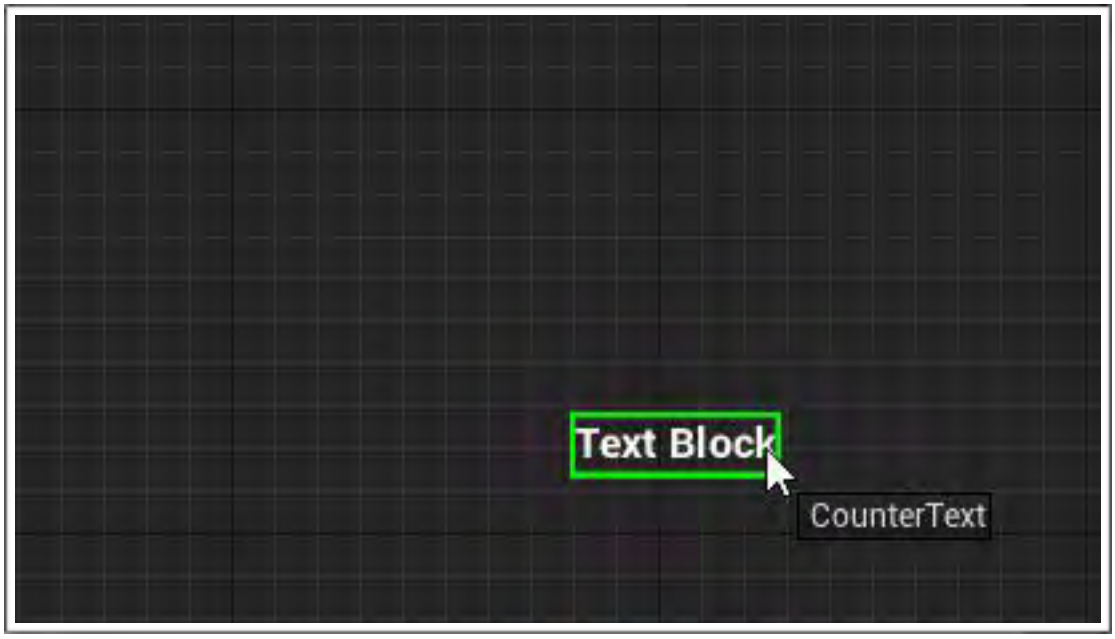


从设计器的 Palette 面板中搜索找到 Text 这个 widget，使用鼠标左键将其拖动到 Designer 面板。
现在暂且不用管文本框中的内容，我们将很快替换其中的内容。

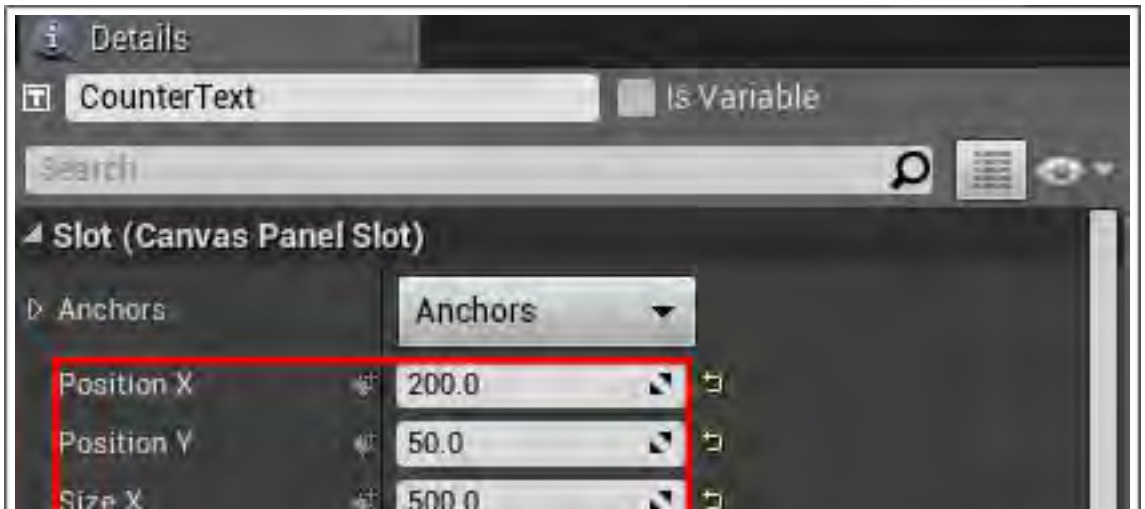


将新创建的 widget 重命名为 CounterText。具体的操作是选中这个文本 widget，然后在 Details 米板中输入顶部的文本框区域。

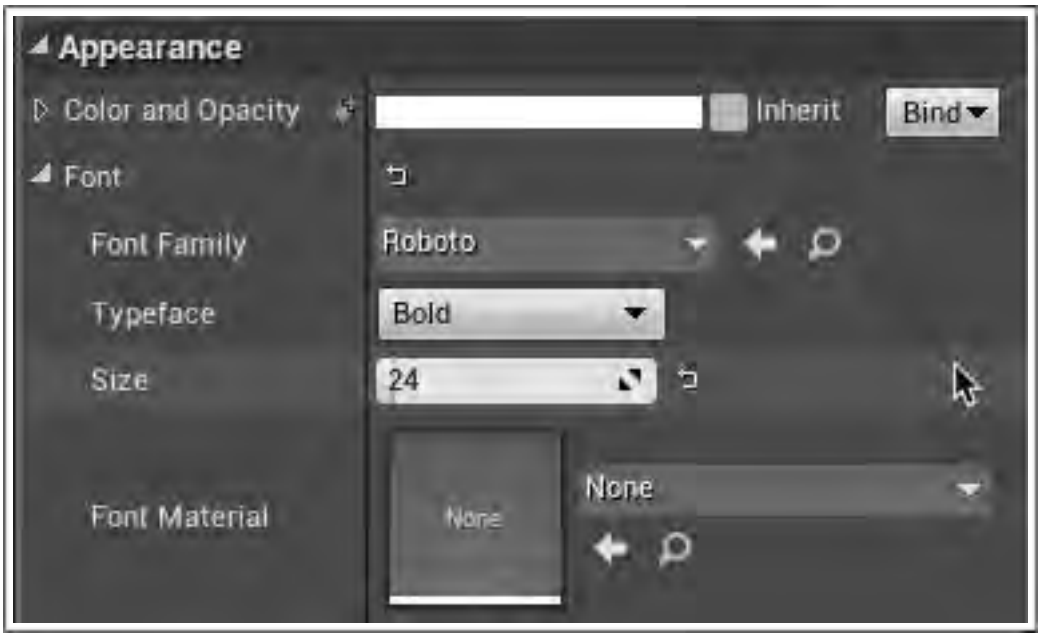
如果你对文本 widget 的位置不满意，那么可以使用鼠标左键在设计器中任意拖动。



接下来我们可以在 Details 面板中设置文本框的精确大小和位置。



此时，文本框中的文字仅仅占用了一小部分空间，我们可以适当调整下字体的大小。



为此，可以在 *Details* 面板中找到 *Appearance* 部分，然后在 *Font* 属性的 *Size* 就是设置字体大小的地方。

这里我们可以将字体设置为 *68*.

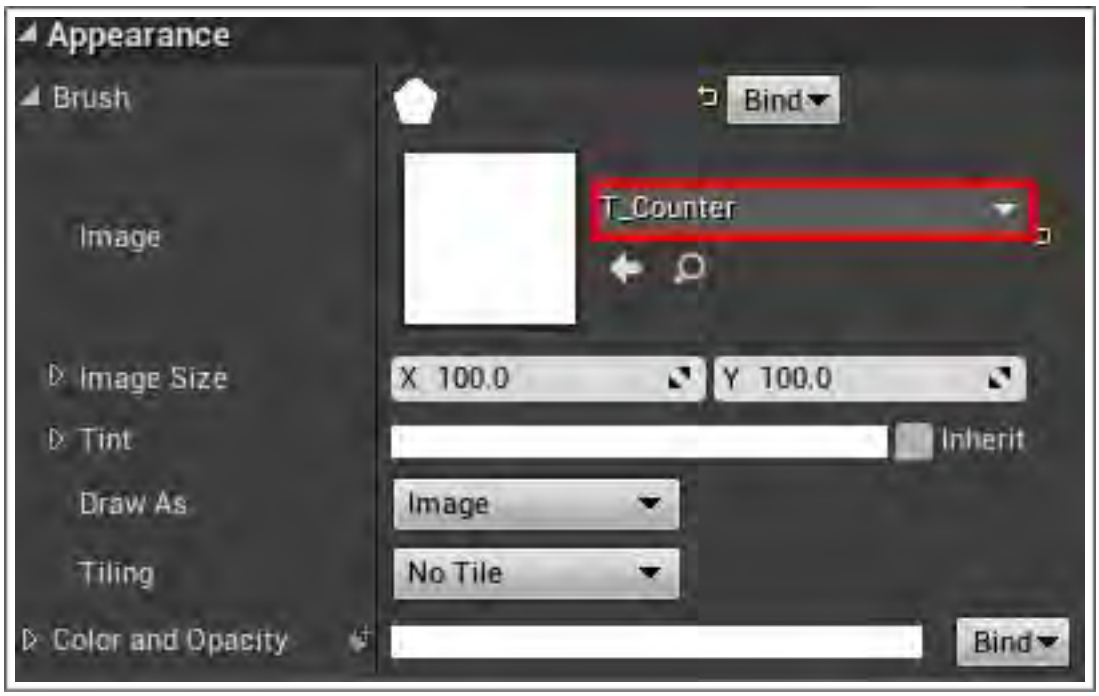
接下来，我们想让计数器变得更漂亮，因此我们需要在其附近添加一个小图标。

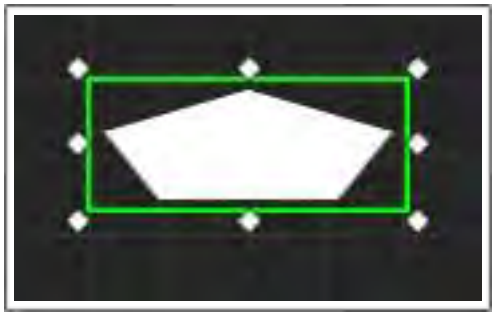
创建 *Image Widget*

图片 widget 可以在 UI 中展示各种图像，比如图标。

使用创建文本 widget 类似的方式创建一个 Image widget，并将其命名为 CounterIcon。将其 Position X 和 Position Y 属性设置为 50，这样就会让它靠近 CounterText。

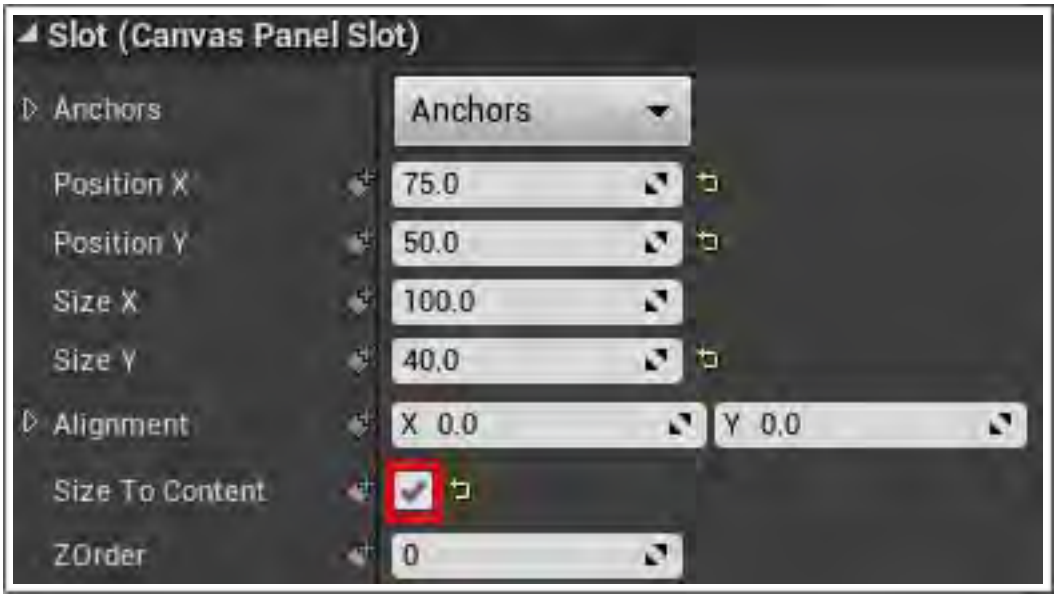
接下来就是设置图片的内容了。在 *details* 面板中找到 *Appearance* 部分，展开 *Brush* 属性，然后从 *Image* 的下拉列表中选择 *T_Counter*。
此时图片的显示有点拉伸的感觉，这是因为 *widget* 的尺寸和图片尺寸是不一致的。



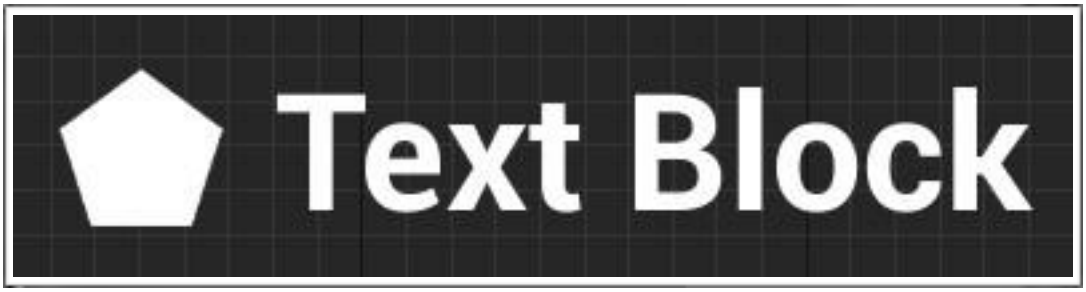


这里我们不需要重新设置 widget 的大小，而是使用 Size To Content 选项。这样就会让 widget 的大小自动适配其中的内容。

具体的做法是，在 Details 面板中找到 Slot(Canvas Panel Slot)部分，然后勾选 Size To



Content 旁边的选框。



此时，widget 就会自动调整其大小来适配图片的内容。

当我们在不同的屏幕尺寸下玩游戏时，UI 需要自动调整其中 widget 的内容来适配屏幕大小。为此，我们将用到 anchors（锚点）的概念。

关于 Anchor（锚点）

听到锚点，大家第一印象可能是想到海中的 和它的锚。

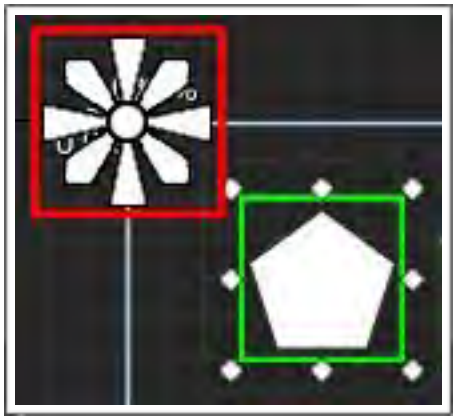
和的锚类似，锚点的作用就是决定一个 widget 的相对位置。默认情况下，widget 的锚点将设置为其父节点的左上角。因此，当我们设置某个锚点的位置时，实际上我们设置的是相对于左上角锚点的相对位置。

记住，每个图片的位置都是相对于其锚点的。通过使用锚点，我们可以确保 UI 在不同的屏幕大

小上有着相同的布局。
此外，我们还可以使用锚点来自动重新设置 widget 的大小。当设置了某个 widget 的两个或更多锚点后，该 widget 就会自动调整其大小，来保持相对大小。

Anchor Medallion（锚点星标）代表了锚点的位置。每当我们选中一个 widget 时，就会看到它的身影~
CounterText 和 CounterIcon 的锚点已经处在正确的位置了，因此我们无需对其进行调整。

好了，本课的学习到此结束。在下一课的内容中，我们将学习创建计时器的文本和图片 widget，并使用锚点将其放置在界面的右上角。



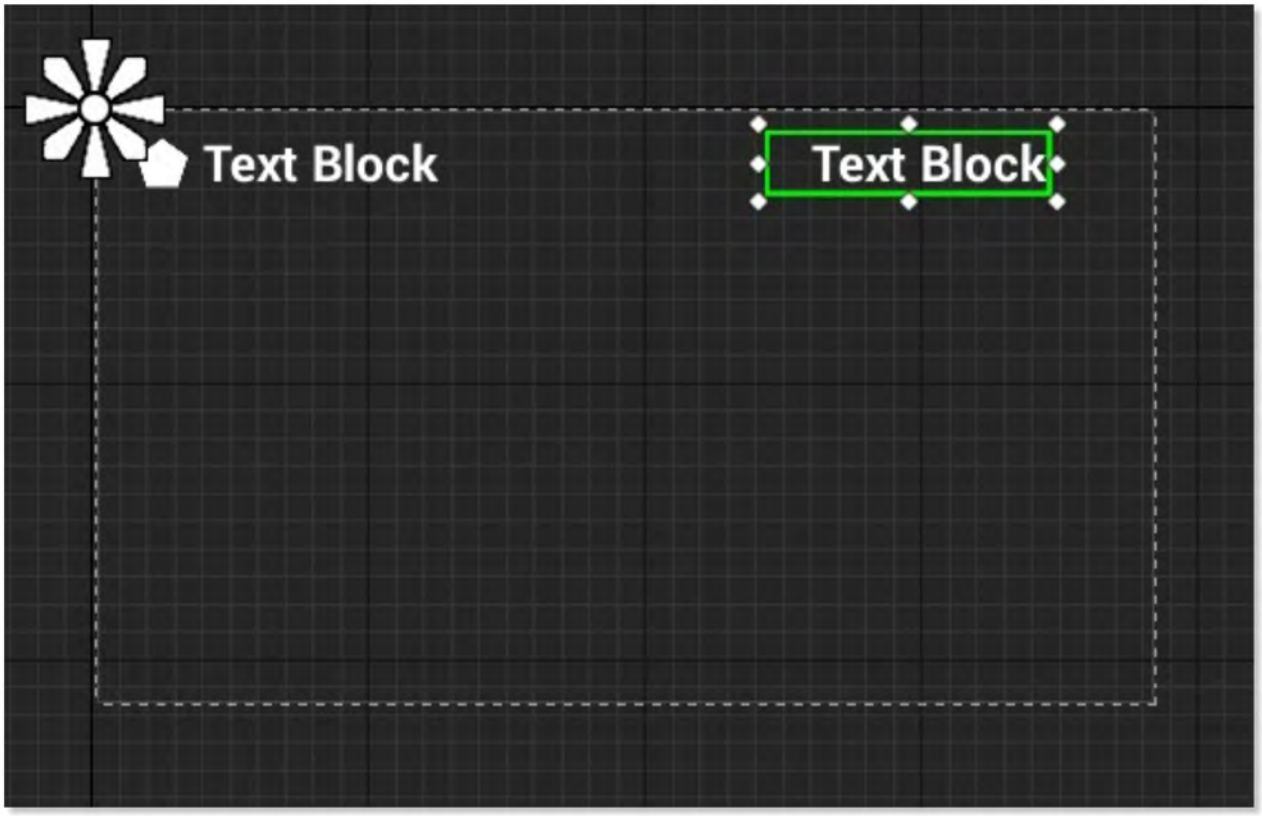
在上一课的内容中，我们创建了第一个 UI widget 计数器。在这一课的内容中，

创建计时器

在虚幻 4 中打开项目，在 Content Browser 中找到 UI 中的 WBP_HUD 蓝图文件，双击将其打开。

使用上一课中所介绍的方法创建一个新的文本 widget，并将其命名为 TimerText。接下来设置以下属性：

- Position X: 1225
- Position Y: 50
- Size X: 500

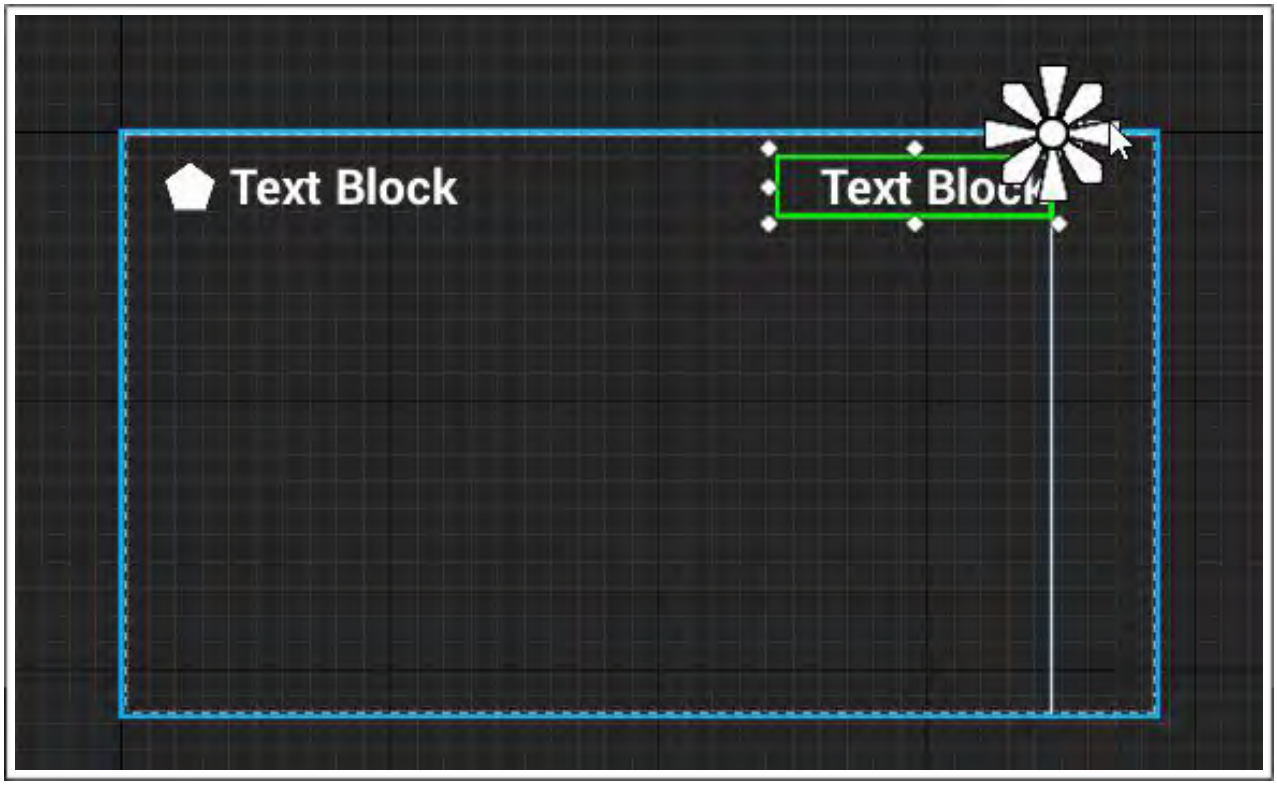


- Size Y: 100
- Font Size: 68
- Justification: Align Text Right (通过该设置将让文本对齐到 widget 的右侧）

接下来我们将把锚点设置到右上角。具体的操作很简单，使用鼠标选中锚点，然后拖动 Anchor Medallion（锚点星标），让其移动到右上角，让蓝色线和虚线对齐，如下图。

注意此时文本 widget 的位置已经更新了，现在是相对锚点的位置。

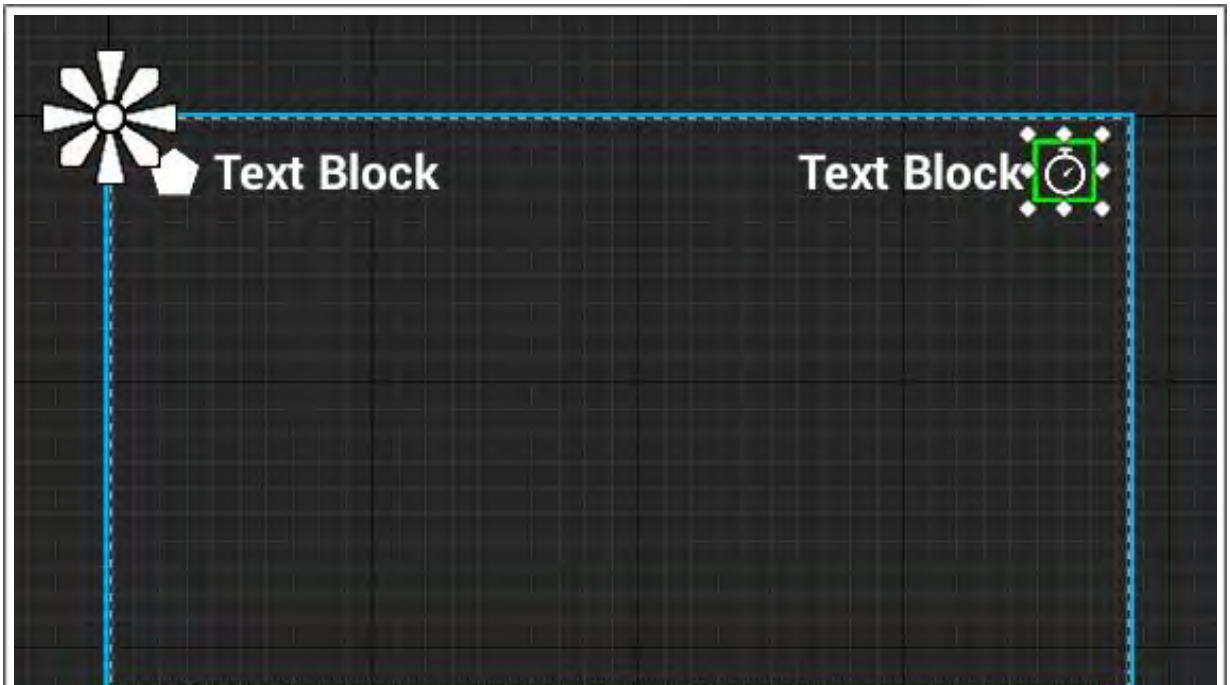
接下来创建一个新的 Image widget，然后将其命名为 TimerIcon。然后设置其属性如下：



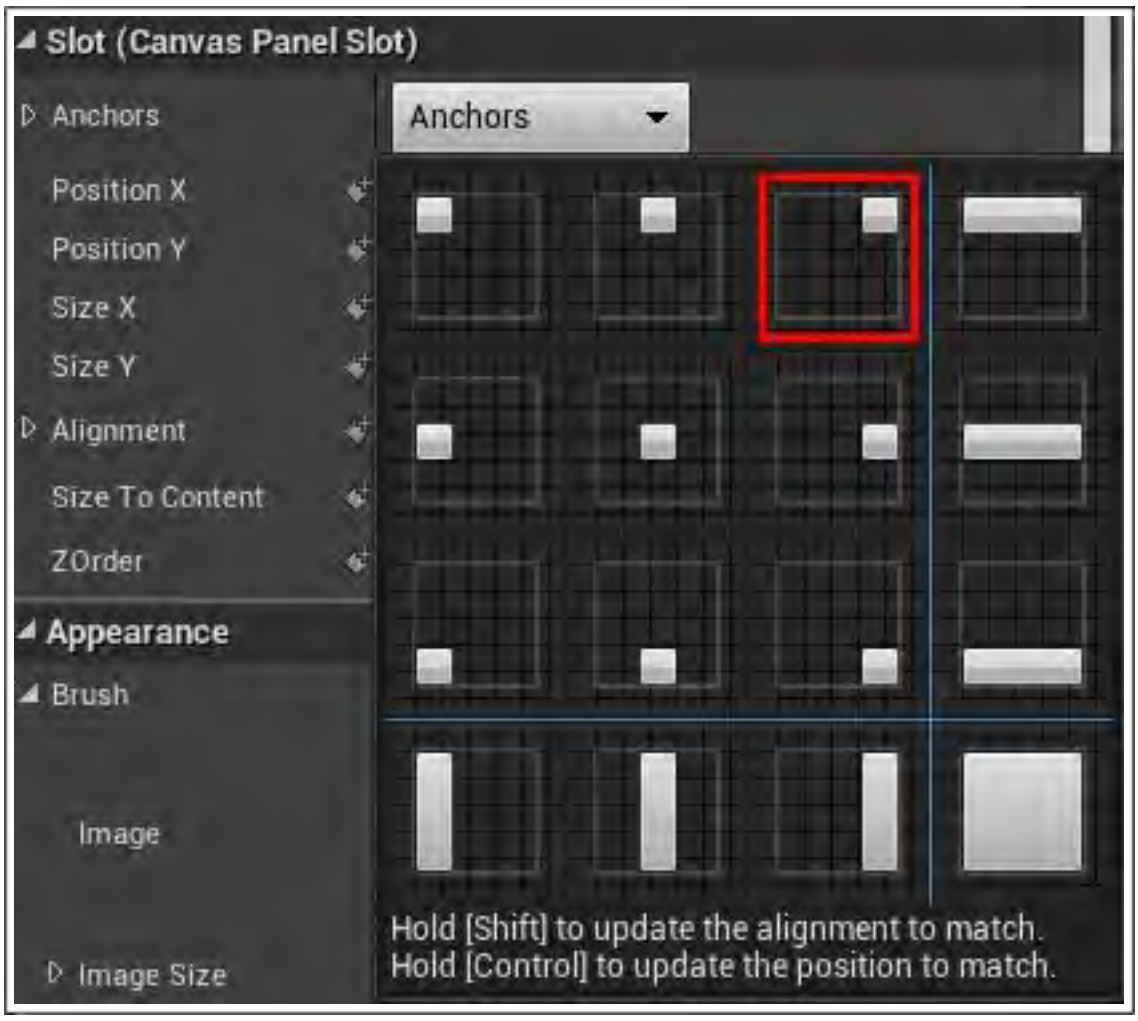
- Position X: 1750
- Position Y: 50

- Size To Content: Checked
- Brush\Image: T_Timer

接下来我们将换一种方式来设置锚点。在 Details 面板中，点击 Anchors 右侧的下拉列表，然后可以显示一些预设选项，从中选择第一排第三个，记住同时按下 Ctrl 键。如图所示。



好了，现在 UI 布局已经完成了。接下来我们将模拟不同屏幕尺寸下的显示。在 Designer 面板中



点击 Screen Size 下拉列表。

从中选择不同的屏幕大小来测试 UI 布局的变化。

可以看到，因为锚点的加入，即便屏幕大小发生了变化，但是整体布局并没有发生变化。

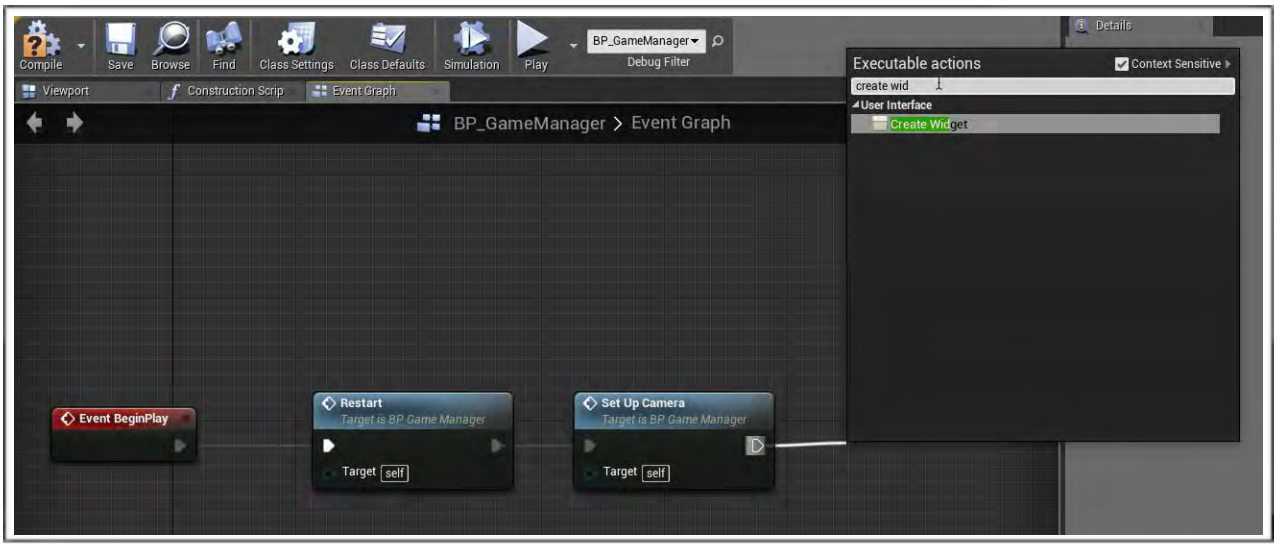
好了，本课的内容到此结束。
在下一课的内容中，我们将学习如何让 WBP_HUD 这个 widget 显示在游戏中。

在这一课的内容中，我们将学习如何在游戏中显示 WBP_HUD widget。

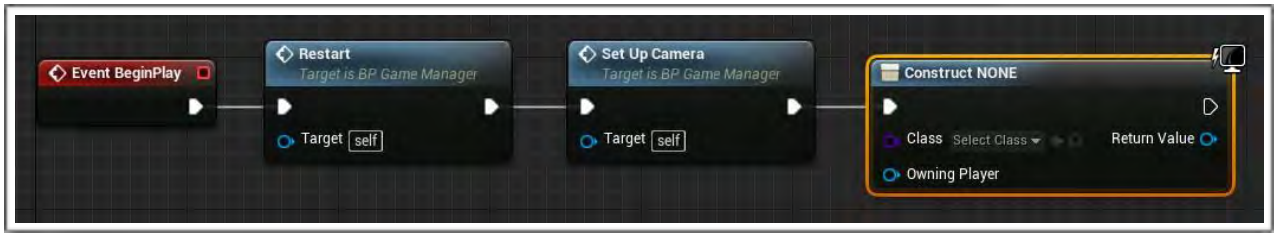
在虚幻 4 主编辑器的 Content Browser 中找到 Blueprints 文件夹，然后双击打开 BP_GameManager 文件。

按照常理，HUD 应该在游戏开始时就显示，为此，我们需要使用 Event BeginPlay 节点。

在 Event Graph 中找到 Event BeginPlay 节点，在白色执行链的最后一个白色输出口处拖出



一条线，然后选择 Create Widget，如图所示。

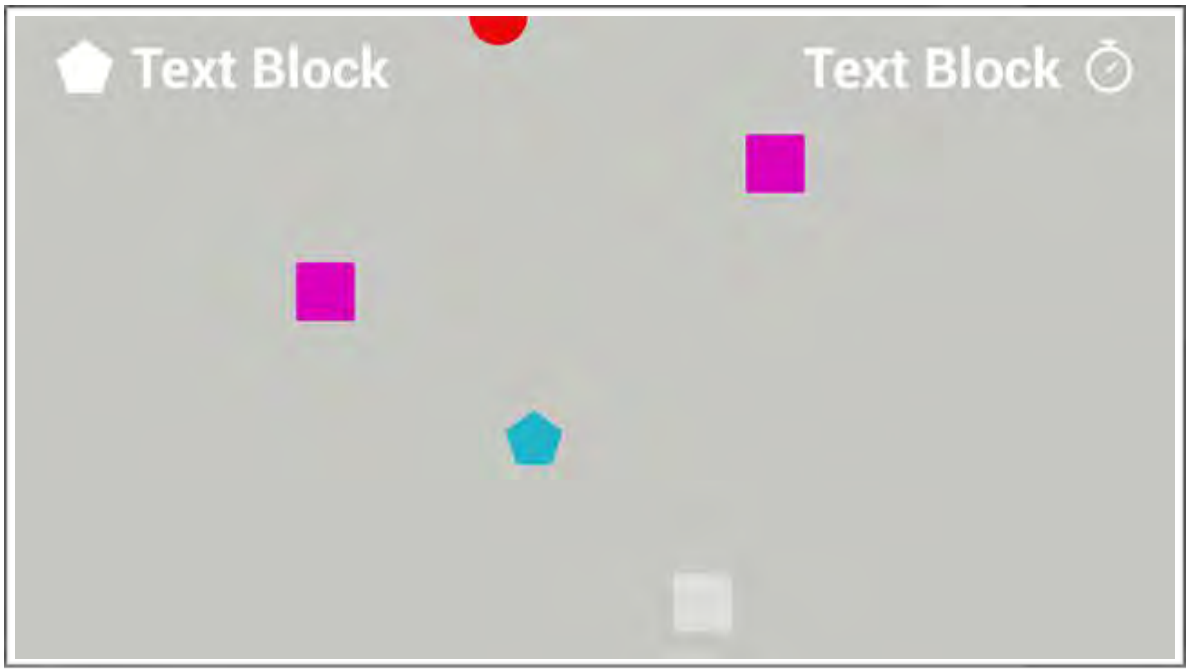


此时，Event Graph 中将出席一个指定 widget 的实例。

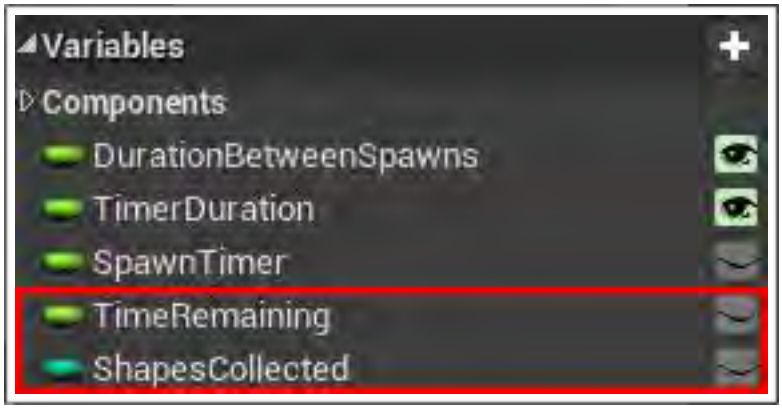
从 Class 旁边的下拉列表中选择 WBP_HUD。

为了显示 HUD，我们需要使用 Add a Viewport 节点，为此，从刚才的 Create Widget 节点的 Return Value 蓝色输出接口拖出一条线，然后从弹出的菜单中选择 Add to Viewport，从而添加一个 Add to Viewport 节点。
接下来让我们回顾一下事件的执行顺序：

1.一旦虚幻 4 引擎生成 BP_GameManager 后，就会执行 Restart 和 SetUpCamera 函数。这些函数的作用是设置了一些参数和摄像机。如果你对函数这个概念还一无所知，没关系，在教程的后面会介绍相关内容。
2.Create Widget 节点创建了一个 WBP_HUD 的实例



3.Add to Viewport 节点将显示 WBP_HUD



好了，点击蓝图编辑器工具栏上的 Compile 按钮，然后返回虚幻 4 主编辑器。点击工具栏上的 Play 按钮来预览游戏效果。

到了这一步，我们已经在游戏场景中显示除了 widget。接下来，我们需要使用变量来保存信息。而这些变量在 BP_GameManager 中已经存在了。

为了使用这些变量，我们需要找到一种方式，可以从 WBP_HUD 中访问 BP_GameManager。

为此可以使用 reference variable 来完成这项工作~

Reference Variable（引用变量）

通过保存一个引用，可以更加方便的访问一个特定的实例。

假定你有一个盒子，里面放着一只球。想从这个盒子里找到球并对其进行检查是很容易的，因为只有一个盒子。

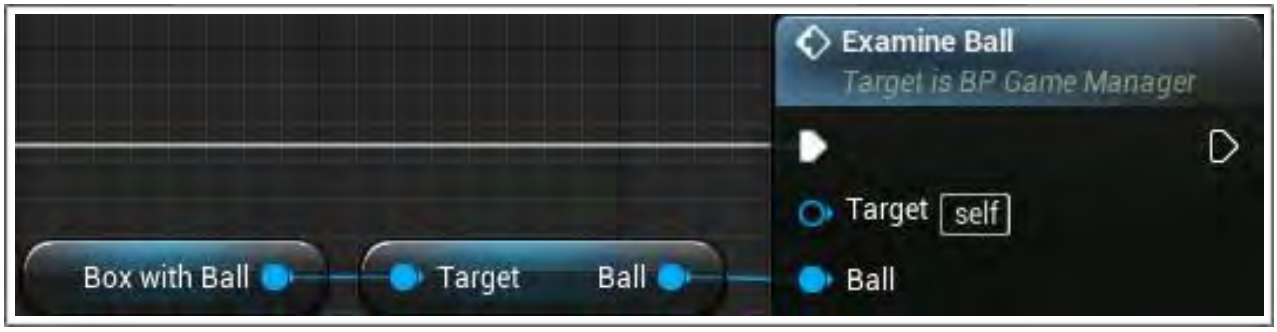
现在假设你有 100 个盒子，但只有其中的一只盒子里面装着一只球。那么我们需要检查每个盒子，直到找到装有球的那只盒子。

每当我们需要检查球的时候，都必须得执行这种操作，这样很快就会导致性能问题。

而通过使用引用，我们可以记住装有球的那只盒子。通过这种方式，就无需再检查每个盒子。

创建引用变量

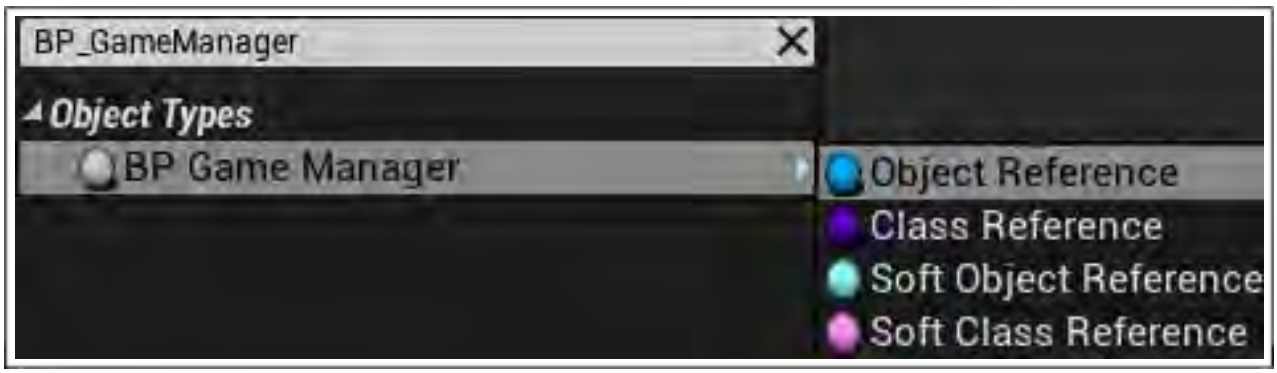
打开 WBP_HUD 蓝图文件，然后切换到 Graph 模式，



接下来在 My Blueprint 选项卡处创建一个新的变量，将其命名为 GameManager。

然后在 Details 面板中点击 Variable Type 旁边的下拉列表，找到 BP_GameManager，并选

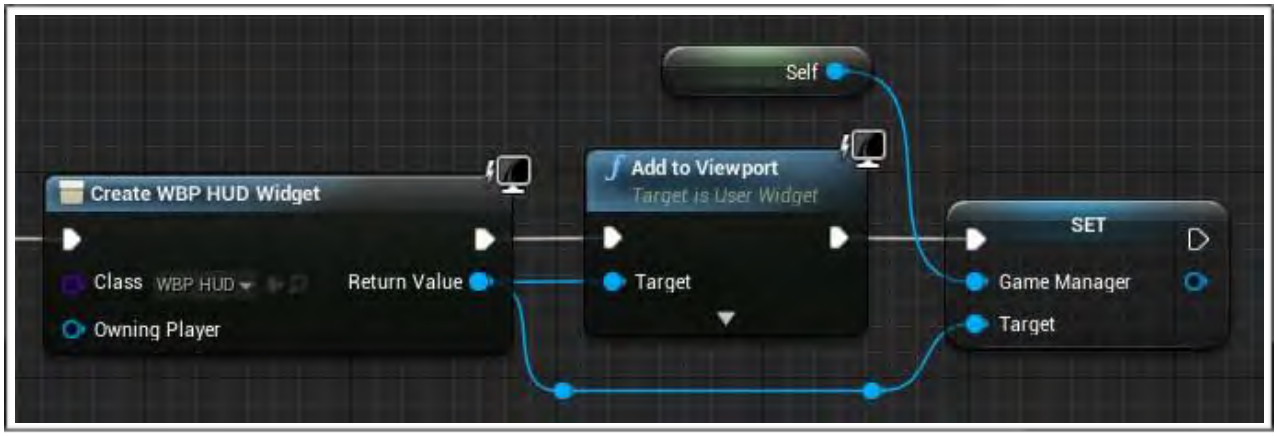
择 BP Game Manager\Object Reference。



设置引用变量

点击工具栏上的 Compile 按钮，然后打开 BP_GameManager。
在 Event Graph 中找到 Create Widget 节点，然后从它的 Return Value 蓝色输出接口处拉一条线到空白处，选择 Set Game Manager 节点，接着从 Add to Viewport 的白色输出接口拉一条线到这个新的节点，如图所示。

接下来，创建一个 Get a reference to SELF 节点，并从它的蓝色输出接口拉一条线到 Set Game Manager 节点的 Game Manager 蓝色输入接口。Self 节点将获得一个到自身的引用。



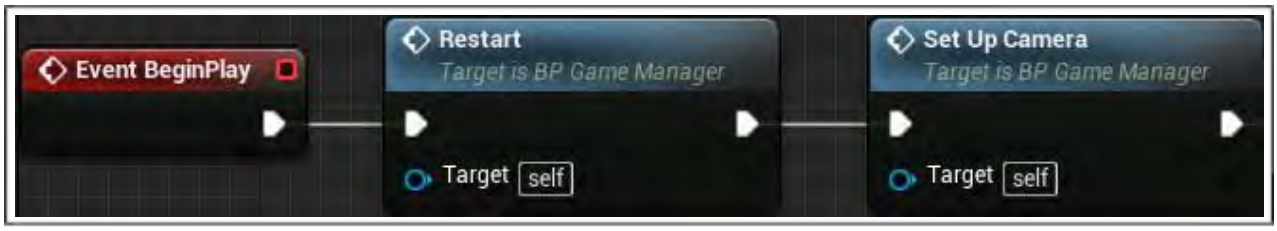
好了，现在 WBP_HUD 已经创建完成，而且拥有一个到 BP_GameManager 的引用。

在下一课的内容中，我们将学习如何在 function 函数的帮助下更新 widget 中的信息。

这一课的内容就到这里了，我们下一课再见~

好了，继续我们的学习。
在这一课的内容中，我们将了解函数的概念。使用函数的一个重要原因就是方便代码的组织。通过使用函数，我们可以把多个节点整合成单个节点。

以 *BP_Gamemanager* 蓝图中的 *Event BeginPlay* 为例，其中用到了两个函数：*Restart* 和 *SetUpCamera*。



如果不使用函数，那么以上部分可能是下面这个样子的~

如你所见，有了函数，一切看起来更加清爽了。

可重用性

我们之所以喜欢函数，还有一个重要的原因就是其可重用性。例如，如果我们希望重置计数器和

计时器，那么久可以使用 *Restart* 函数轻松实现。

通过使用函数，我们可以节省大量的时间。

好了，现在你大概已经知道函数的作用了，那么接下来我们将使用函数来更新 *CounterText* 这个 *widget*。

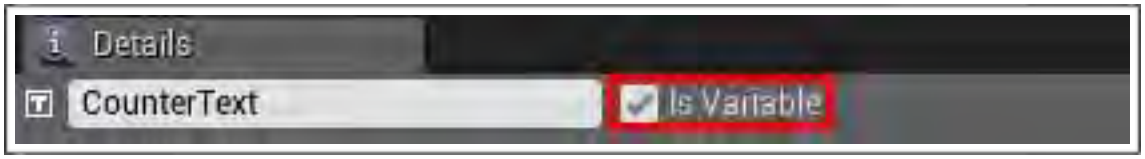
更新 *widget*

当我们创建一个 *widget* 的时候，虚幻 *4* 引擎会自动创建一个到该 *widget* 的引用变量。不过遗憾的是，文本 *widget* 默认情况下并没有引用变量。因此我们无法直接设置其 *Text* 属性。不过幸运的是这一点很容易修复。

在 Content Browser 中找到并打开 WBP_HUD，然后切换到 Designer 模式。
在编辑器左侧的 Hierarchy 面板中选中 CounterText，然后在 Details 面板中勾选 Is Variable checkbox。

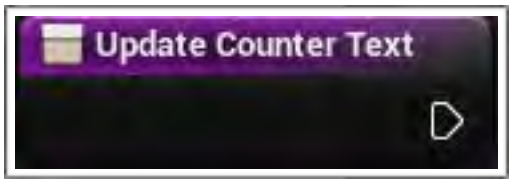
现在我们就可以更新 CounterText 的内容了。接下来要做的就是创建一个函数，来更新文本内容。

创建 Update 更新函数



在编辑器中切换回 Graph 模式，然后在 My Blueprint 选项卡中点击 Functions 部分右侧的+号。
这样我们就创建了一个新的函数，然后自动打开其 Graph 视图。将函数重新命名为 UpdateCounterText。

默认情况下，在 Graph 视图中已经包含了一个 Entry 节点。当函数执行时，这里就是函数的起点。



为了让 CounterText 可以显示 ShapesCollected 变量的数值，我们需要在二者之间创建关联。

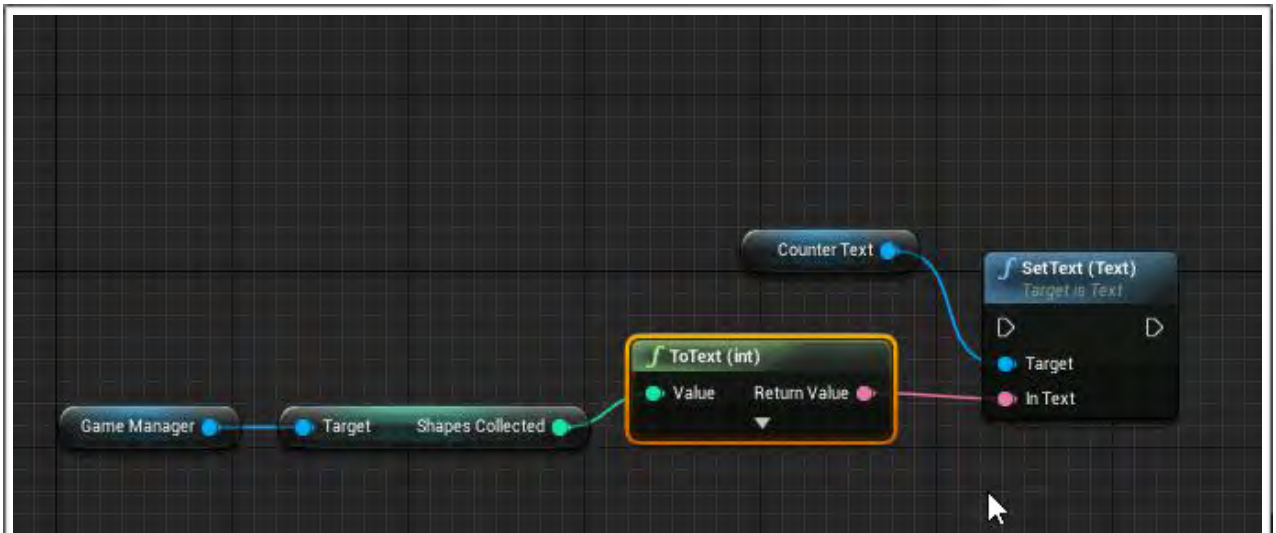
将 GameManager 变量拖动到 Event Graph 视图中。点击节点右侧的蓝色输出接口，拖出一

条线到空白区域，然后从弹出的菜单中选择 Get Shapes Collected。

如果要设置文本，我们就需要使用 SetText(Text)节点。将 CounterText 变量拖动到视图中。

点击节点右侧的蓝色输出接口，拖出一条线到空白区域，然后从弹出的菜单中选择 SetText(Text）节点。

SetText（Text）节点只接受 Text 类型的输入，但 ShapesCollected 变量则是 Integer 整数类型的。不过虚幻 4 显然已经替我们考虑到了这一点，当我们将某个 Integer 变量的输出接口连接

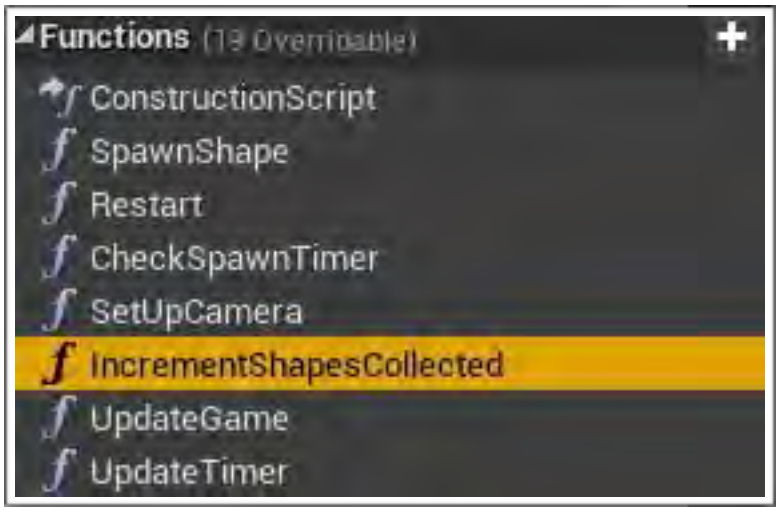


到 Text 类型的输入接口时，会自动帮我们完成转换工作。

为了完成函数，我们还需要将函数的 Entry 节点连接到 SetText 函数节点上，如图所示。

事件的执行顺序如下：
1.当我们调用 UpdateCounterText 函数时，首先它会从 BP_GameManager 中获取 ShapesCollected 变量
2.ToText(int)节点将把 ShapesCollected 的值转换成 Text 类型。
3.SetText(Text)函数将会把 CounterText 的文本内容设置为 ToText(int)的输出结果。

接下来我们要做的就是当玩家收集某个形状的时候调用 UpdateCounterText 方法。



调用 Update 函数

最适合调用 UpdateCounterText 函数的地方就是当游戏中所收集到的形状 ShapesCollected 数量增加时。这里我已经创建了一个名为 IncrementShapesCollected 的函数，可以增加计数器的数量。每当形状叠加到玩家上时，就会调用该函数。

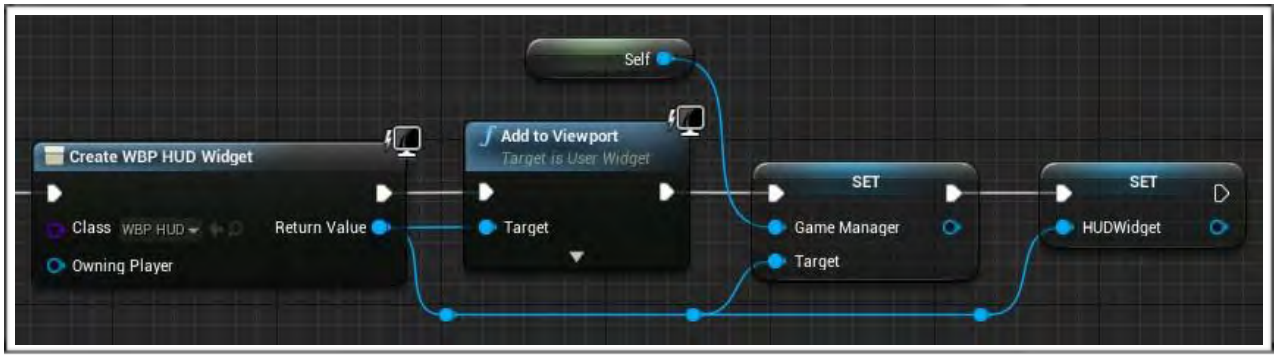
点击工具栏上的 Compile 按钮，然后返回 BP_GameManager。

最后，在我们调用 UpdateCounterText 之前，需要为 WBP_HUD 创建一个引用。

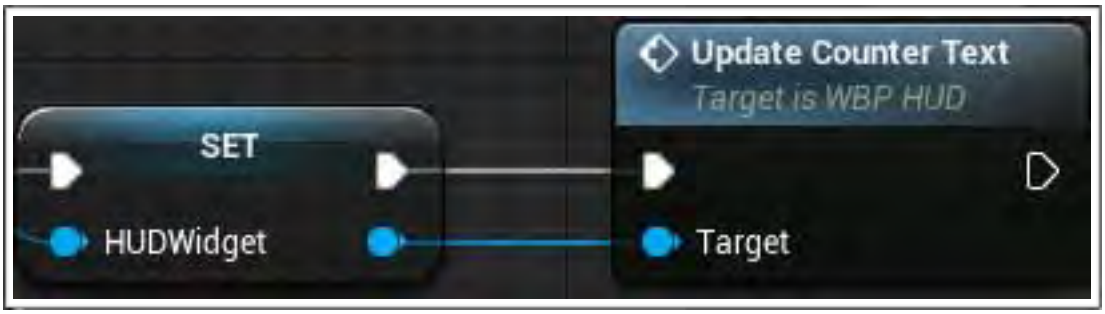
具体操作如下：
1.在 Event Graph 中找到创建和显示 WBP_HUD 的地方
2.左键单击，然后从 Create Widget 节点的 Return Value 接口处拖一条线出来。
3.在空白处释放，并选择 Promote to variable
4.将新的节点添加到节点链中。

当我们创建了变量后，可以将其重命名为 HUDWidget。
接下来，从 Set HUDWidget 节点的右侧接口拖一条线到空白区，然后添加一个 UpdateCounterText 节点。这样就可以在游戏开始后让 CounterText 现实 ShapesCollected 的值。

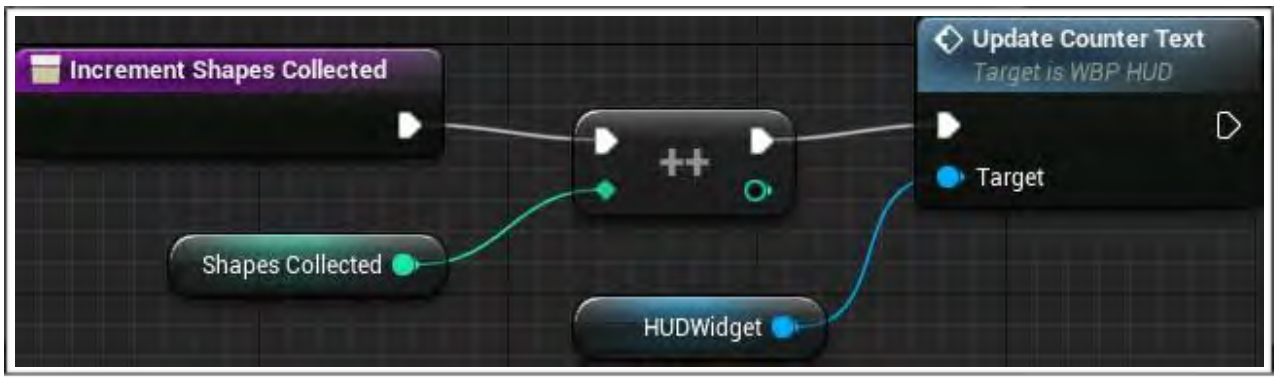
接下来在 My Blueprint 面板找到 Functions 部分，双击 IncrementShapesCollected，打开其 graph。
把 HUDWidget 变量拖动到 graph 中，按住鼠标左键从其接口拖一条线到空白区，添加一个



UpdateCounterText 节点，然后使用以下方式连接：



好了，现在只要 IncrementShapesCollected 开始执行，就会自动将 ShapesCollected 加 1，并调用 UpdateCounterText 函数。而这个函数则会将 CounterText 的内容更新为



ShapesCollected 的数值。

点击工具栏上的 Compile 按钮，然后关闭 BP_GameManager。点击主编辑器你上的 Play 按钮，然后可以开始收集形状，并观察 CounterText widget 的内容变化。

好了，本课的内容到此为止。

在下一课的内容中，我们将使用另一个名为 *binding* 的方法来更新 *TimerText widget* 的内容。

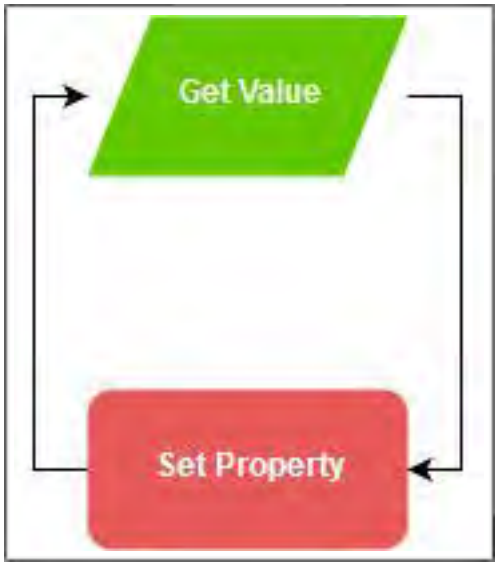
在本课的内容中，我们将使用另一个名为 *binding* 的方法来更新 *TimerText widget* 的内容。

Bindings

使用 *binding*，可以自动更新特定的 *widget* 属性。但在此之前，*widget* 的相关属性必须有

Bind 下拉列表。

我们可以将属性绑定在 *widget* 中的函数或变量,*binding* 就会不断从该函数或变量中获取数值，

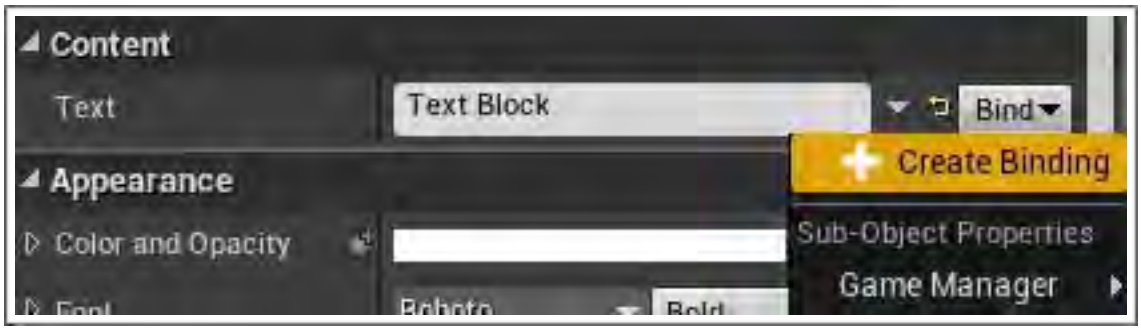


然后将绑定的属性值设置为所获取的值。

可能你要问了，既然 *binding* 这么方便，为什么我们不干脆一直就使用 *binding* 呢？这是因为 *binding* 是比较低效的做法，因为它需要不断更新内容。这就意味着即便没有更新的信息，游戏也需要消耗资源更新相关属性。

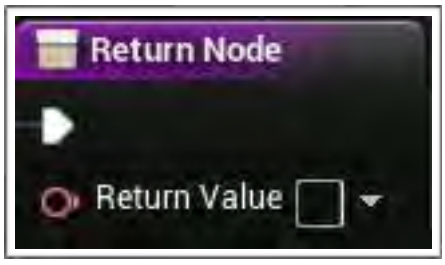
话虽这么说，对于变化频率非常高的元素，比如 *timer* 这种，*binding* 还是很好用的。接下来让我们为 *TimerText* 创建一个 *binding*。

创建 *binding*



打开 *WBP_HUD*，然后切换到 *Designer* 模式。

选中 *TimerText*，然后在 *Details* 面板中找到 *Content* 部分。你会看到此处的 *Text* 属性是可绑定的，点击 *Bind* 的下拉列表，然后选择 *Create Binding*。



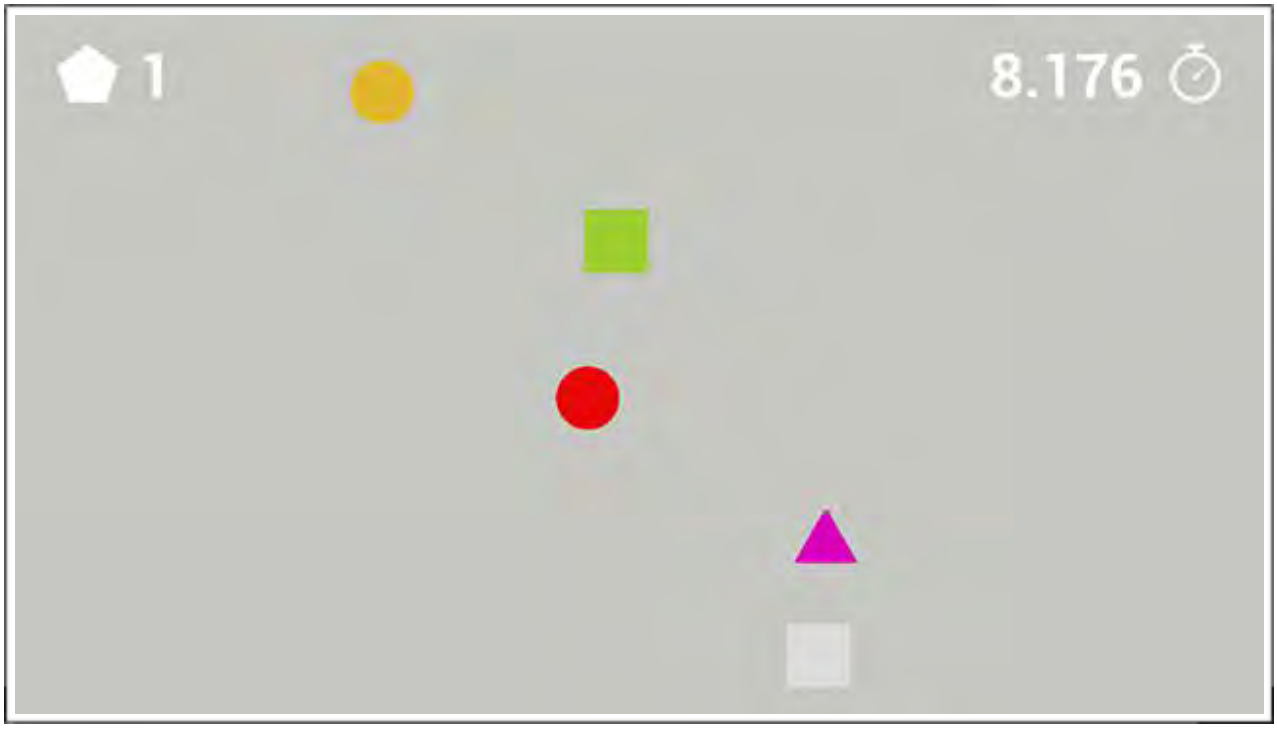
此时会创建一个新的函数，并打开其 *graph*。这里将其更名为 *UpdateTimerText*。

可以看到，该函数有一个 *Return* 节点，同时还有一个 *Text* 类型的 *Return Type* 接口。

TimerText 中将显示所有连接到该接口的文本。

将 GameManager 拖动到 graph 中，然后从中获取 TimeRemaining 变量。

将 TimeRemaiining 变量连接到 Return 节点的 Return Value 接口，这样虚幻就会自动为我们添加一个转换节点。
小结一下:



1.binding 会持续不断的调用 UpdateTimerText 函数
2.该函数从 BP_GameManager 中获取 TimeRemaining 变量。
3.ToText(float)节点将会把 TimeRemaining 的值转换为 Text 类型
4.转换后的值将输出到 Return 节点

好了，HUD 到此就完全完成了。点击编辑器上的 Compile 按钮，然后关闭 WBP_HUD.点击 Play 预览最终的游戏效果。

好了，到此为止,我们的 UI 部分教程就结束了，该部分的示例项目链接：
链接:https://pan.baidu.com/s/1kVDxb75 密码:z4yn

在本部分内容中，我们了解了 UMG 的基础知识，在此基础上，我们可以创建更为复杂的交互界面。

如果你想了解 UI Widget 的更多知识，可以参考虚幻 4 的官方文档：

Widget Type Reference

从下一课开始，我们将使用已经学到的知识制作一个简单的游戏。
好了，我们下一课再见。

欢迎继续我们的学习。

https://www.raywenderlich.com/168091/create-simple-game-unreal-engine-4

在之前的内容中，我们已经了解了关于虚幻 4 引擎的基本知识，包括蓝图，材质，UI 等等。

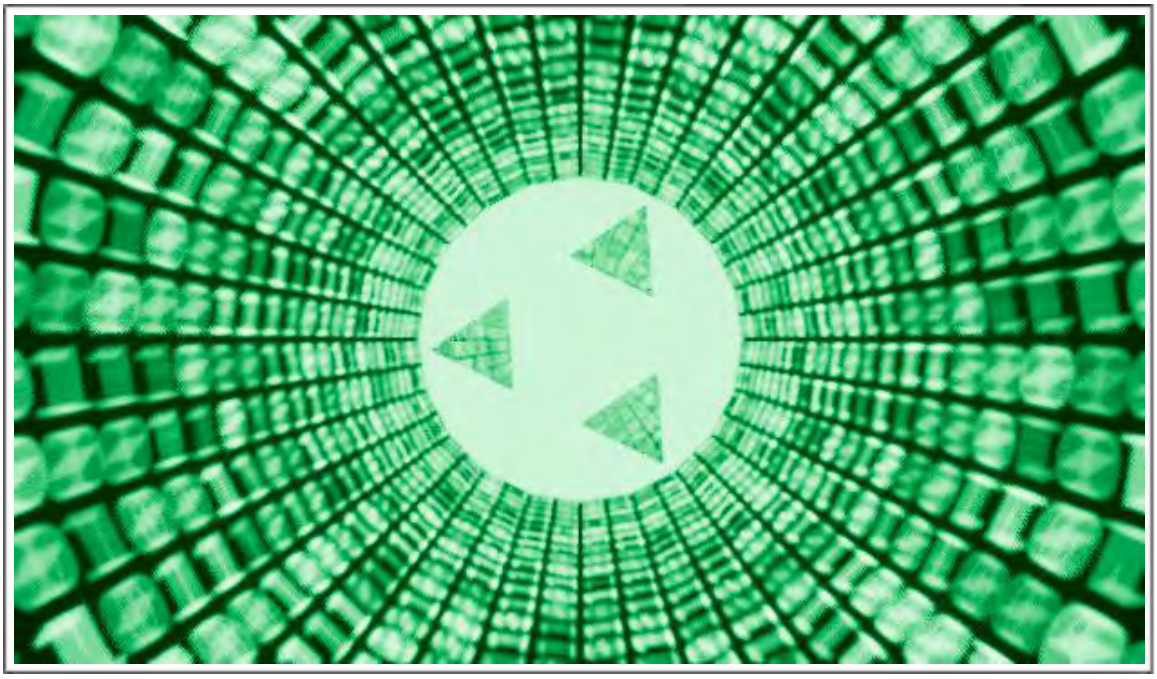
而关于游戏开发，最重要的是什么呢？当然就是从零开始创建一款简单的游戏了。因为只有在游戏实战开发的过程中，我们才能了解到基本的游戏机制，以及游戏世界中的对象是如何互相进行交互的。

从这一课开始，我们将创建一个简单的 FPS 游戏。在这个过程中，我们将学习以下知识：

1.让玩家角色可以持续向前移动

2.生成玩家角色必须避开的障碍物

3.让障碍物随机分布

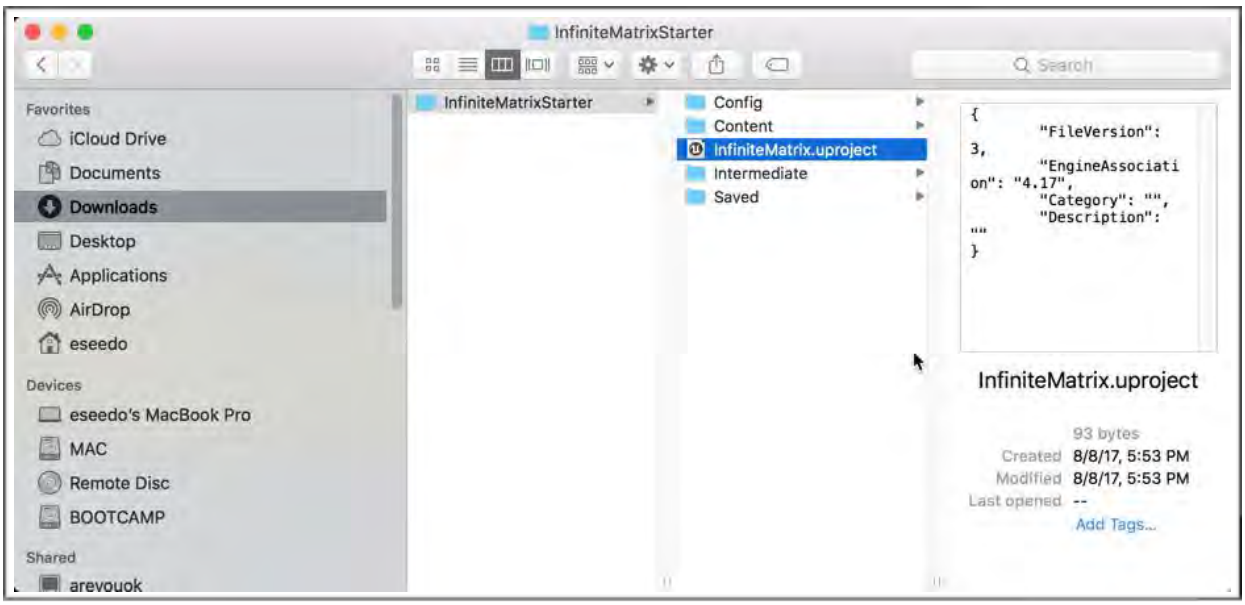


4.当玩家角色不小心（或者有意~）碰到障碍物的时候，显示一个重新开始的按钮。

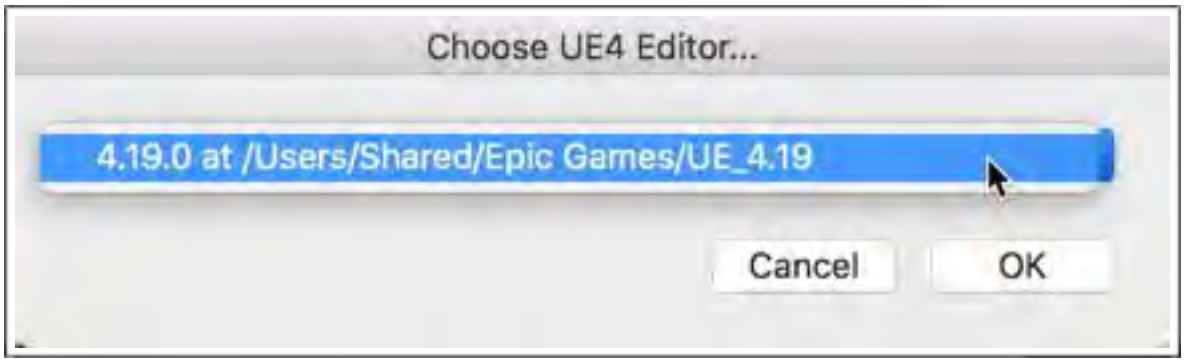
需要注意的是，在本系列教程中我们将用到蓝图和 UMG。

如果你对蓝图和 UMG 比较陌生，建议先复习一下之前的教程。

好了，一切就绪，开始我们的学习吧~



开始前的准备

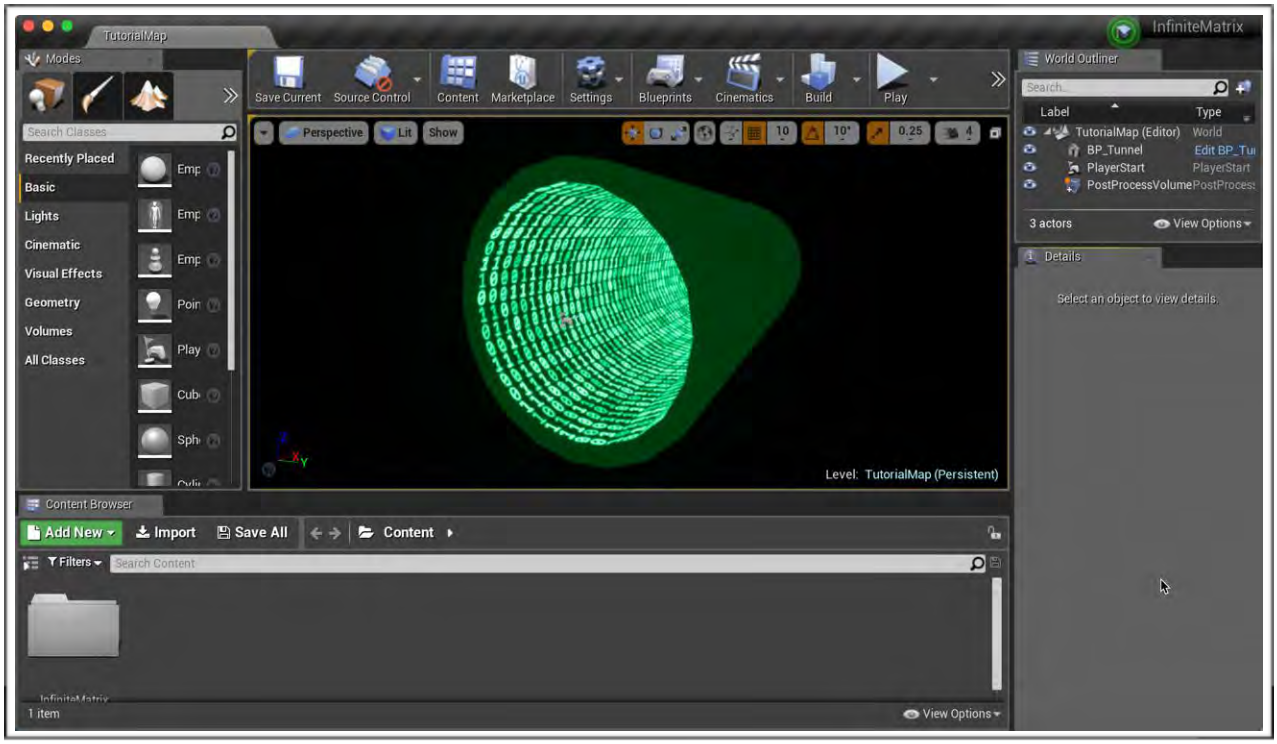


首先，从这里下载起始项目，然后把它解压缩到本地硬盘。打开解压后的项目文件夹，然后双击打开 *InfiniteMatrix.uproject*。

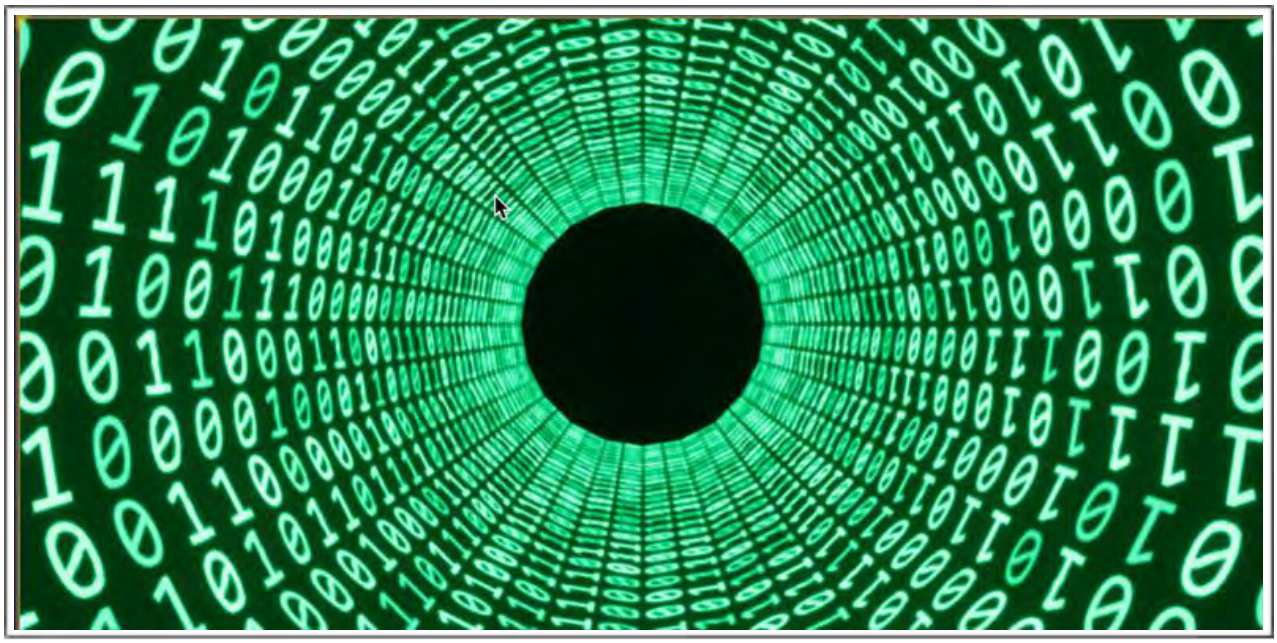
在打开项目的时候， 可能会让你选择自己的 *UE4 Editor* 版本，如下图：

打开之后会看到类似下面的界面。

点击工具栏上的 *Play* 按钮，可以通过上下左右移动鼠标在场景中运动。



怎么样？是不是有点黑客帝国的味道~



接下来我们要做的第一件事就是让玩家角色在场景中不断向前运动。

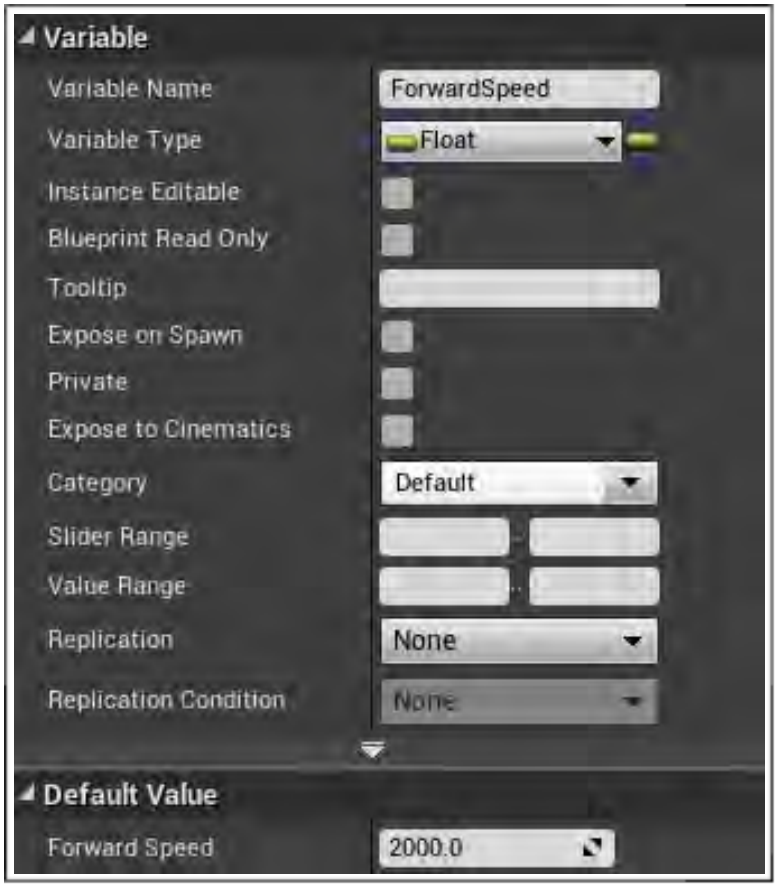
让玩家向前运动

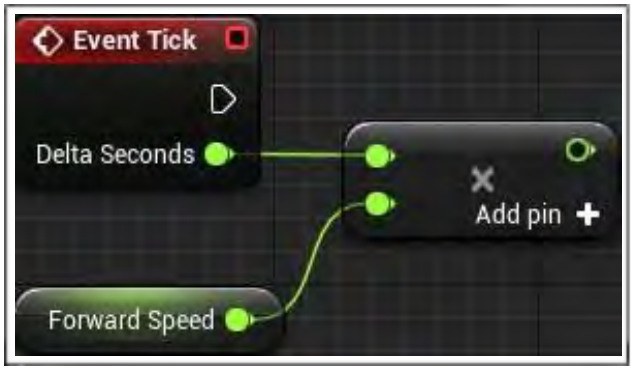
在 Content Browser 中找到 Blueprints 文件夹，双击打开 BP_Player 这个蓝图文件。

为了让玩家角色向前运动，我们需要对玩家的位置每帧提供一个偏移。

为此，首先需要创建一个变量，用于定义玩家角色向前运动的速度。
在蓝图编辑器的左侧找到 Variables，点击+号创建一个新的变量，将其命名为 ForwardSpeed，然后把类型更改为 Float，把默认值更改为 2000。

需要注意的是，在进行相关设置之前，要先点击主编辑器工具栏上的 Stop 按钮，停止游戏运行。
接下来在蓝图编辑器中切换到 Event Graph，并找到 Event Tick 节点，创建以下的设置：



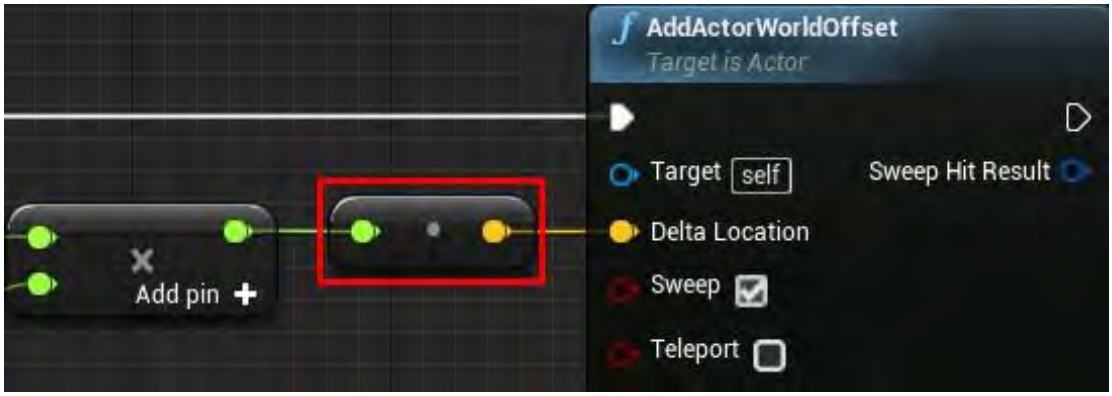


通过让 *ForwardSpeed* 乘以 *Delta Seconds*，就可以得到一个跟帧速无关的结果。

如果你对这一点有点困惑，不妨回过头去复习下蓝图部分的内容。
接下来，我们将使用上面的这个运算结果让玩家角色沿着某个单一轴向运动。

沿着单一轴向运动

为了让玩家角色运动，需要创建一个 *AddActorWorldOffset* 节点，将 *Sweep* 设置为 *true*。
此时，如果我们直接将刚才得到的 *Float* 结果连接到 *Delta Location* 输入端口，虚幻会自动把



它转换成一个 Vector 类型。

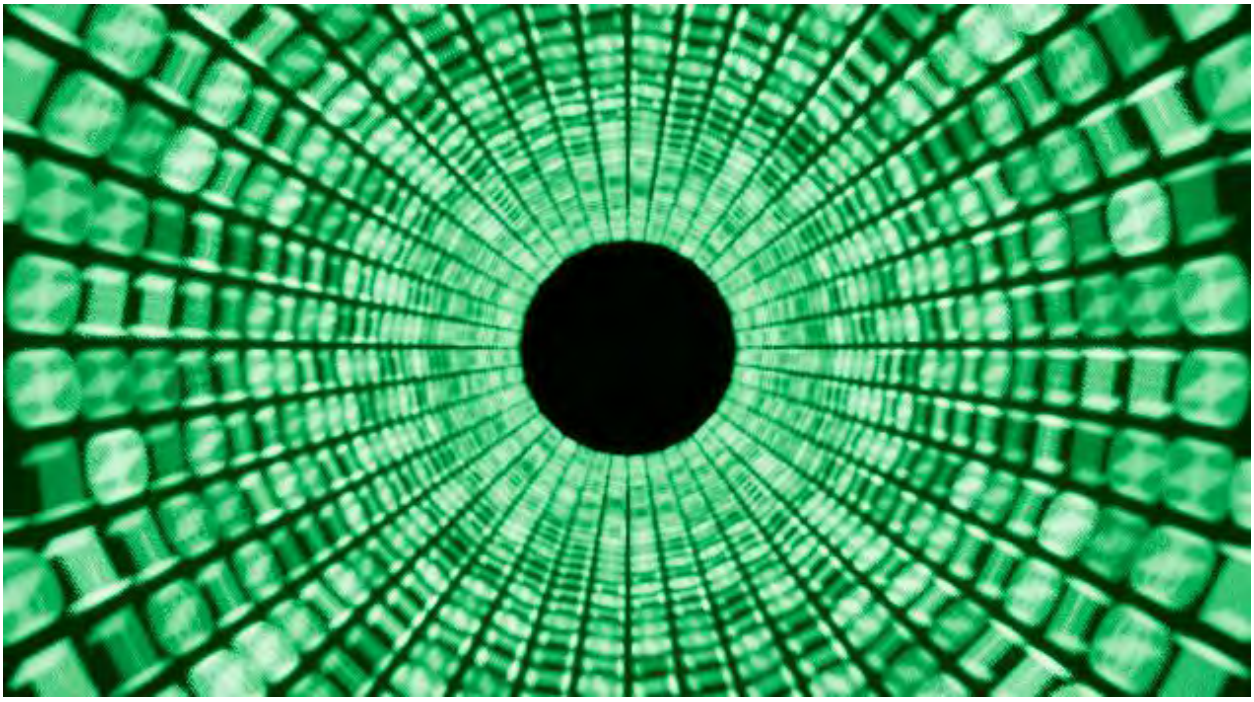
不过这样做的话，会让 Float 数值分别填充到 Vector 的 X,Y 和 Z 中。对这个游戏来说，我们只希望玩家角色沿着 X 轴运动。因此，需要将 Vector 展开为三个 Float 组成部分。

首先确保 AddActorWorldOffset 节点的 Delta Location 端口没有跟其它节点创建关联。右键

单击 Delta Location 端口，然后选择 Split Struct Pin。

最后参考下面的连线方式：

好了，让我们简单回顾下刚才所完成的事情：
1.在游戏的每一帧，都会使用 ForwardSpeed 乘以 Delta Seconds，从而获得独立于帧速的运算结果。
2.AddActorWorldOffset 节点将使用该结果让玩家角色沿着 X 轴运动。
3.因为启用了 Sweep，所以玩家角色只要遇到任何阻碍物体，都会自动停下来。



好了，点击蓝图编辑器上的 *Compile* 按钮。然后返回到主编辑器，点击 *Play* 按钮，就可以在数字世界的黑洞中穿行了~

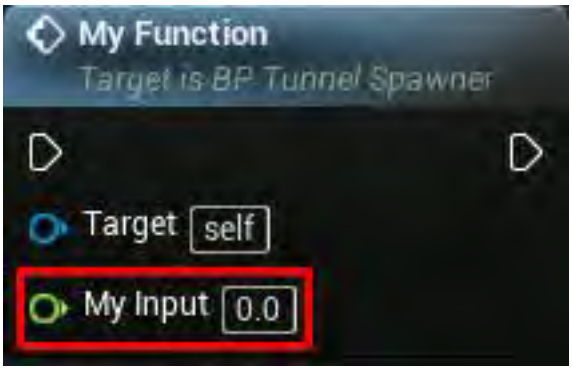
在上一课的内容中，我们实现了玩家角色的持续运动。但是在测试的时候你应该已经发现很快 0 和 1 组成的数字隧道就到了终点，我们希望可以让它持续不断的生成。
在这一课的内容中，我们希望可以让数字隧道自动生成，而不是手动来放置新的数字隧道。

创建隧道生成器

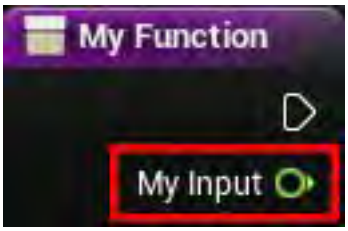
在虚幻 4 的主编辑器中打开 Content Browser，然后进入 Blueprints 文件夹。创建一个新的 Blueprint Class，并选择 Actor 作为 parent class。将其命名为 BP_TunnelSpawner，然后双击在蓝图编辑器中打开。

因为我们希望在游戏中数字隧道可以持续延伸，因此需要创建一个用于生成数字隧道的函数。
在蓝图编辑器中找到 My Blueprint 面板，创建一个新的函数，将其命名为 SpawnTunnel。该函数的作用是在指定位置生成一个数字隧道。

为了将位置传递到该函数，需要设置一个 input parameter（输入参数）。这样当我们调用函

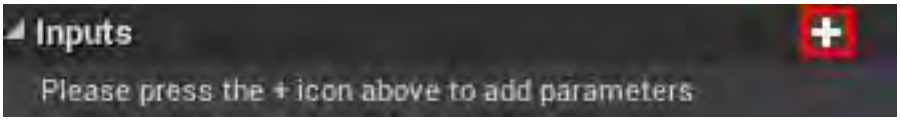


数的时候就会在输入的端口处显示出来。

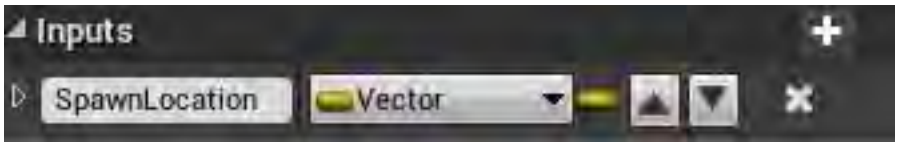


此外，该参数将还在函数的 Entry 节点中作为输出端口显示。

现在让我们创建一个输入参数。首先确保我们在 SpawnTunnel 函数的 Graph 视图中。

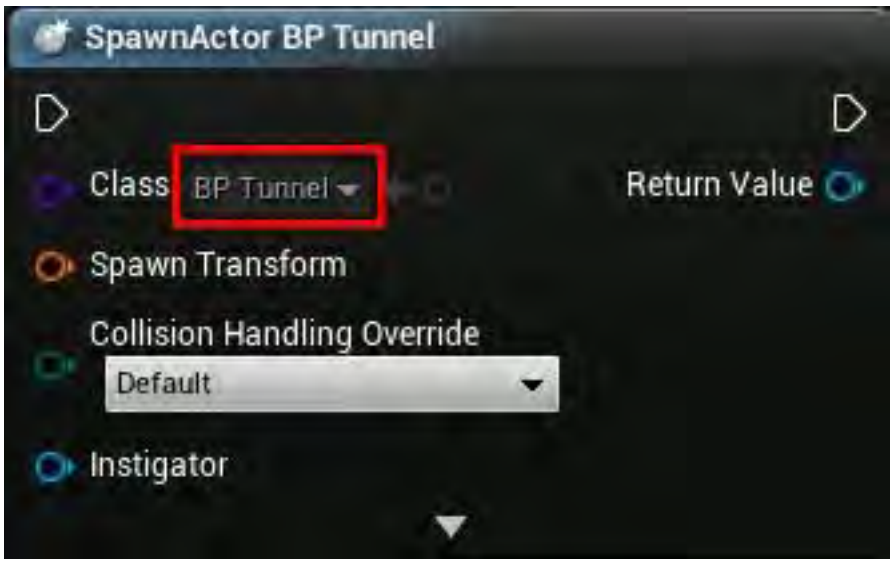


选中 SpawnTunner 节点，然后找到 Details 面板，在 Inputs 部分点击+号。



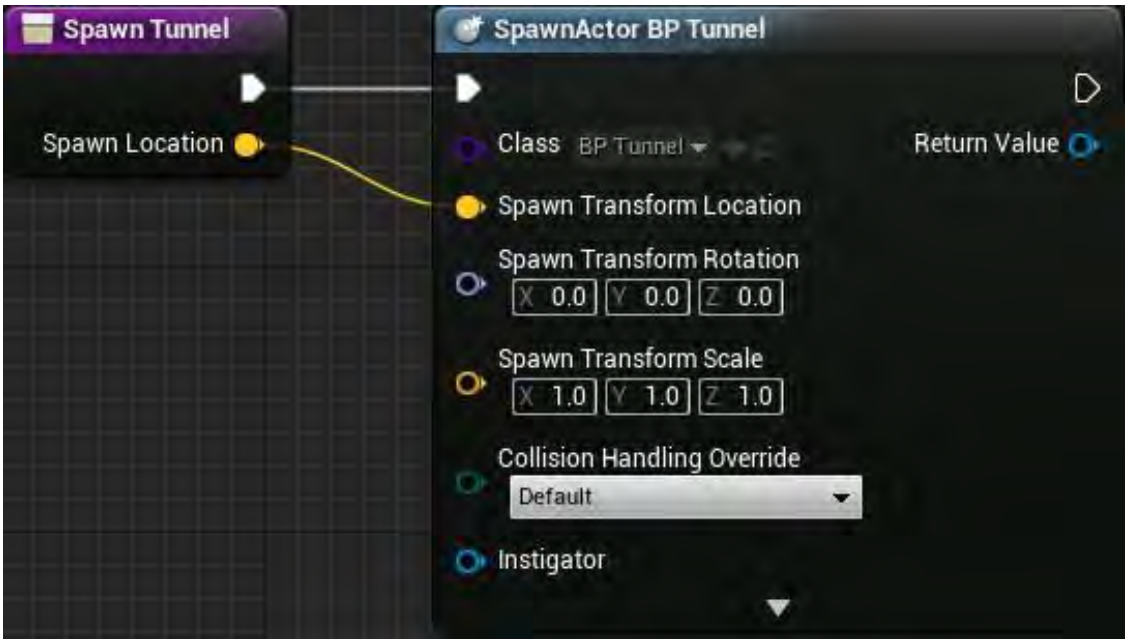
将输入参数更名为 SpawnLocation，然后将类型更改为 Vector。

为了在场景中生成隧道，需要添加一个 Spawn Actor From Class 节点。然后点击 Class 端口



右侧的下拉列表，从中选择 BP_Tunnel。

为了设置生成的位置，让我们右键单击 Spawn Transform 端口，选择 Split Struct Pin。



然后将 Spawn Actor From Class 节点连接到 SpawnTunnel 节点：
好了，现在只要我们调用 SpawnTunnel 函数，就会在指定的位置生成一个 BP_Tunnel 的实例对象。

让我们来测试一下。

测试隧道生成器

切换到 Event Graph，然后找到 Event BeginPlay 节点。在视图中添加一个 SpawnTunnel 节点，然后将其连接到 Event BeginPlay 节点上。

在 SpawnTunnel 节点上，将 Spawn Location 设置为(2000,0,500)。

现在，当游戏开始的时候，它会向上生成一个隧道，并逐渐远离玩家角色。点击蓝图编辑器上的 Compile 按钮，然后返回到主编辑器。
首先我们需要删除关卡中已有的 BP_Tunnel。在 World Outliner 中左键选中 BP_Tunnel，然后按下 Delete 键，或者使用右键-Edit-Delete 删除。

接着在 Content Browser 中找到 BP_TunnerSpawner，然后使用鼠标左键将其拖动到 Viewport 视图中，这样就向关卡中添加了该蓝图的一个实例对象。

此时按下工具栏上的 Play 键，就会看到游戏生成了一个向上的隧道，并且逐渐远离玩家角色。

完成测试后，返回 BP_TunnelSpawner 的蓝图编辑器。将 SpawnTunnel 节点的 Spawn Location 重置为（0，0，0）。

之后点击 Compile 按钮，然后再次返回主编辑器。

在下一课的内容中，我们将设置 BP_Tunnel 的功能。

好了，这一课就到此结束了，休息一下吧~

在上一课的内容中，我们一起创建了一个隧道生成器。
而从这一课开始，我们将一起来学习设置 Tunnel 的蓝图。

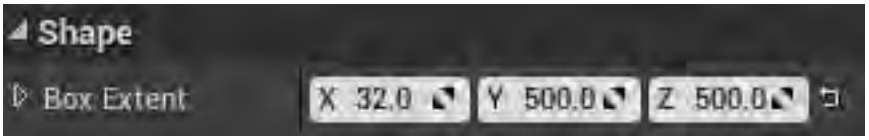
BP_Tunnel 这个蓝图将负责完成两项任务。其中之一是检测和确认何时游戏需要生成一个新的 Tunnel 隧道。为此，我们需要创建一个 trigger 触发区。一旦触发，BP_Tunnel 将通知 BP_TunnelSpawner 生成一个新的隧道。通过这样，就可以制造一种假象，让玩家觉得隧道是

无穷无尽的。
而另一项任务则是确定一个生成点。这样 BP_TunnelSpawner 将使用这个点作为隧道的下一个生成位置。

好了，首先让我们来创建触发区。

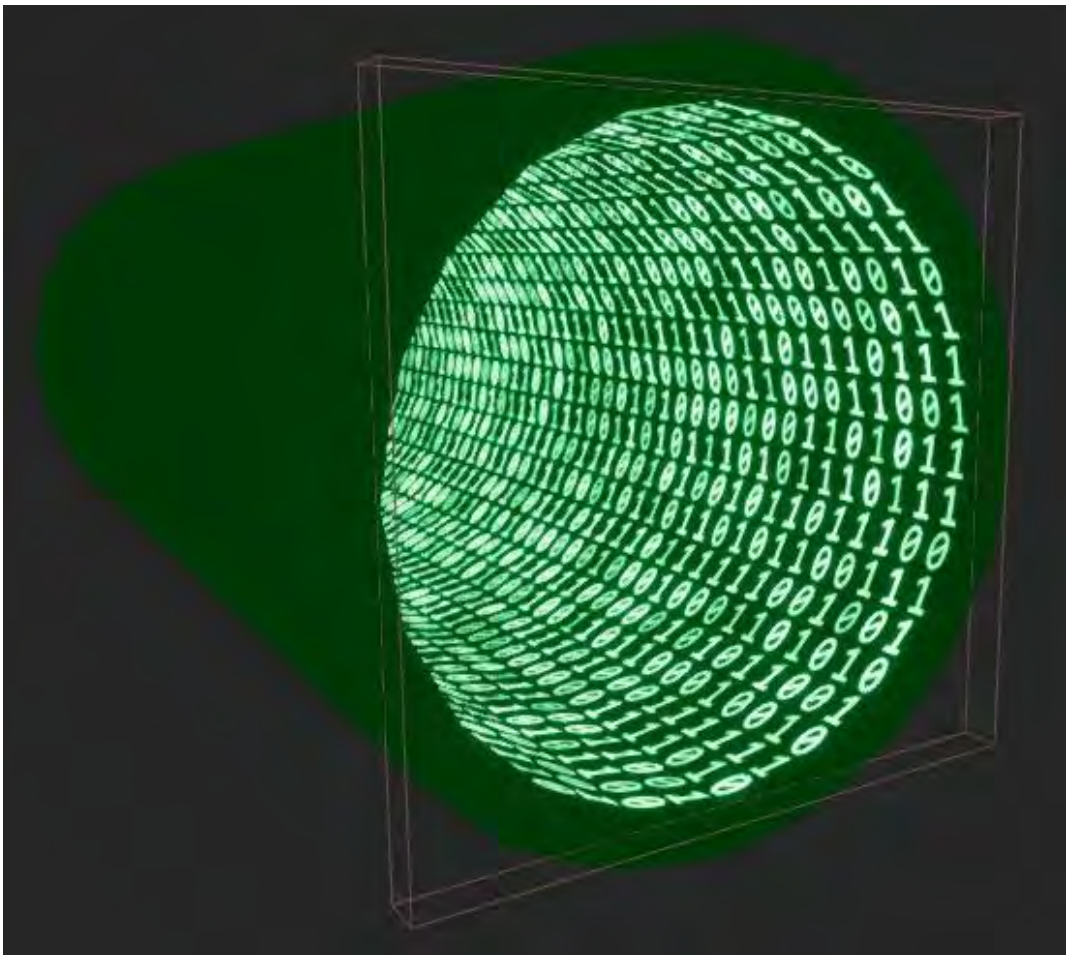
从主编辑器中打开 BP_Tunnel 蓝图文件，然后找到 Components 面板。添加一个 Box Collision 组件，并将其命名为 TriggerZone。

默认的碰撞区很小。为此，我们需要在 Details 面板中找到 Shape 部分。将 Box Extent 的属



性更改为(32,500,500）。

接下来，将 Location 的属性值设置为（2532，0，0）。这样 TriggerZone 就会被放置在隧道纹理的最末端。也就是说只有当玩家达到前一个隧道的末端时才会生成一个新的隧道。



完成这一步之后，接下来就是创建生成点了。

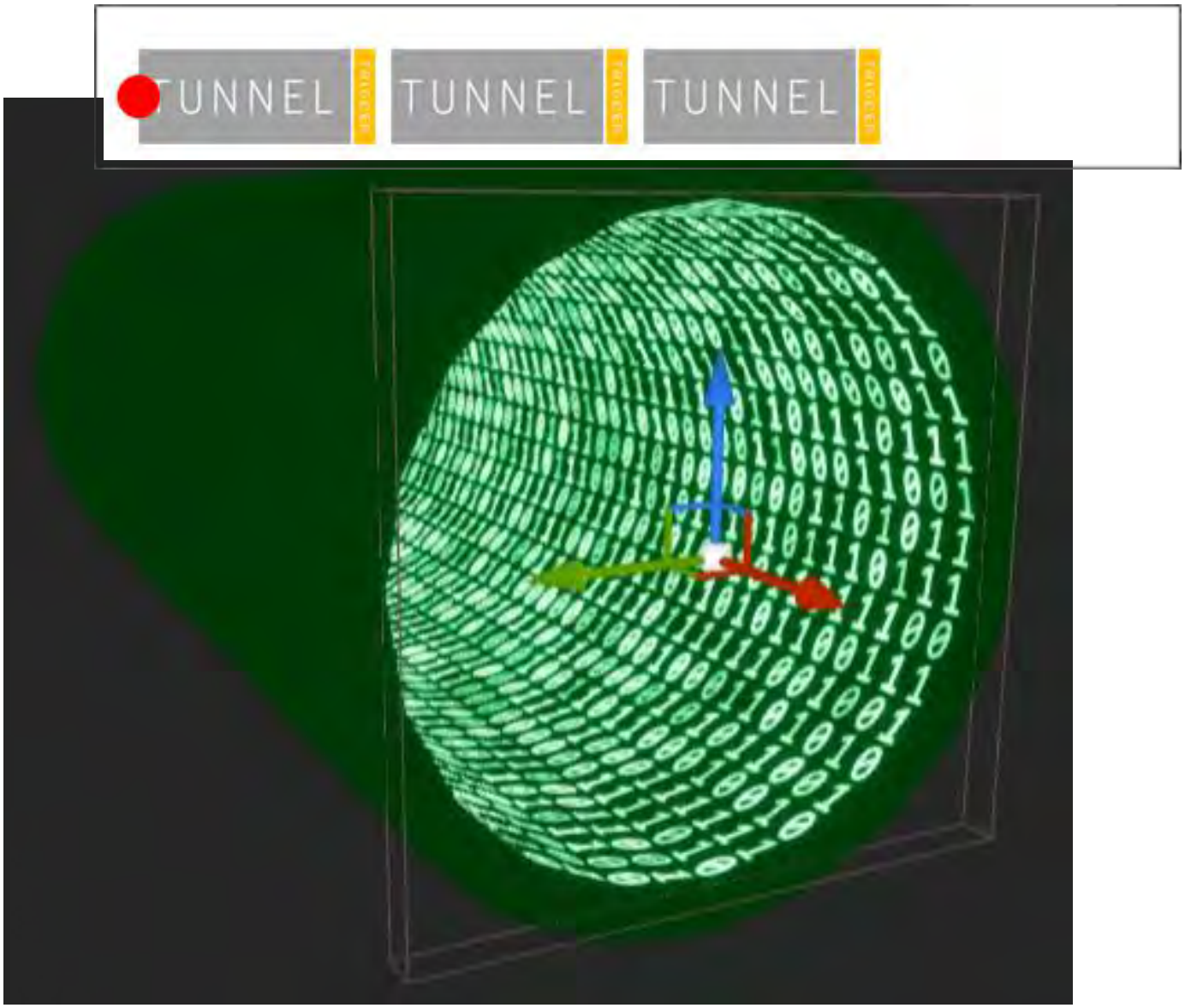
创建生成点

为了定义生成点的位置，我们需要使用 *Scene* 这个 *component*。因为它只包含一个 *Transform* 属性，所以用 *Scene* 组件来定义位置是最合适的了。此外，这类组件在 *Viewport* 游戏视图中也是可见的。这样我们就知道生成点的具体位置在哪里。

在蓝图编辑器中找到 *Components* 面板，确保没有选中任何东西。点击 *Add Component*，从

而添加一个 Scene 类型的组件，并将其命名为 SpawnPoint。

隧道纹理在 X 轴上的长度是 2500 单位，所以我们这也是连接点应该所处的位置。
在 Details 面板中将 Location 属性更改为(2500,0,0）。



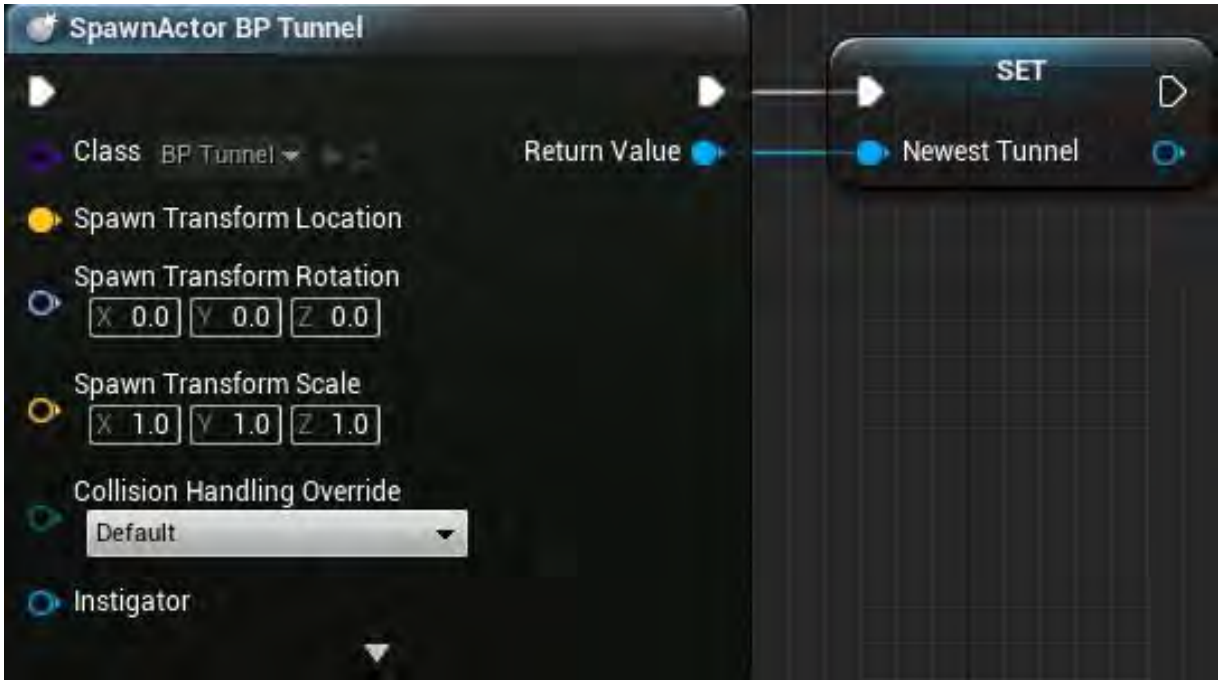
接下来我们需要创建一个函数，从而在 SpawnPoint 这个位置生成隧道。

在 SpawnPoint 生成隧道

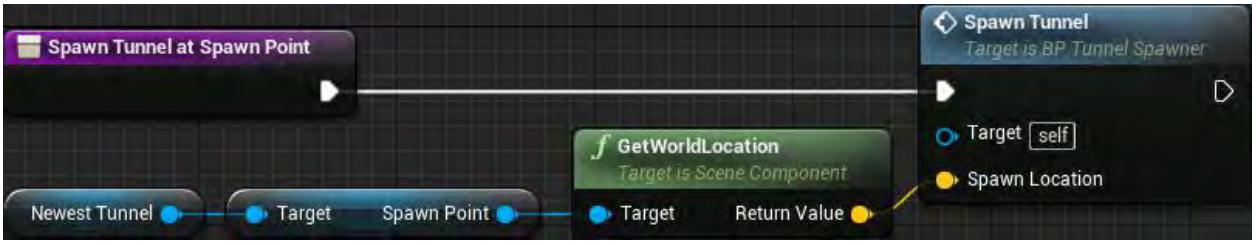
在 BP_Tunnel 的蓝图编辑器工具栏上点击 Compile，然后切换到 BP_TunnelSpawner。
为了让玩家在视觉上感觉到隧道是没有止境的，我们需要让下一个 BP_Tunnel 隧道出生在最远的隧道的 SpawnPoint 点上。

因为最远的隧道永远是最后一个生成的隧道，所以我们很容易就获取到它的引用。

在 BP_TunnelSpawner 的蓝图编辑器中切换到 Spawn Tunnel 这个函数的视图。
右键点击 Spawn Actor From Class 节点的 Return Value 端口。选择 Promote to Variable,然后将变量更名为 NewestTunnel。



这样一来，我们就始终可以有到最远的隧道的引用。



接下来创建一个新的函数，将其命名为 SpawnTunnelAtSpawnPoint，并创建如下的连接：

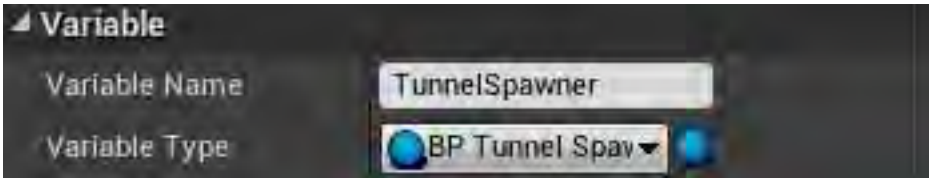
通过以上设置，可以获取最新的隧道，及其 SpawnPoint 的位置。接着游戏会在这个位置生成一个新的隧道。

为了让 BP_Tunnel 蓝图和 BP_TunnelSpawner 蓝图之间建立通讯，需要创建一个引用。否则的话 BP_TunnelSpawner 并不知道应该合适生成下一个隧道。

创建到隧道生成器（Tunnel Spawner)的引用

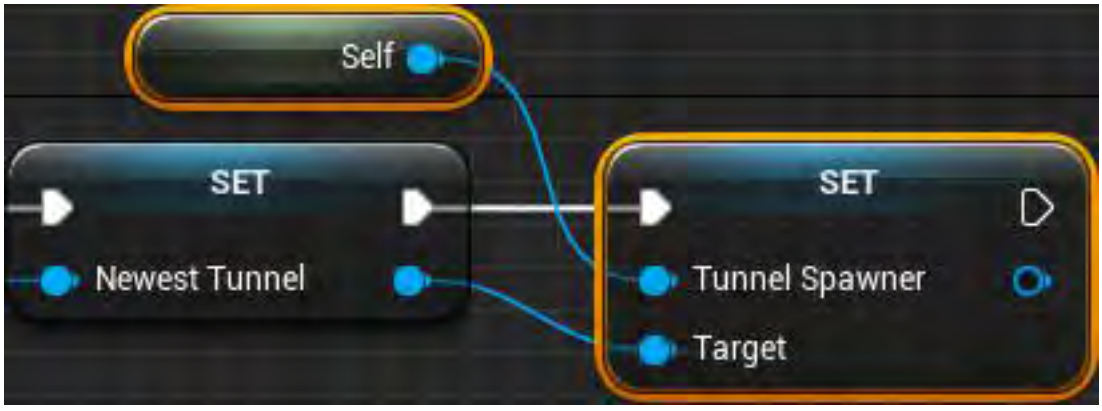
点击工具栏上的 Compile 按钮，然后关闭 SpawnTunnelAtSpawnPoint 这个视图。接着切换到 BP_Tunnel 蓝图。

添加一个新的变量，将其命名为 TunnelSpawner。将变量类型更改为



BP_TunnelSpawner\Object Reference。

点击 Compile 按钮，然后再次切换回 BP_TunnelSpawner 蓝图。



打开 SpawnTunnel 视图，并添加以下节点：

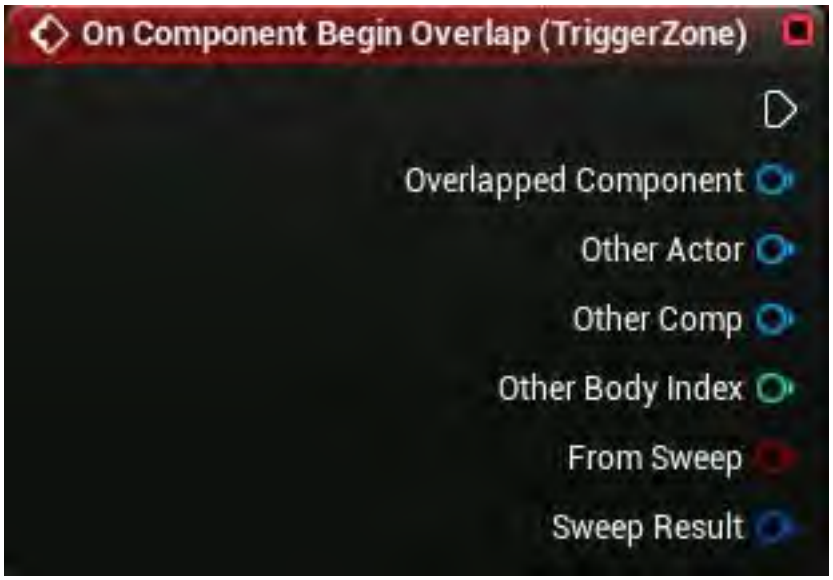
好了，现在每个隧道都有了到 BP_TunnelSpawner 的引用。

接下来，我们需要通知 BP_TunnelSpawner，当玩家进入 TriggerZone 区的时候生成下一个隧道。

添加 TriggerZone 的相关代码

在 BP_TunnelSpawner 的蓝图编辑器上点击工具栏上的 Compile 按钮，然后切换回 BP_Tunnel。

在 Components 面板中右键单击 TriggerZone。选择 Add Event\AddOnComponentBeginOverlap。这样会在 Event Graph 中添加以下节点：

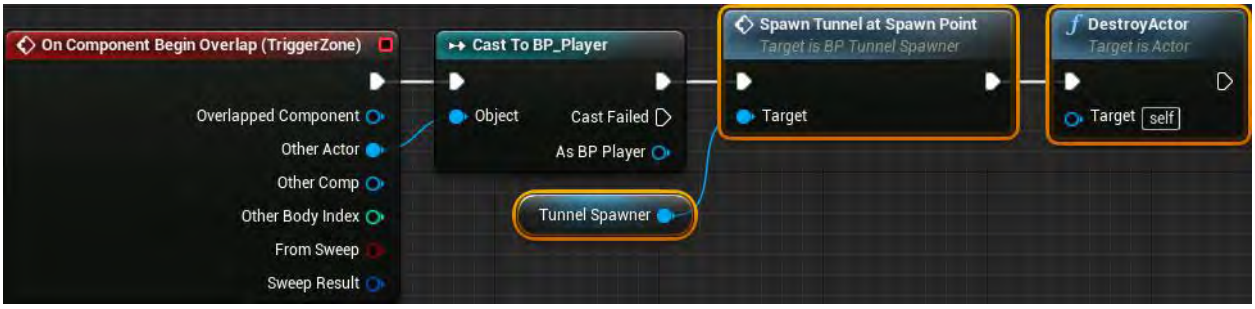


每当有另外一个角色重叠到 TriggerZone 上时，就会执行该节点。

首先，我们需要检查重叠到 TriggerZone 的角色是否是玩家。

从 Other Actor 端口拖出一条线，从菜单中选择 Cast to BP_Player。

注意：因为所生成的隧道都是在之前隧道的末端，所以就会触发隧道的 TriggerZone。Cast to BP_Player 可以防止其它的节点执行。



接下来，在 Cast to BP_Player 节点之后添加相关的节点。

让我们解释下所发生的一切：

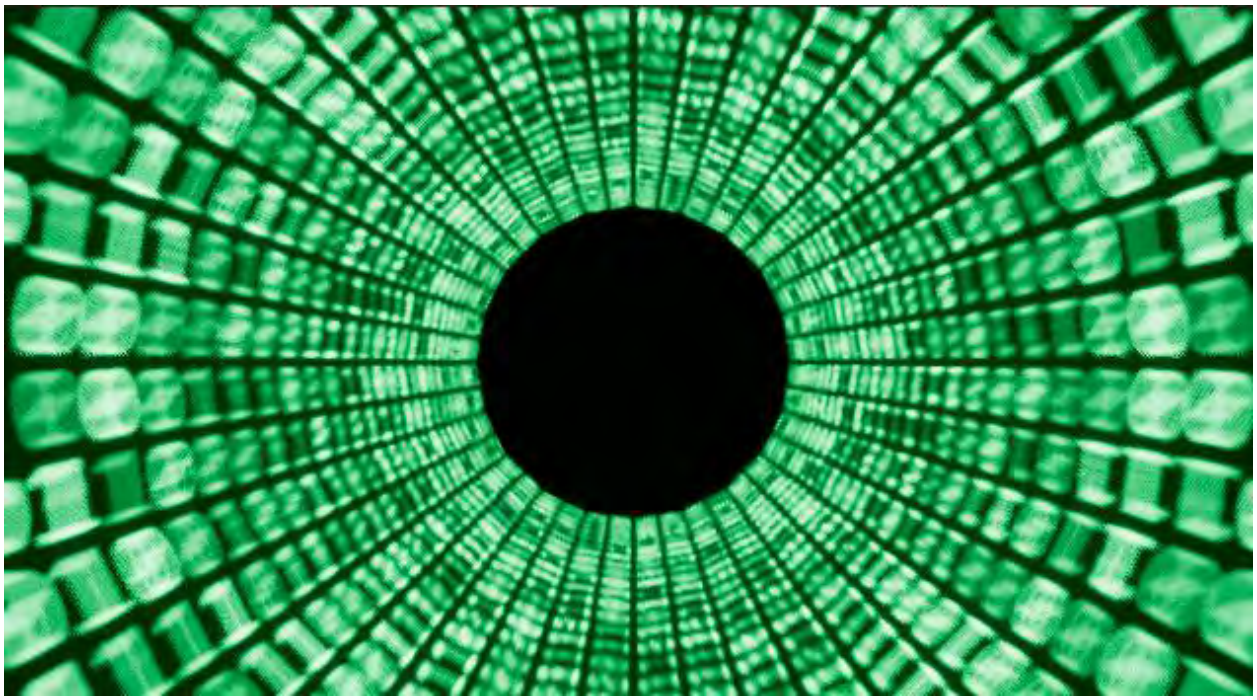
1.当某个角色重叠到 TriggerZone 区时，就会执行 On Component Begin Overlap (TriggerZone)节点。

2.Cast to BP_Player 节点会检查该角色是否是玩家角色

3.如果是玩家角色，BP_TunnelSpawner 就会生成一个新的隧道。它的位置就在最后一个生成的隧道的 SpawnPoint 组件的位置处。

4.既然原有的隧道已经没有作用了，游戏就会使用 DestroyActor 节点来删除之前的隧道。

点击 Compile 按钮，然后返回到主编辑器，点击 Play 运行。
一旦到了前一个隧道的末端，游戏就会自动生成一个新的隧道。



遗憾的是，虽然游戏在不断生成新的隧道，但是看起来却不是无止尽的。
为此，我们需要始终保持几个隧道课件。随后，当我们在添加一个障碍物时，玩家就不会感觉到这些了。

好了，本课的内容就到此结束了，我们下一课再见~

欢迎继续我们的学习。在上一课的内容中，我们学习了如何持续生成隧道。在这一课的内容中，我们将学习如何添加更多的隧道。

添加更多隧道

首先要做的就是创建一个函数，让其生成一定数量的隧道。
打开 *BP_TunnelSpawner*，然后创建一个新的函数，将其命名为 *SpawnInitialTunnels*。

为了生成特定数量的隧道，我们需要使用 *ForLoop* 节点。该节点可以让所连接的节点运行指定的次数。为此，让我们添加一个 *ForLoop* 节点，然后将其连接到 *Spawn Initial Tunnels* 的入口节点。

为了让 *ForLoop* 节点执行 *n* 次，我们需要将 *Last Index* 设置为 *n-1*。
对本教程而言，我们需要生成三个隧道。为了执行三次循环，需要将 *Last Index* 的值设置为 *2*。

注意：
如果没有手动设置 *First Index* 或 *Last Index*，那么默认值是 *0*。

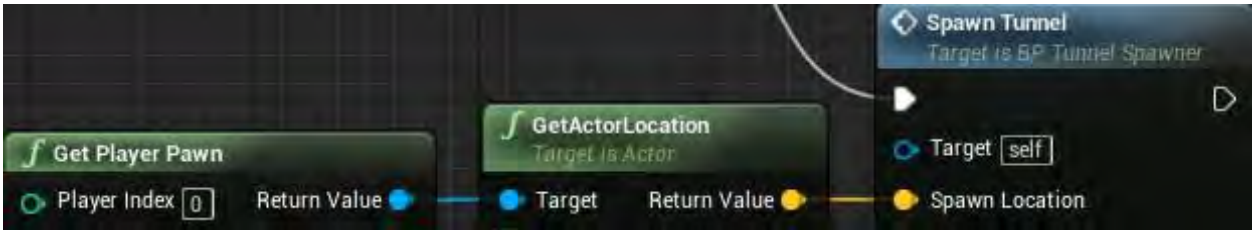
当游戏开始的时候，玩家角色将始终在某个隧道前开始。为此，我们需要将首个隧道的位置设置为玩家角色的位置。

生成首个隧道

为了判断首个隧道是否已经生成，我们需要检查 *NewestTunnel* 是否已设置。如果还没有，那么首个隧道就还没有生成。这是因为只有当游戏生成了一个隧道后，才会设置 *NewestTunnel*。
为了方便检查，需要在 *ForLoope* 节点之后添加一个 *IsValid* 节点。

接下来，需要获取到 *NewestTunnel* 的引用，并将其连接到 *IsValid* 节点的 *Input Object* 端口。

如果 *NewestTunnel* 还没有被设置，那么 *Is Not Valid* 端口之后的节点就会执行，否则就不会。

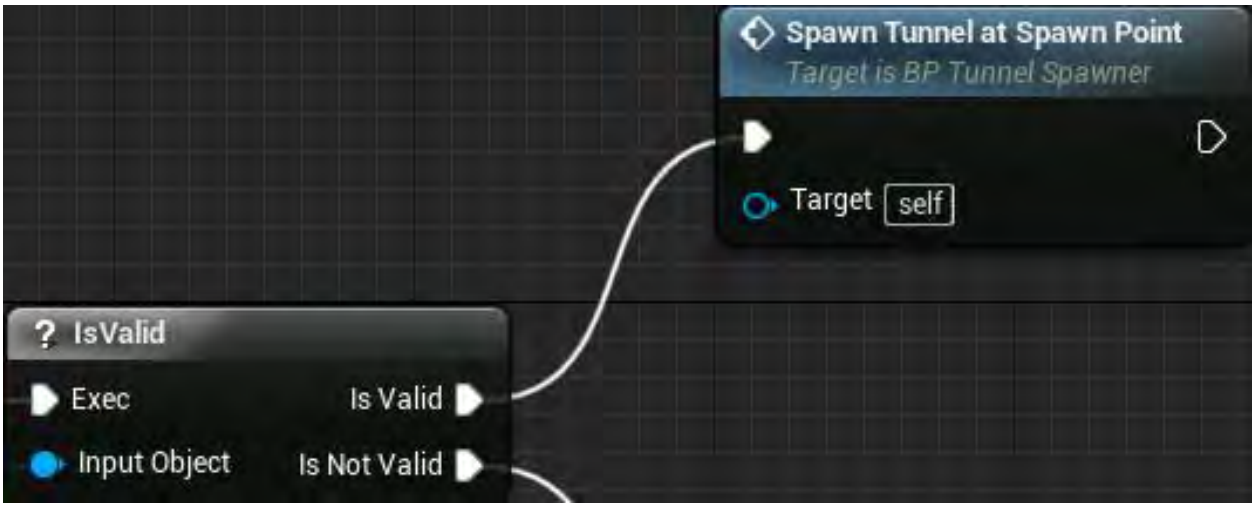


添加以下节点，并将其连接到 *IsValid* 节点的 *Is Not Valid* 端口：

通过以上的操作，就可以在玩家角色所在的位置生成一个隧道。

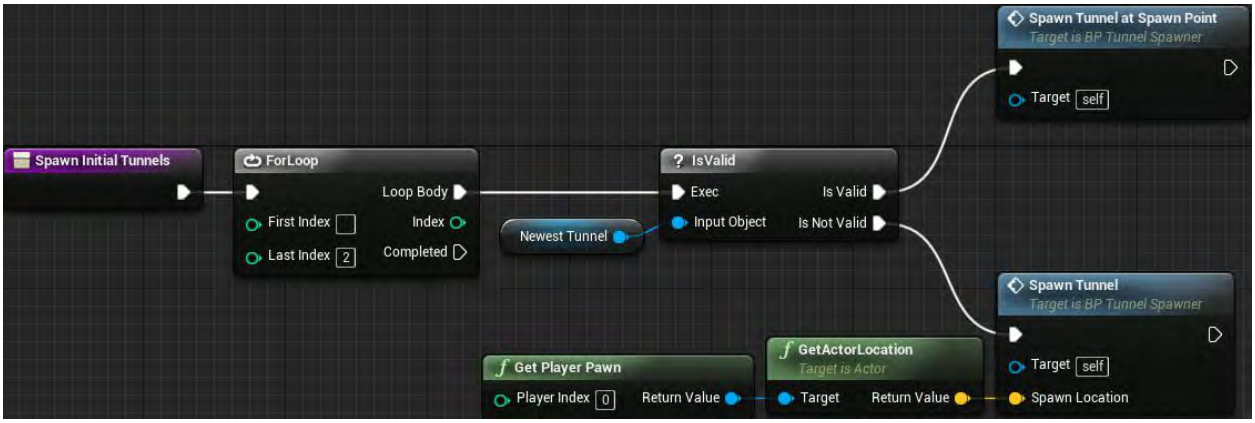
接下来，我们需要生成后续的隧道。

生成后续的隧道



添加一个 *SpawnTunnelAtSpawnPoint* 节点，然后将其连接到 *IsValid* 节点的 *Is Valid* 端口。

至此，我们最终的 *graph* 如下图所示：



总结一下：

1.*ForLoop* 节点将总共执行三次

2.在第一次循环执行时，将在玩家角色的位置生成一个隧道

3.在随后的循环中，将在最新隧道的 *SpawnPoint* 生成隧道。

接下来在蓝图编辑器中切换到 *Event Graph* 视图，然后删除其中的 *SpawnTunnel* 节点。

接着在 *Event BeginPlay* 节点之后添加 *SpawnInitialTunnels* 节点。

现在，当游戏开始的时候，它将自动生成三个隧道。

点击蓝图编辑器上的 *Compile* 按钮，然后返回主编辑器，并按下 *Play* 按钮来测试游戏。现在隧道变得更长了~

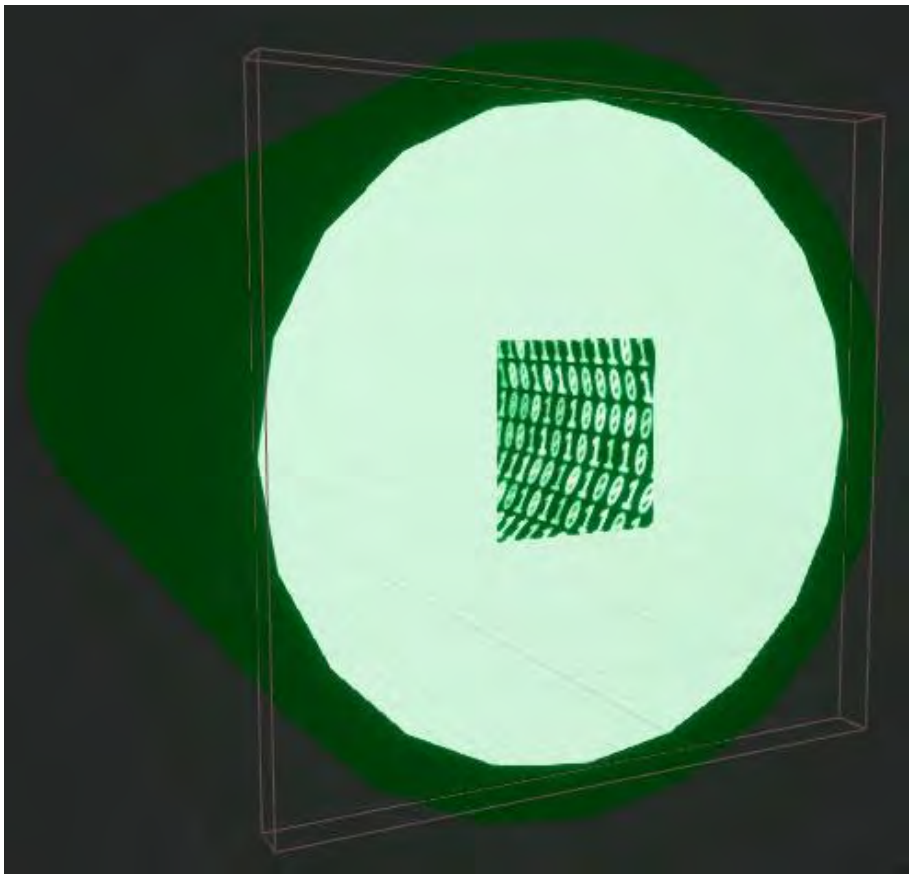
不过到目前为止游戏一点挑战性都没有，在下一课的内容中，我们将学习添加一些障碍物。

好了，这一课就到这里了，休息一下~

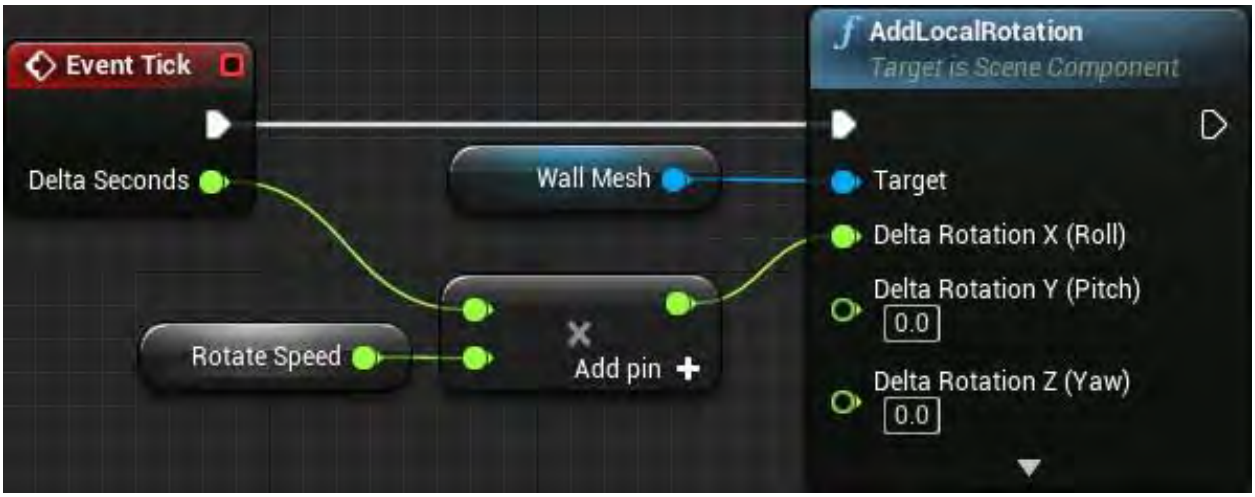
欢迎继续我们的学习。
在这一课的内容中，我们将添加一些障碍物，让游戏增加点挑战性~

下面是我们用作障碍物的纹理贴图：

打开 BP_Tunnel，在 Components 面板处添加一个 Static Mesh 组件，并将其命名为 WallMesh。
接下来在 Details 面板中将 Static Mesh 的属性更改为 SM_Hole_01。
然后将 Location 属性设置为（2470，0，0）。这样就会让障碍物出现在隧道的末端。

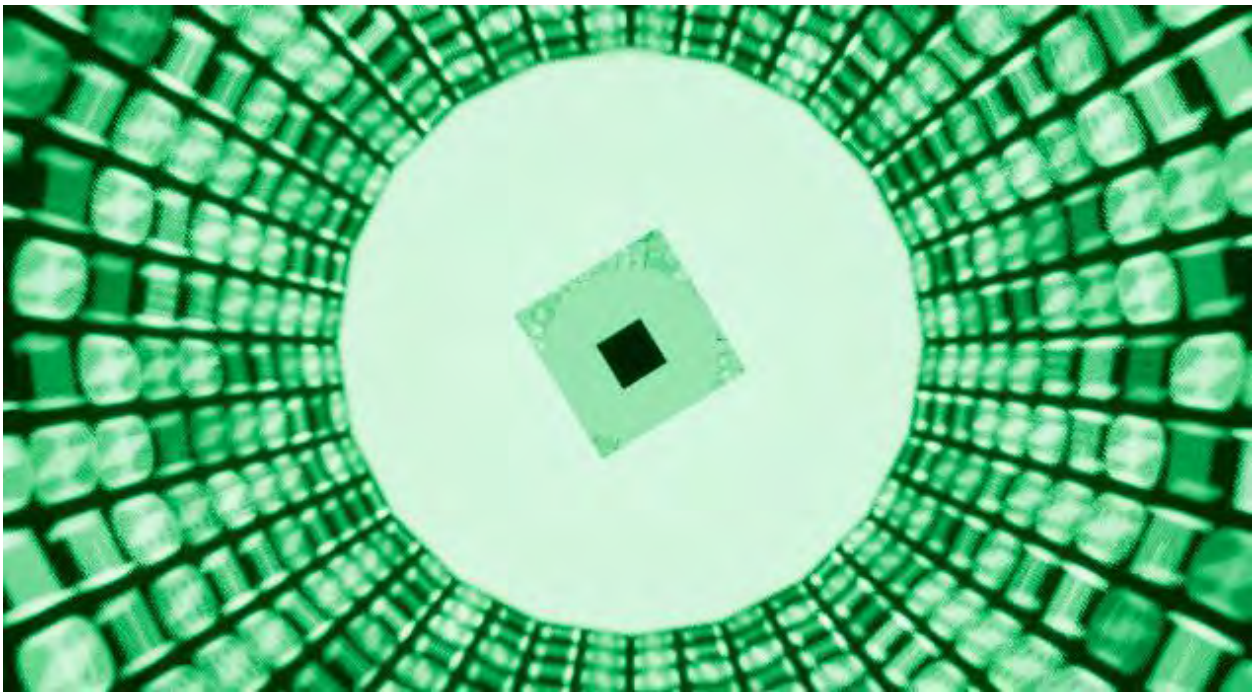


为了让游戏变得更加有趣一点，作为障碍物的墙最好可以旋转。
在 My Blueprint 面板处添加一个新的 Float 类型的变量，并将其命名为 RotateSpeed。将 Default Value 设置为 30。



接下来切换到 *Event Graph* 视图，找到 *Event Tick* 节点，并创建以下的设置：

这样一来，就可以让 *WallMesh* 每帧旋转指定的次数。

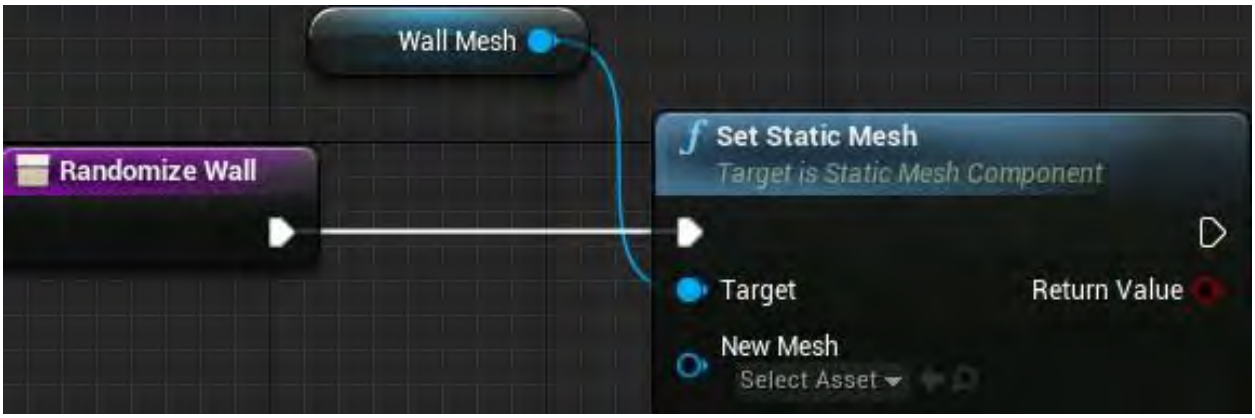


点击蓝图编辑器上的 *Compile* 按钮，然后返回到主编辑器。按下 *Play* 按钮来查看旋转的墙壁。

接下来让我们给这些墙加点花样。

创建不一样的墙

说到创建不一样的墙，或许你第一时间想到的是为每个不同的墙创建一个新的蓝图。其实不然，只需要将当前的 WallMesh 随机化就好。

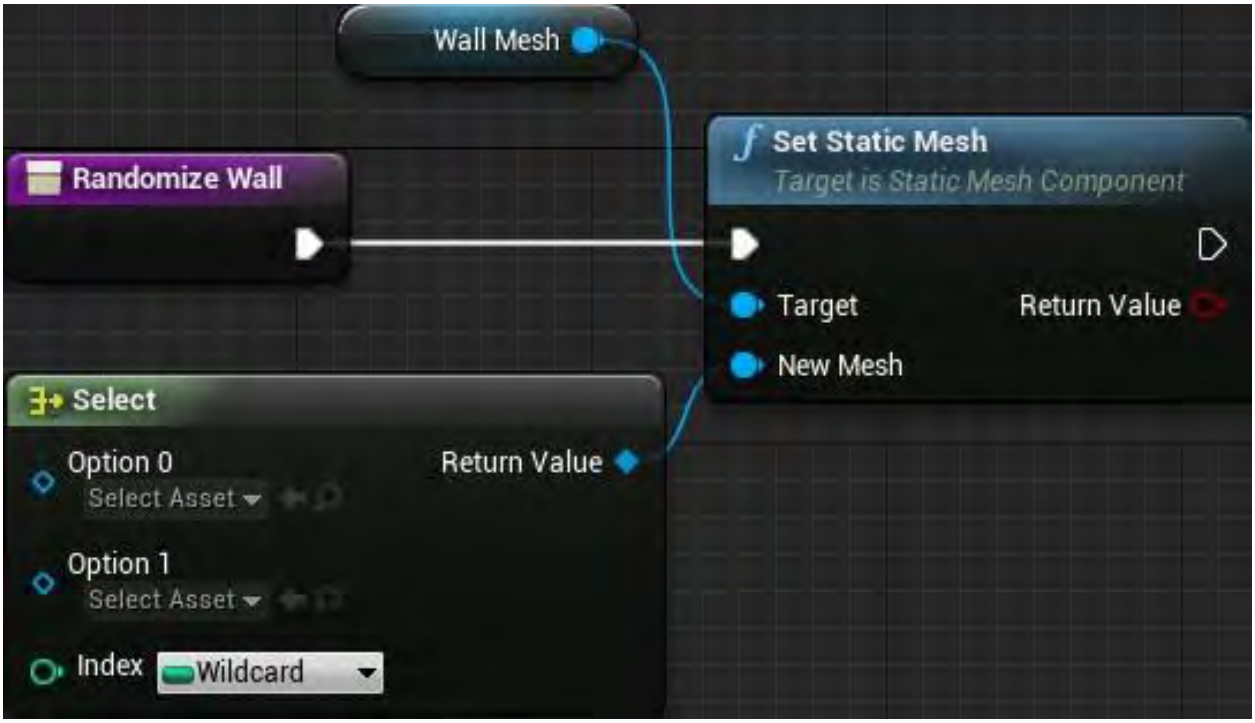


打开 BP_Tunnel，创建一个新的函数，将其命名为 RandomizeWall。接下来创建如下的连线：

顾名思义，Set Static Mesh 节点的作用就是将 WallMesh 设置为所提供的 mesh。

为了创建一系列的 static mesh,我们需要用到 Select 节点。

左键从 New Mesh 端口拖出一条线，然后添加一个 Select 节点如下：



使用 Select 节点，可以让我们设置一系列的选项。其中 Index 处的输入值决定了 Select 节点的输出选项。

因为我们现在有四种 wall mesh 可以用，所以需要再创建两个选项端口。只需在 Select 节点上

右键单击，然后选择 Add Option Pin 即可。重复这个操作，直到 Select 节点上出现四种选项。接下来将每个选项分别按照如下方式来设置：

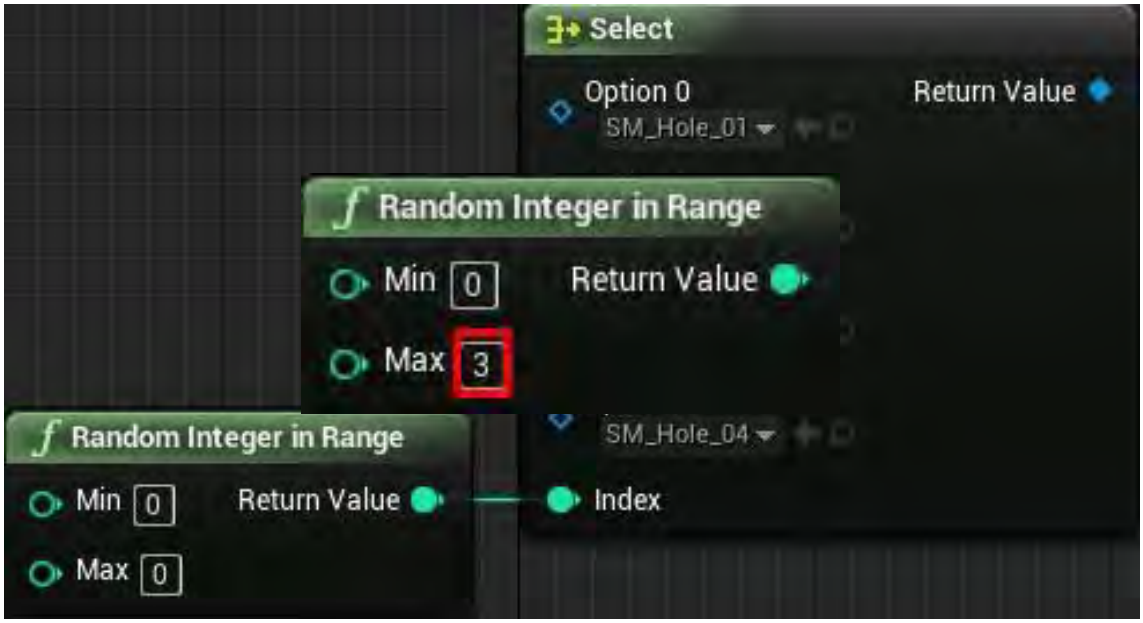
Option 0: SM_Hole_01
Option 1: SM_Hole_02
Option 2: SM_Hole_03
Option 3: SM_Hole_04

好了，现在可以让我们选择其中的一种随机选项。

随机生成墙

我们可以使用 *Random Integer in Range* 节点来获取一个随机数，该节点将返回在 *Min* 和 *Max* 之间的一个随机数。

在视图中添加一个 *Random Integer in Range* 节点，然后将其连接到 *Select* 节点的 *Index* 端口。



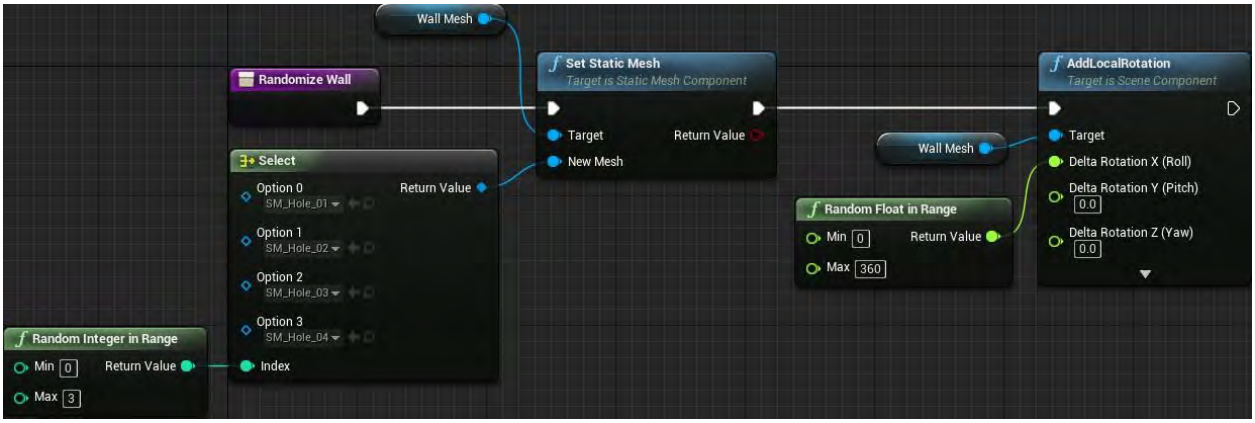
将 *Max* 的数值设置为 3，这样我们就得到了四种可能的数字：*0*，*1*，*2* 和 3。

为了让墙的显示更加随机，让我们再给 *WallMesh* 添加一个随机选中。

在 *Set Static Mesh* 节点之后添加以下节点：

通过这样的方式，我们给 *WallMesh* 添加了 *0* 到 *360* 度的随机旋转角度。

好了，现在来看看最近的连线图：



简单总结下我们已经完成的工作：

1.Select 节点用于选择不同的 mesh 纹理

2.使用 Random Integer in Range 节点来选择一个随机的 mesh

3.使用 Set Static Mesh 节点将 WallMesh 设置为所选择的 mesh

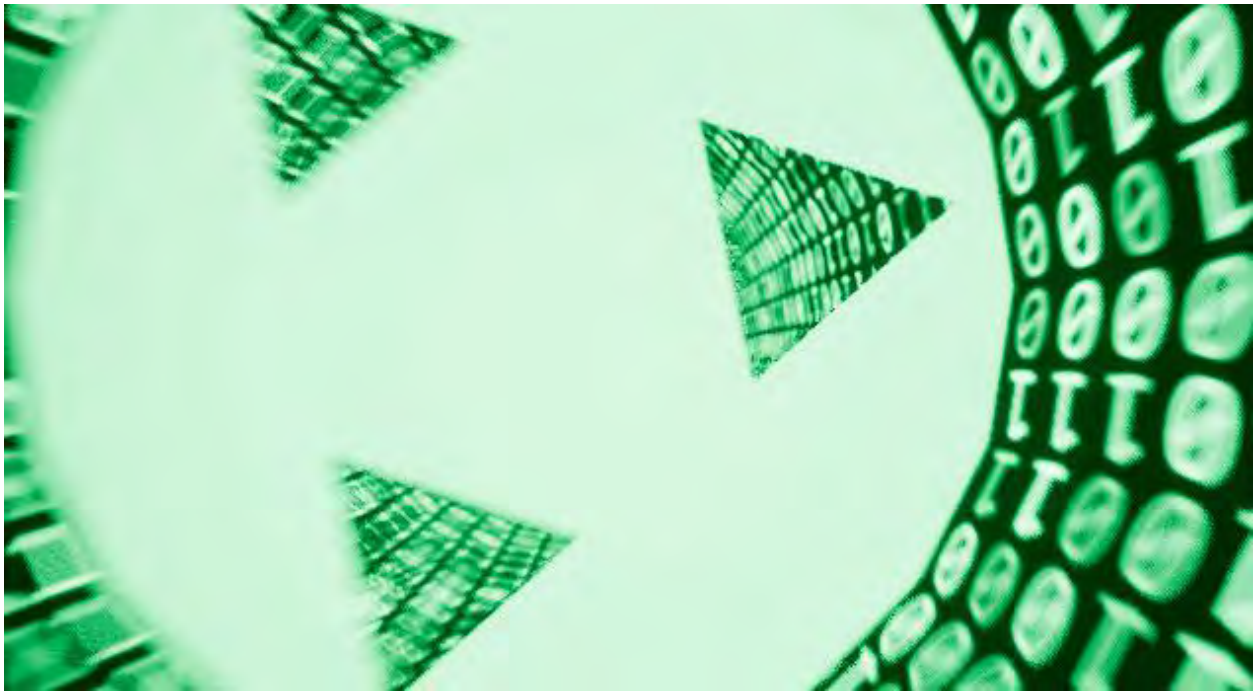
4.使用 AddLocalRotation 节点给 WallMesh 添加一个随机的旋转偏移

点击蓝图编辑器工具栏上的 Compile 按钮，然后关闭 RandomizeWall 视图。

打开 BP_TunnelSpawner，切换到 SpawnTunnel 视图，添加下图中用黄色标出的节点：

好了，现在当隧道生成的时候，会显示一个随机的墙的纹理。

点击蓝图编辑器上的 Compile 按钮。然后返回到主编辑器，并按下 Play 按钮来预览游戏的效果！



当我们碰到某个墙的时候，就会停止向前运动。不过当我们左右移动并穿过某个洞时，就会再次向前运动。

在下一课的内容中，我们需要实现以下的效果。当玩家碰到某个墙的时候，停止向前运动。

好了，本课的内容就到这里了，我们下一课再见~

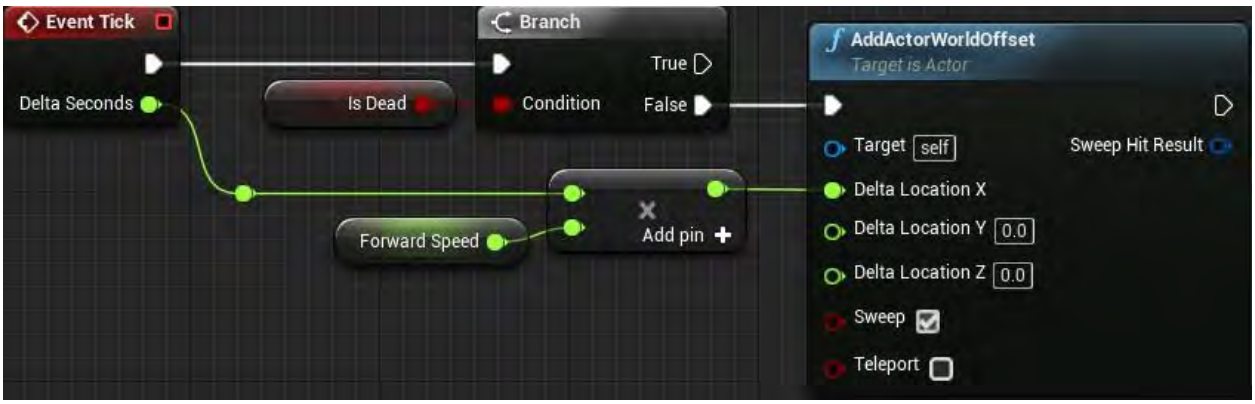
欢迎继续我们的学习。
我们已经可以在场景中随机生成作为障碍物的墙壁，接下来就是添加实际的碰撞机制，让玩家角色在碰到墙之后停止向前运动。

处理墙壁的碰撞机制

为了允许或禁止向前的运动，我们可以使用一个 Boolean 布尔类型的变量。布尔类型的变量只有两种状态：true 和 false，也就是“是”，或者“不是”。

打开 BP_Player 蓝图，创建一个新的 Boolean 变量，将其命名为 IsDead。

接下来，找到 Event Tick 节点，创建一个 Branch 节点。
接着获取到 IsDead 的引用，并将其连接到 Branch 节点的 Condition 端口。
将 Event Tick 节点连接到 Branch 节点上。随后将 Branch 节点的 False 端口连接到

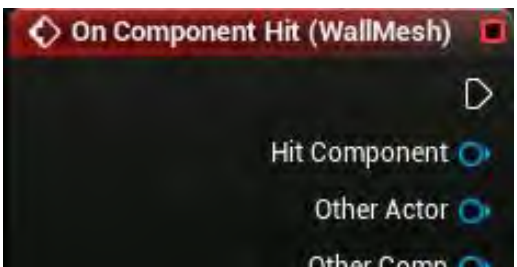


AddActorWorldOffset 节点上。

好了，现在只要 IsDead 被设置为 true，玩家就会停止向前运动。
接下来，当玩家碰到某个墙壁的时候，我们需要设置 IsDead 的布尔值。

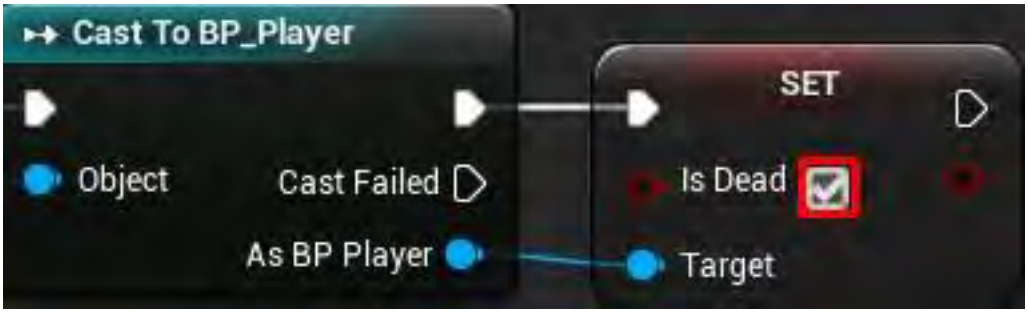
设置 IsDead 变量值

点击 Compile 按钮，然后切换到 BP_Tunnel。在 Components 面板中，右键单击 WallMesh，并选择 Add Event\Add OnComponentHit。这样就可以把以下节点连接到 Event Graph 上。



当另一个 Actor 碰到 WallMesh 时，就会执行该节点。
首先，我们需要检查跟 WallMesh 碰撞的 Actor 是否是玩家角色。

从 Other Actor 端口拖出一条线，在空白处释放，从弹出菜单中选择 Cast to BP_Player。
接着从 Cast to BP_Player 节点的 As BP Player 节点拖出一条线，从菜单中选择 Set Is



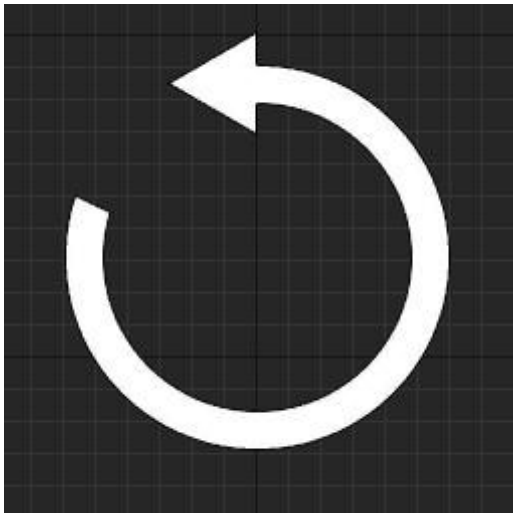
Dead 节点。

在 Set Is Dead 节点中，将 IsDead 后的勾选框选中。

完成后，点击 *Compile* 按钮，然后返回主编辑器。点击工具栏上的 *Play* 按钮，然后试着跟某个墙碰撞。此时当你移向某个洞时，就不能穿过了。

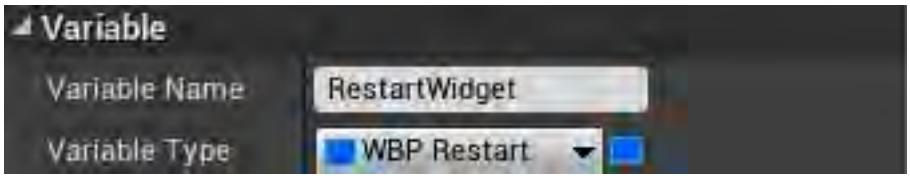
好了，这部分的内容就到此结束了。
在下一部分的内容中，我们将在玩家碰到某个墙的时候显示一个重新再来的按钮~

欢迎继续我们的学习。
从本课开始，我们将学习添加一个重新来过的按钮，它将被命名为 WBP_Restart。在 UI 文件夹

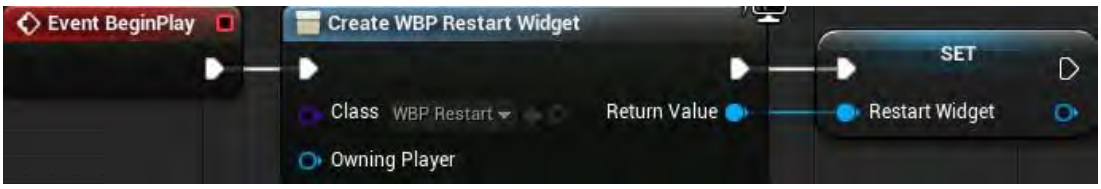


中可以找到这个按钮，如下图所示：

为了显示或隐藏这种 widget，首先需要获取到它的引用。打开 BP_Player，然后创建一个新的

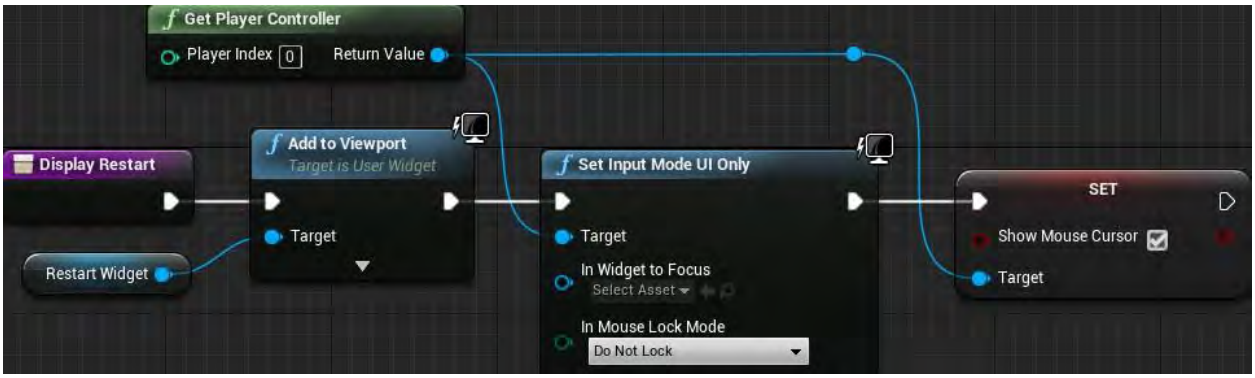


变量名，将其命名为 RestartWidget。将 Variable Type 设置为 WBP .
然后切换到 Event Graph 视图，找到 Event BeginPlay 节点。
如果没有，那么需要手动添加该节点。
接着添加一个 Create Widget 节点，并将 Class 的值设置为 WBP_Restart。



之后，再添加一个 Set Restart Widget 节点，并使用如下的连线方式：
好了，现在当玩家角色生成的时候，就会创建一个 WBP_Restart 的实例。接下来就是设计一个函数，让其显示该实例。

创建显示函数



创建一个新的函数，将其更名为 *DisplayRestart*。完成后参考下图创建连线：

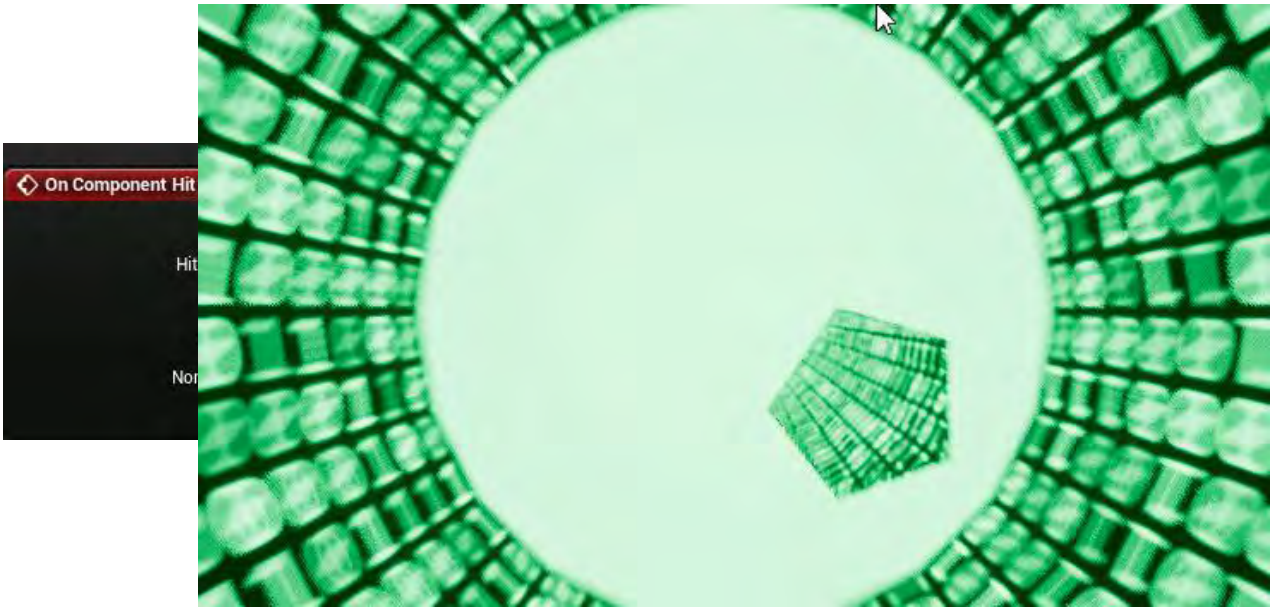
上图的连线都做了些什么呢？
1.Add to Viewport 节点将在屏幕上显示 *RestartWidget*。
2.Set Input Mode UI Only 节点将让玩家角色的互动仅限于跟 *UI* 元素。通过这样的限制，可以让玩家角色死亡后无法移动。
3.Set Show Mouse Cursor 节点的作用就是显示　的光标。

为了显示重新来过的按钮，只需在玩家角色碰到墙壁后调用 *DisplayRestart* 即可。

调用显示函数

关闭 *DisplayRestart* 视图，然后点击工具栏上的 *Compile* 按钮。

切换到 *BP_Tunnel*，然后找到 *On Component Hit(WallMesh)*节点。在节点链条的最后添加 *DisplayRestart* 节点。



现在点击蓝图编辑器上的 *Compile* 按钮，然后关闭 *BP_Tunnel*。返回主编辑器，点击 *Play* 按钮，当玩家角色碰到墙壁的时候，就会显示重启按钮。

最后一步就是当用户点击按钮的时候重新开始游戏。

我们将在下一课的内容中详细讲解，同时也将结束这个小游戏的开发~

好了，这一课的内容就到这里了。

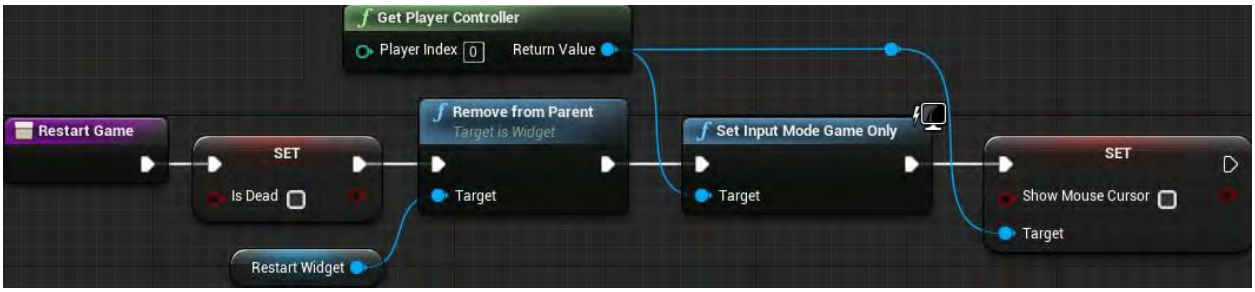
欢迎继续我们的学习。
在这一课的内容中，我们将完成重新开始游戏的功能，并结束这个小游戏的开发~

重新启动游戏

当重新游戏时，我们需要完成两件事情：
1.重置玩家角色
此外，还需要把重新来过的按钮从屏幕中移除。
2.重新生成隧道
只有这样玩家角色才会在隧道的起始端开始游戏。

首先让我们重置玩家角色

重置玩家角色



打开 BP_Player，然后创建一个新的函数，将其命名为 RestartGame。创建如下的连线：

简单解释一下：
1.Set Is Dead 将 IsDead 设置为 false。这样就可以重新启动向前的运动。
2.Remove From Parent 将 RestartWidget 从屏幕中移除。
3.Set Input Mode Game Only 节点将重新启用游戏输入，这样玩家角色就可以四处移动了。
4.Set Show Mouse Cursor 节点会将鼠标光标隐藏

好了，接下来让我们重新设置隧道。

重新生成隧道

点击工具栏上的 Compile 按钮，然后关闭 BP_Player。

打开 BP_TunnelSpawner，并切换到 SpawnInitialTunnels 视图。
在生成新的隧道之前，首先需要移除已经存在的隧道。

在 *Entry* 节点之后添加一个 *Sequence* 节点，将 *Then 1* 端口连接到 *ForLoop* 节点上。

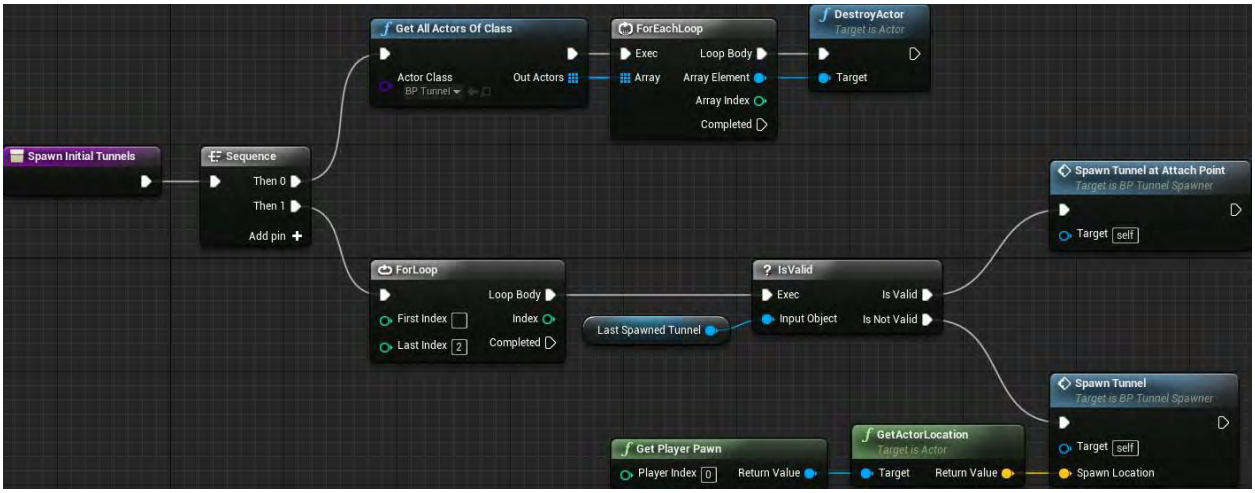
注意：*Sequence* 节点将连续执行其输出。当节点链变得特别长的时候，可以用这种节点让节点图纵向放置。

接下来，创建以下节点：

以上设置将从游戏中移除所有的隧道。

最后，将 *Sequence* 节点的 *Then 0* 端口连接到 *Get All Actors of Class* 节点上，这样就可以在生成之前移除已有的隧道。

最后的连线图如下图所示：



处理按钮点击

点击工具栏上的 Compile 按钮，然后关闭 BP_TunnelSpawner。
回到主编辑器，找到 UI 文件夹，双击打开 WBP_Restart。

在 Hierarchy 面板中找到 RestartButton，然后查看 Details 面板。找到 Events 部分，点击

OnClicked 旁边的按钮。

UE4 会自动帮我们创建一个名为节点，名为 On Clicked(RestartButton)。当玩家角色点击 RestartButton 时，就会执行该节点。

在视图中创建以下连线：

简单解释一下：
1.Get Owning Player Pawn 节点会返回玩家角色当前在控制的 Pawn
2.Cast to BP_Player 节点将检查 Pawn 是否是 BP_Player 类
3.如果是，就会调用 RestartGame 函数。该函数将重置玩家角色，并隐藏重新来过的按钮。

4.Get All Actors of Class 和 Get 节点将返回 BP_TunnelSpawner，然后调用 SpawnInitialTunnels。该函数将移除已有的隧道，并生成新的隧道。

点击 Compile 按钮，然后关闭蓝图编辑器。
点击主编辑器上的 Play 按钮来测试一下重新来过的按钮。

好了，我们的这个小游戏就开发完成了~

项目源代码请参考：
链接:https://pan.baidu.com/s/1QtNAfsz3Lwb5-r3v_fWGng 密码:gapf

这个部分的课程就到此结束了。
从下一课开始，我们将学习蓝图中的动画。

我们下一课再见~

笨猫学编程 QQ 群：375143733
知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。
在上一部分的内容中，我们使用虚幻 4 开发了一个简单的游戏。
在接下来的这部分内容中，我们将了解虚幻 4 中的动画系统。
在任何一款主流的游戏中，都少不了动画的存在。如果没有动画，游戏场景中的角色会缺少灵魂，即便在运动，也会显得非常假。
不过幸运的是，在虚幻 4 引擎中添加角色动画非常的轻松。
在这部分的内容中，我们将学习以下内容：
1.导入带骨骼的角色纹理
2.导入动画
3.创建一个动画蓝图，从而让角色在不同的动画状态之间切换
4.创建混合动画。

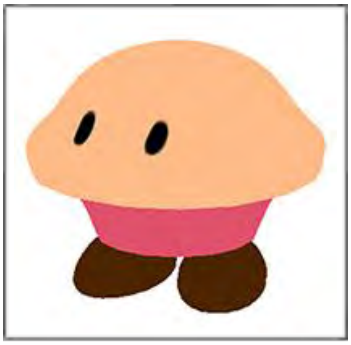
当然，跟之前一样，本系列的教程内容还是基于蓝图系统的。

开始前的准备

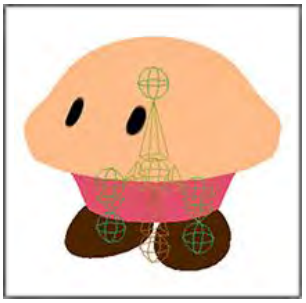
通过这里的链接来下载起始项目(链接:https://pan.baidu.com/s/1b3BScAMZfZhGcP-TBvzrIw 密码:5c0s)。在根目录下，你会看到一个名为 Animation Assets 的文件夹。

该文件夹中包含了我们将要导入的角色和动画。
找到项目文件夹，双击打开 SkywardMuffin.uproject。
点击编辑器工具栏上的 Play 按钮启动游戏。这个小游戏的目标是跳到尽可能多的云朵上，而不掉

下来。点击鼠标左键就可以跳到第一片云朵上面。

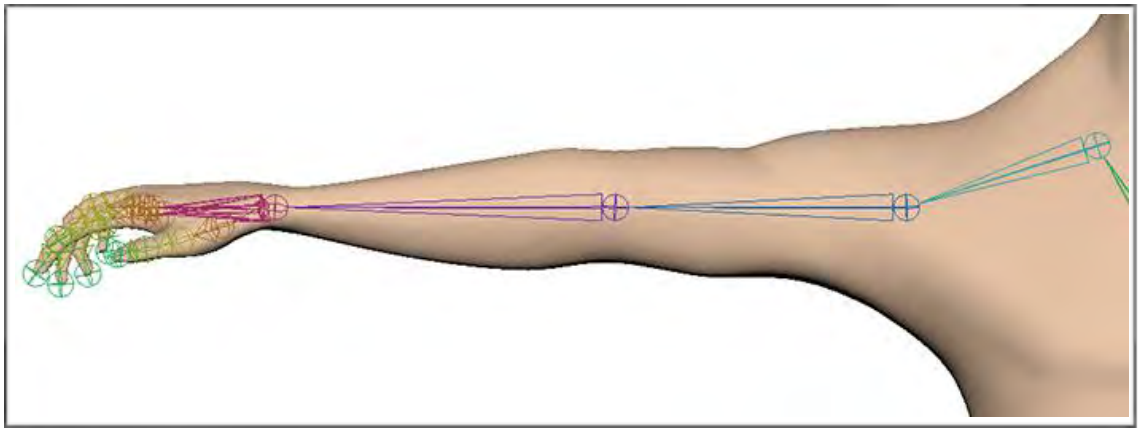


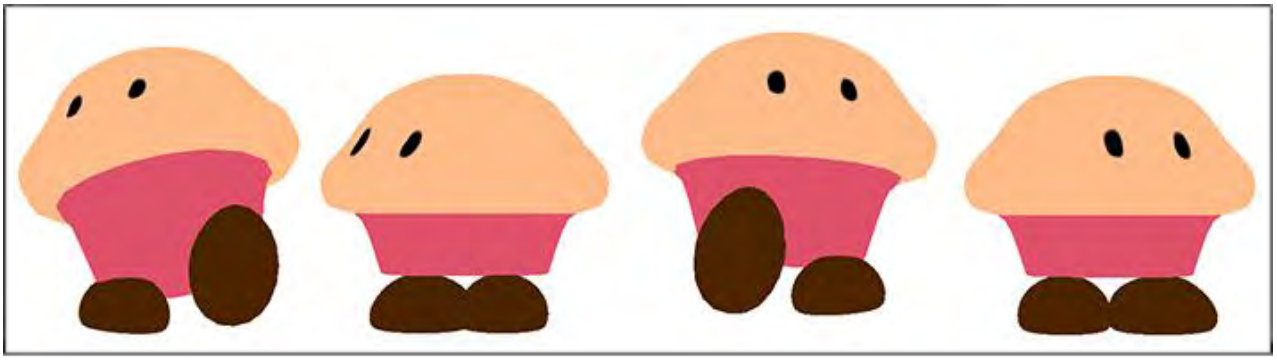
现在游戏中我们控制的是一个最基本的红色圆圈，接下来希望把它替换成下面的这个松饼小人。



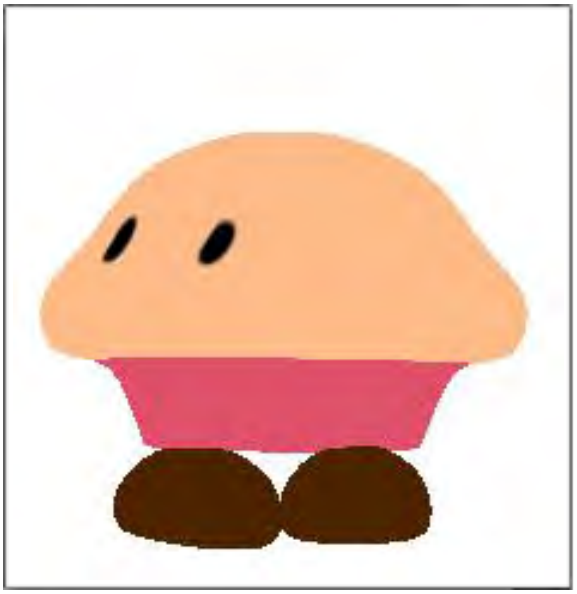
松饼小人包含了骨骼 skeleton，可以使用骨骼让它动起来~

那么问题来了，什么是 skeleton 骨骼呢？





在 3D 的应用中，骨骼就是一系列的相连的关节点（joints）。在下图中，每个小球都是一个关节点：



通过操控这些关节点（或者 bone 骨骼），就可以为角色创建不同的姿态：
当我们从某个姿态切换到另外一种姿态时，就创建了一个动画。
当我们把多个姿态关联在一起时，就可以让动画显得更加自然顺畅~
在虚幻 4 引擎中，任何带骨骼的纹理都是 skeletal mesh。
在下一课的内容中，我们将导入松饼小人的 skeletal mesh（骨骼纹理）。

笨猫学编程 QQ 群：375143733
知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:

http://blog.sina.com.cn/eseedo

Github:

https://github.com/eseedo

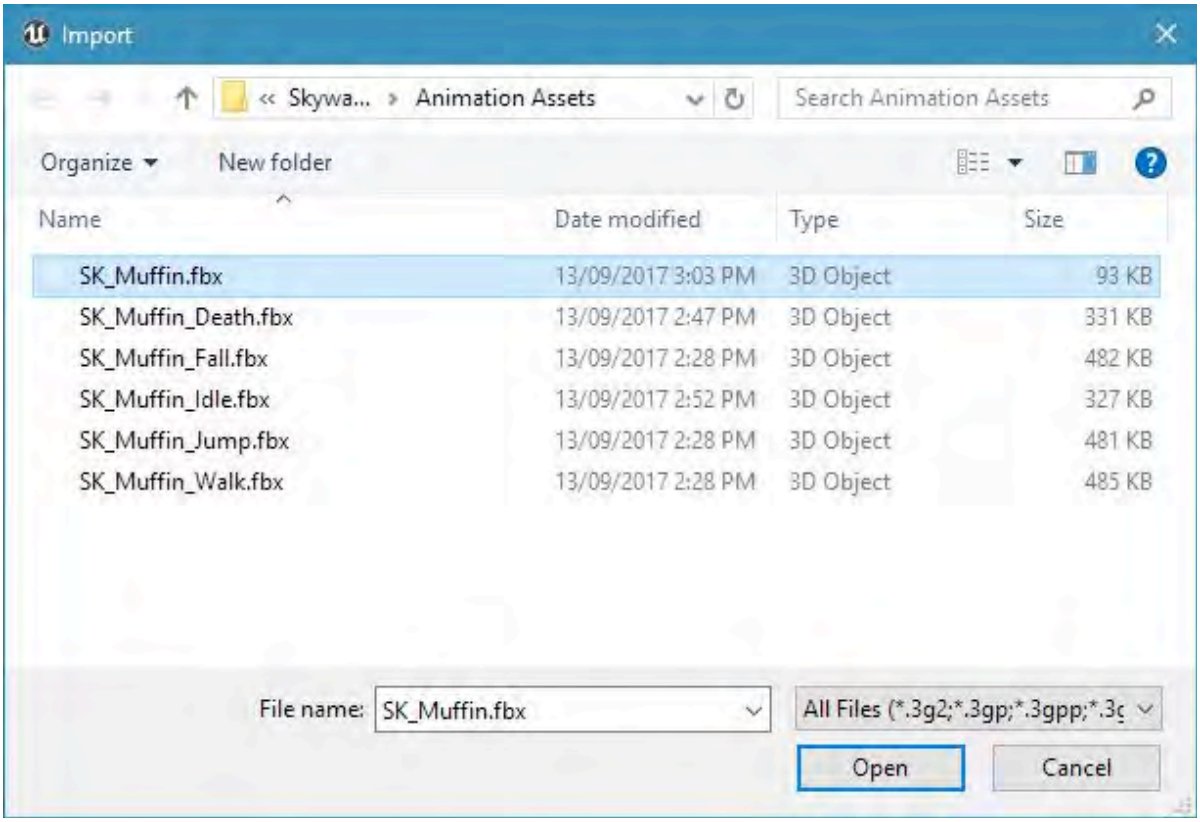
个人网站：

http://icode.ai/

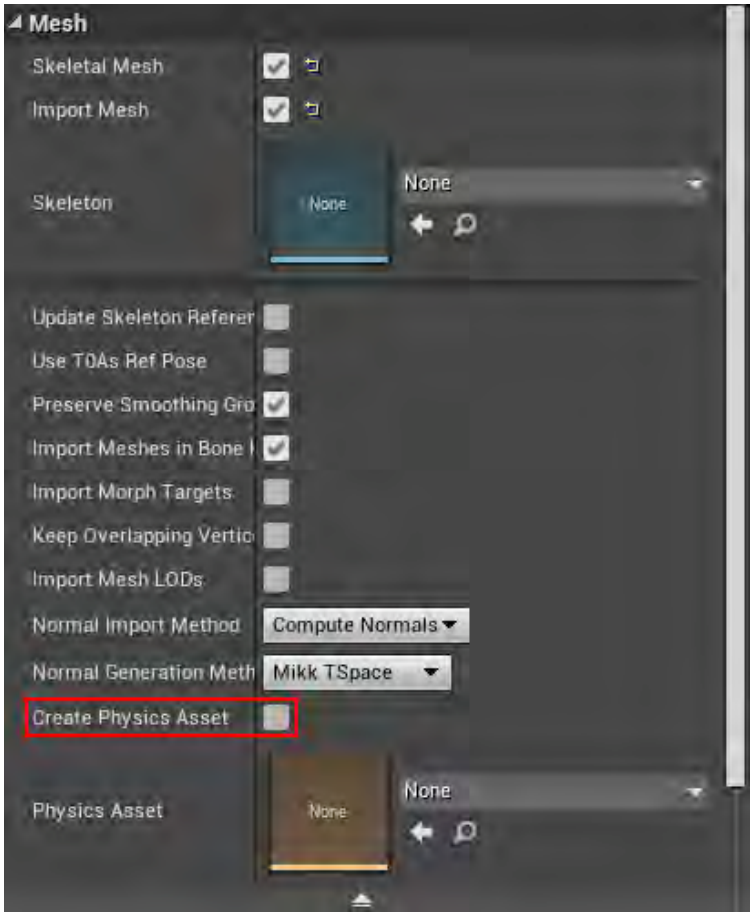
欢迎继续我们的学习。

导入Skeletal Mesh

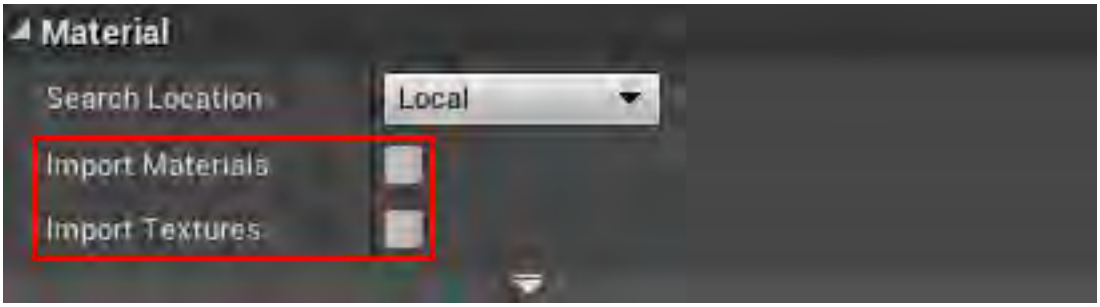
在虚幻4引擎的主编辑器中打开Content Browser，找到Characters\Muffin。点击Import,然后找到SkywardMuffinStarter\Animation Assets。选择SK_Muffin.fbx,然后点击Open。



在导入设置界面，找到Mesh部分，取消勾选Create Physics Asset选项。Physics Asset会帮助创建布娃娃效果。不过这部分教程暂时用不到这个功能，所以要取消勾选。
需要注意的是，该选项是在Mesh的Advanced高级设置中才有。



此外，因为项目中已经提供了muffin的材质和纹理，所以这里也不需要再次导入，取消勾选Import Materials和Import Textures选项。

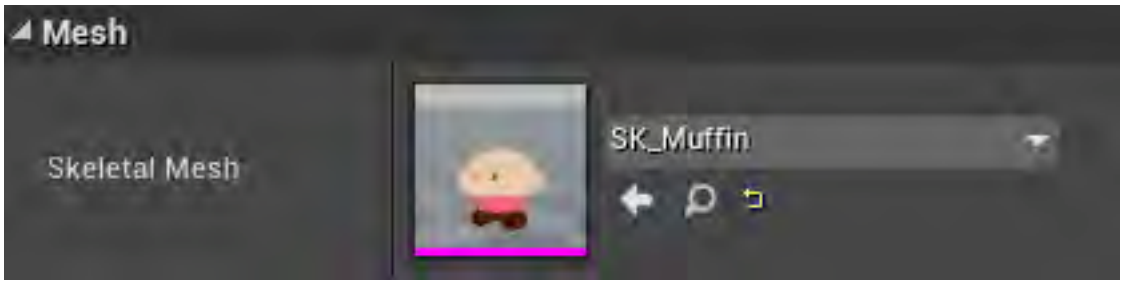


保持其它选项为默认值，然后点击Import（注意不是Import All），就会创建以下的资源：
SK_Muffin: Skeletal Mesh资源，简单来说，就是带有到Skeleton资源的链接的mesh。
SK_Muffin_Skeleton: Skeleton资源，它包含了一系列的关节点和其它信息，比如关节点的层级关系。

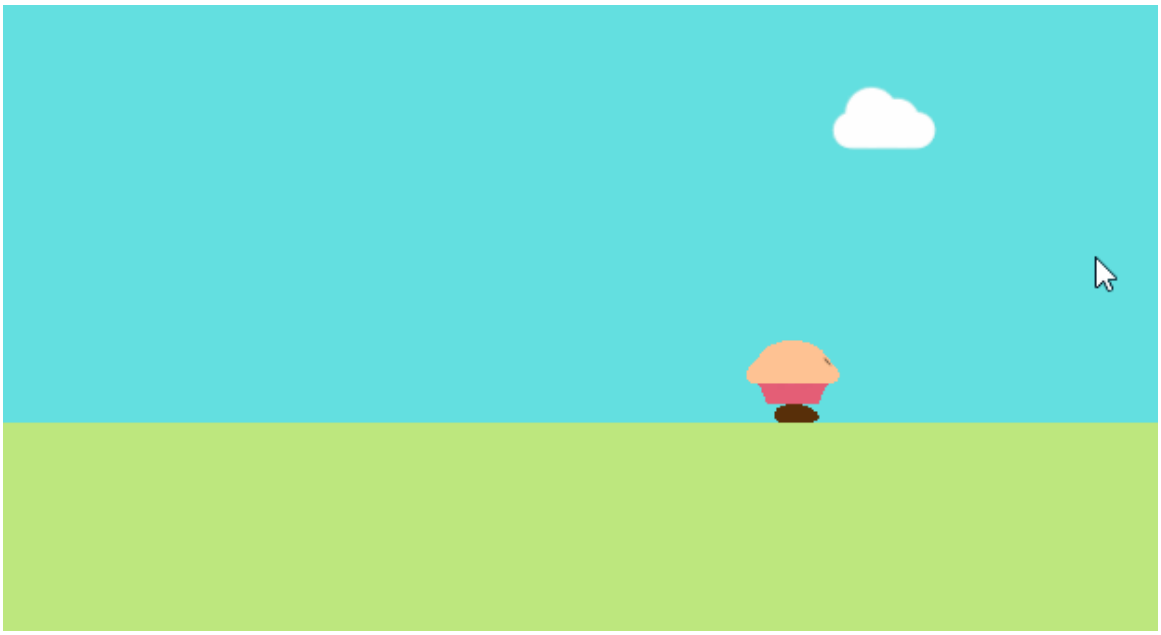
使用Skeletal Mesh

在使用新的Skeletal Mesh之前，需要提供一个材质，否则就会像个灰块一样。双击SK_Muffin打开它。
在Asset Details面板中找到Material Slots部分，为其分配M_Muffin的材质，然后关闭SK_Muffin。

好了，现在我们已经成功的让松饼小人成了游戏角色。
返回主编辑器,在Content Browser中双击打开BP_Muffin。
在Components面板中选中Mesh(Inherited)组件，然后在Details面板中找到Mesh部分。
设置Skeletal Mesh的属性为SK_Muffin。



点击工具栏上的Compile按钮，然后返回主编辑器。
点击Play，就可以看到，现在我们可以操控松饼小人来愉快的玩耍了~



现在游戏的效果比起最初当然要好玩多了。
接下来，我们将给松饼小人添加一些动画。
让我们下一课再见~

笨猫学编程QQ群：375143733
知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

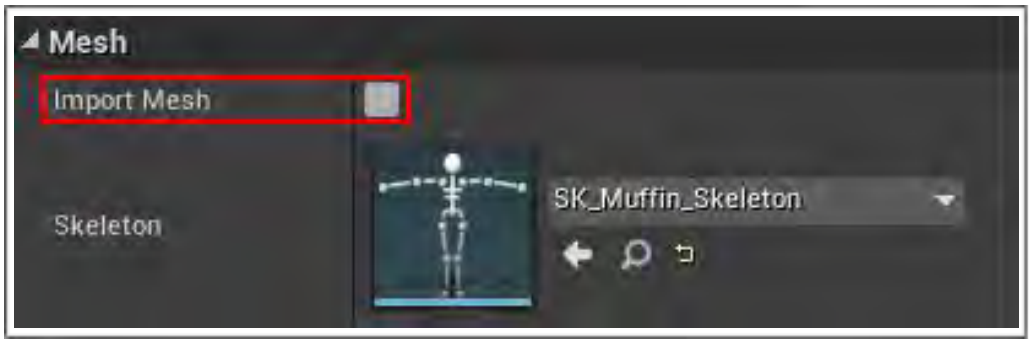
个人网站：
http://icode.ai/

欢迎继续我们的学习。
在上一课的内容中，我们让松饼小人成了游戏的主角。
而在这一课的内容中，我们将导入动画，让小人可以更加栩栩如生。

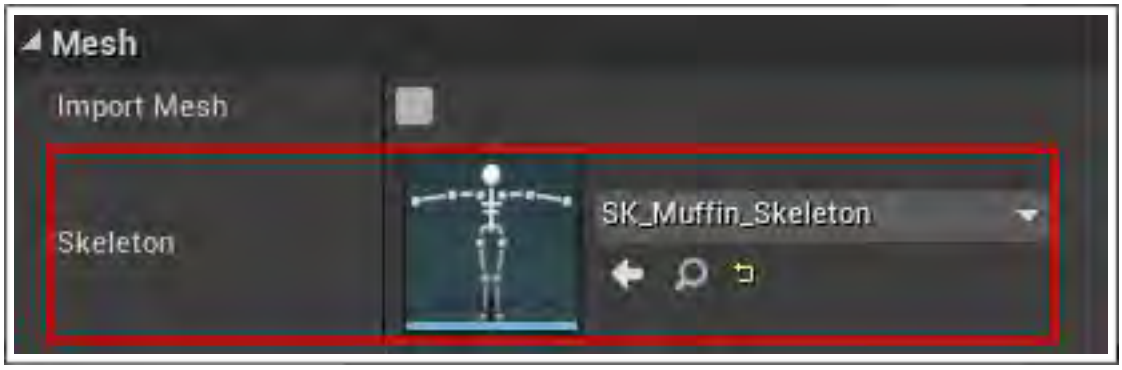
导入动画

在虚幻 4 主编辑器中找到 Content Browser，然后点击 Import，切换到文件夹 SkywardMuffinStarter\Animation Assets。选择以下文件：
SK_Muffin_Death.fbx
SK_Muffin_Fall.fbx
SK_Muffin_Idle.fbx
SK_Muffin_Jump.fbx
SK_Muffin_Walk.fbx

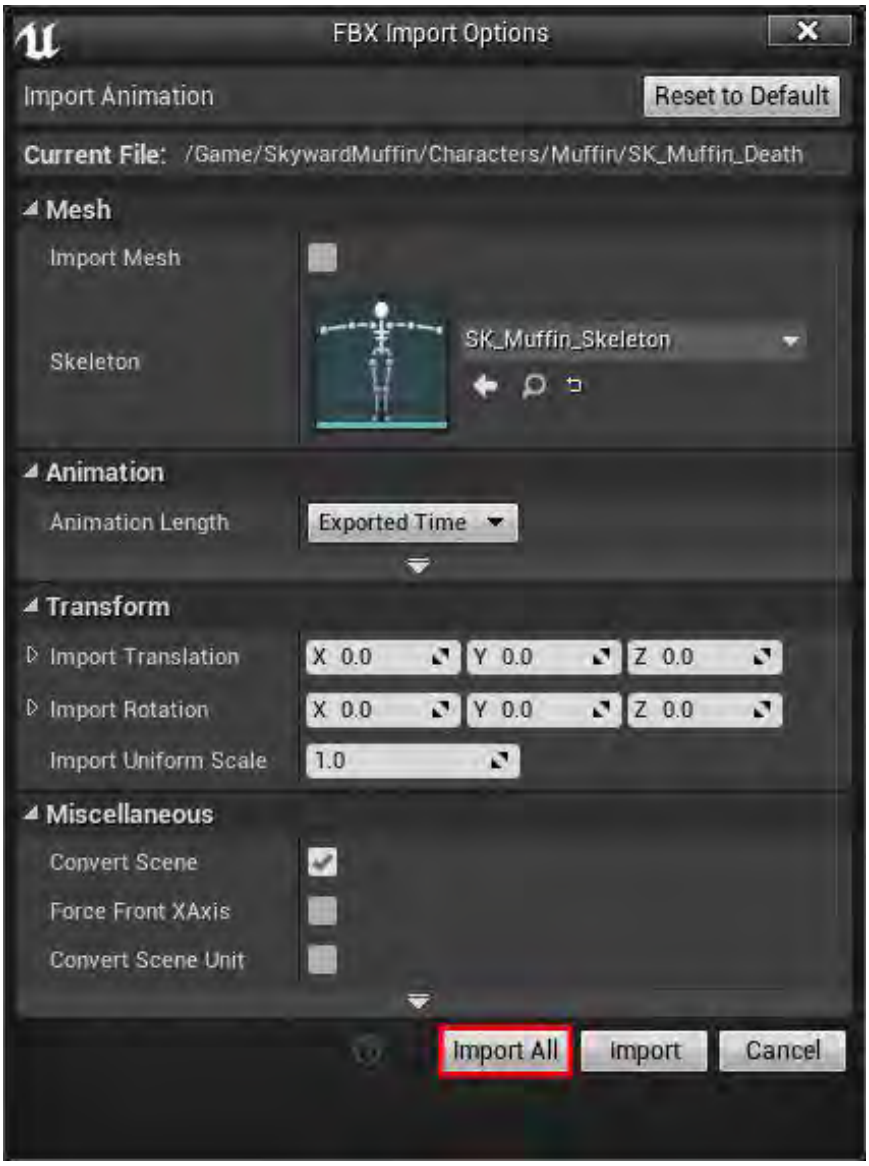
全部完成之后，点击 Open。
在导入设置窗口，找到 Mesh 部分，取消勾选 Import Mesh 选项，这样就可以确保不会再次导入 Skeletal Mesh。



接下来将 Skeleton 属性设置为 SK_Muffin_Skeleton，该选项设置了动画所使用的骨骼。



最后点击 Import All。这样就会使用同样的设置导入所有的动画。



现在我们已经导入了所有所需的动画，接下来要学习的就是如何来使用这些动画。
为此，我们需要使用 Animation Blueprint（动画蓝图）~

创建动画蓝图

动画蓝图和普通的蓝图比较类似，不过它包含了一个针对动画任务的 graph。
在虚幻 4 的主编辑器中找到 Content Browser，点击 Add New 按钮，选择 Animation\Animation Blueprint。
在弹出的创客中，找到 Target Skeleton 属性，然后选择 SK_Muffin_Skeleton。

接下来，点击 OK 按钮以创建动画蓝图。

将其命名为 ABP_Muffin。接着双击在动画蓝图编辑器中将其打开。

动画蓝图编辑器

动画蓝图编辑器和普通的蓝图编辑器比较类似，不过多了四个额外的面板：

1.Anim Graph:
这个 Graph 是专门针对动画的，我们将在这里播放动画。

2.Preview Scene Settings:
这个面板可以让我们在视图中调整预览场景

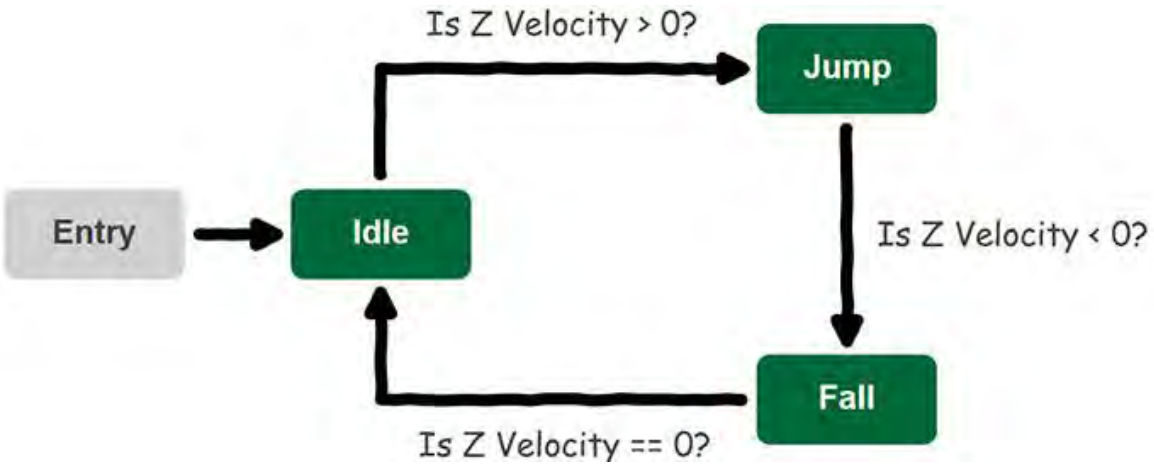
3.Anim Preview Editor:
我们所创建的变量也会在这里显示，使用该面板可以预览变量对最终动画所产生的作用。

4.Asset Browser:
该面板包含了当前骨骼所使用的一系列动画。

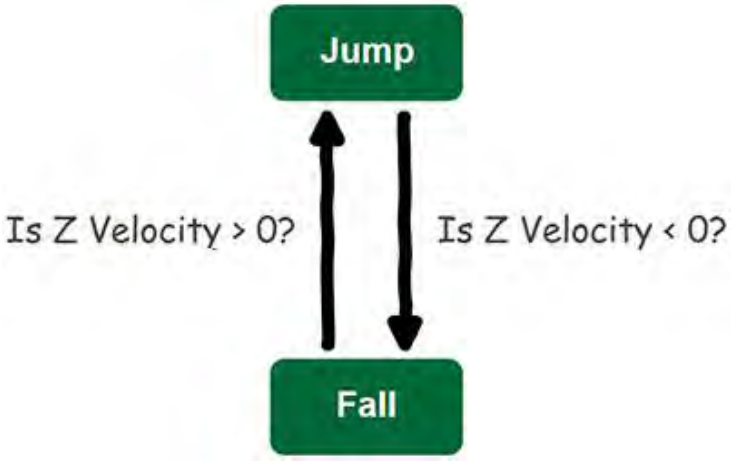
为了定义某个动画应该如何播放，我们需要使用 State Machine（状态机）。

什么是状态机？

简单来说，一个状态机就是一系列的状态和规则。对于本系列的教程来说，你可以把一个状态看做一个动画。
在任何一个时刻，状态机只能处在某个特定的状态，而为了切换到另外一个状态，就需要满足特定的条件（由规则来定义）。
下图就是一个简单状态机的示例，它显示了不同状态之间的切换规则。



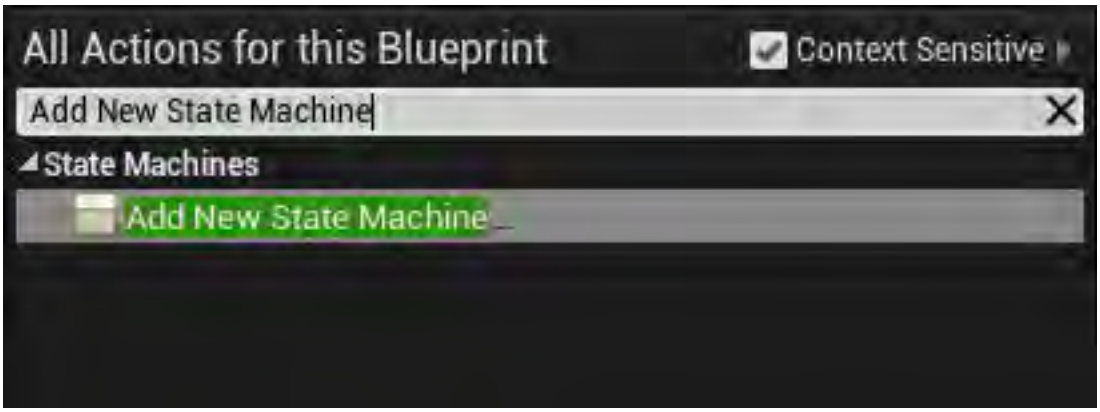
状态机之中的状态也可能是双向关联的，在下面这个例子里面，Jump 状态和 Fall 状态可以相互转化：



如果没有这种双向关联，游戏角色就没办法完成连续两次跳跃。
好了，关于状态机就先说到这里。接下来，让我们实际创建一个状态机。

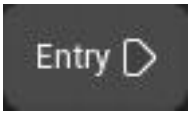
创建状态机

首先确保我们当前在 Anim Graph 中，然后右键单击空白区域，从菜单中选择 Add New State Machine。



此时，视图中将添加一个状态机节点，把它的名字更改为 Locomotion。随后将 Locomotion 状态

机连接到 Final Animation Pose 节点上。
现在 Locomotion 状态机将决定松饼小人的最终动画。



接下来双击 Locomotion 状态机将其打开，此时可以看到 Entry 节点。
连接到该节点的状态将被设置为默认状态。

对本教程来说，这里的默认状态就是 idle 动画。

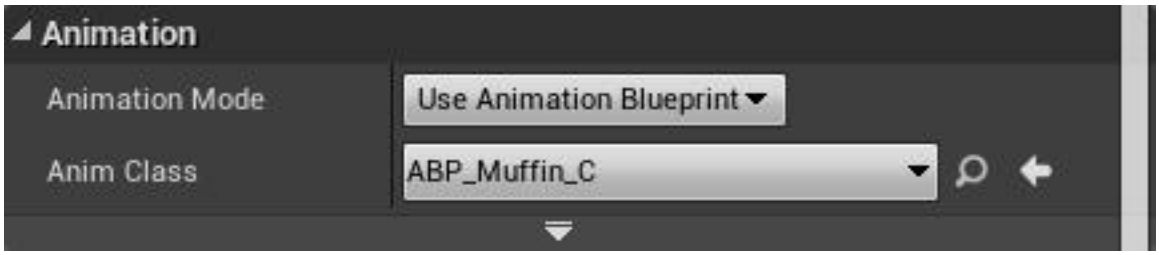
在视图中右键单击空白区域，从菜单中选择 Add State，并将其命名为 Idle。
接下来从 Entry 节点拉一条线关联到 Idle 状态上：

虽然我们从弹出菜单中创建了一个状态，但是系统并没有为我们自动分配一个动画。
因此这一点需要修复。

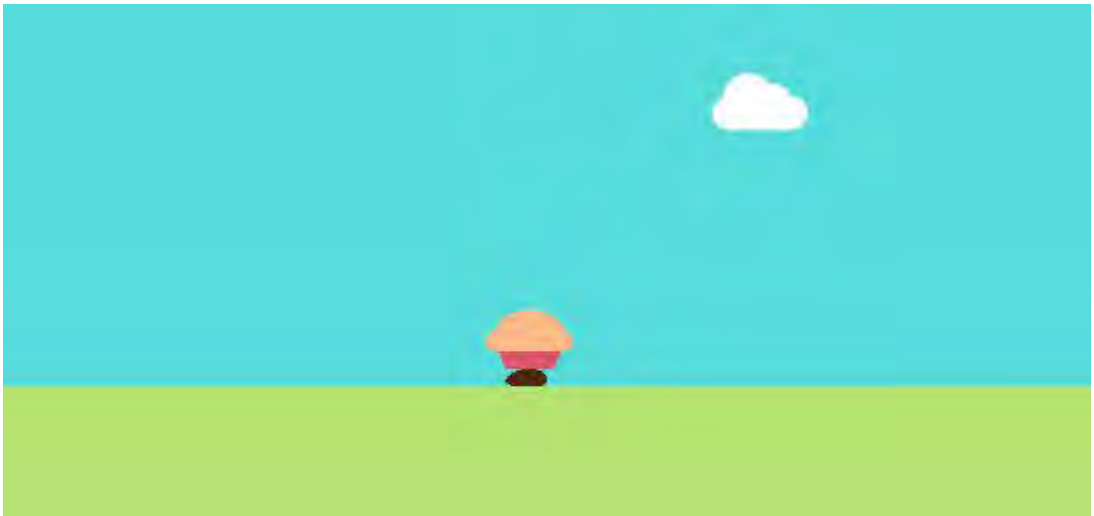
将动画关联到状态上

双击 Idle 状态将其打开。从 Asset Brower 中找到 SK_Muffin_Idle 动画，并将其拖到视图当中。
接着从 Play SK_Muffin_Idle 节点连一条线到 Final Animation Pose 节点。
为了使用动画蓝图，我们需要更新 BP_Muffin。

使用动画蓝图



点击工具栏上的 Compile 按钮，然后切换到 BP_Muffin。
在 Components 面板中选择 Mesh(Inherited)组件。在 Details 面板中找到 Animation 部分。
将 Animation Mode 设置为 Use Animation Blueprint。然后设置 Anim Class 为 ABP_Muffin。



好了，现在 Skeletal Mesh 将使用 ABP_Muffin 作为自己的动画蓝图。
点击工具栏上的 Compile 按钮，然后关闭 BP_Muffin。
在主编辑器中点击 Play 按钮以测试动画蓝图。
因为 Idle 是默认状态，所以松饼小人将默认使用 idle 动画。
好了，这一课的内容就先到这里。
在下一课的内容中，我们将分别创建跳跃和跌倒的动画状态。
我们下一课再见~

讨论群-笨猫学编程 QQ 群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

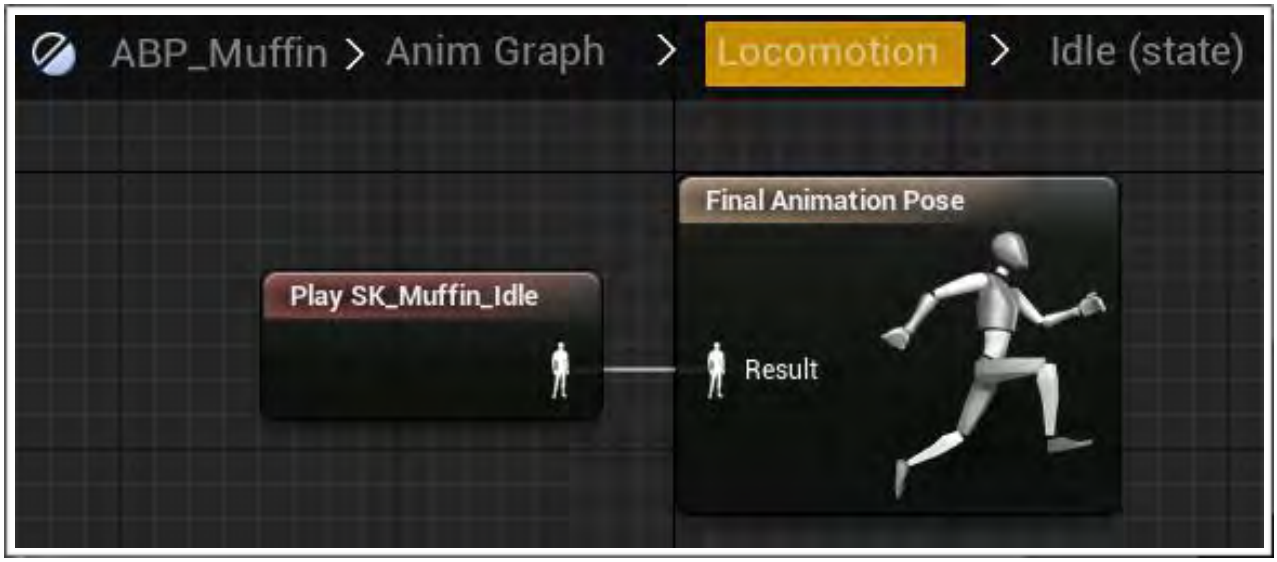
新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

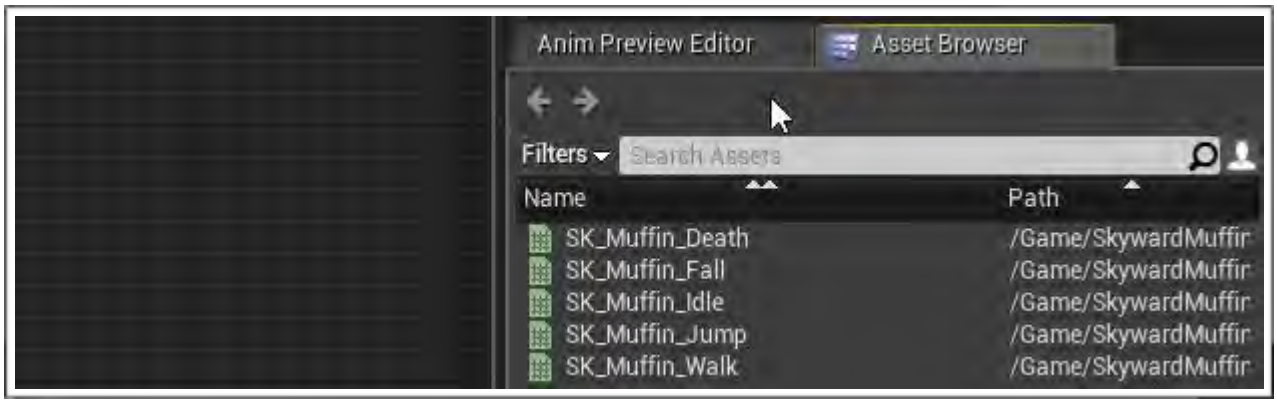
个人网站：
http://icode.ai/

欢迎继续我们的学习。

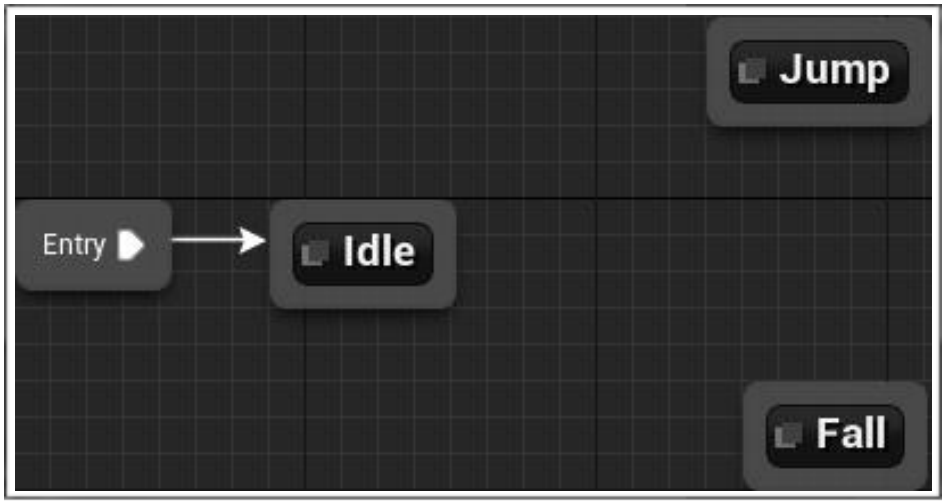
在这一课的内容中，我们将给松饼小人添加更多的动画状态。
在蓝图编辑器中打开 ABP_Muffin，切换到 Locomotion 的状态机视图。



在之前的操作中，我们先创建了一种状态，然后将其和动画关联起来。其实也可以直接创建一个已关联动画的状态。对于跳跃动画，我们就可以这样来操作。
从蓝图编辑器右侧的 Asset Browser 中找到 SK_Muffin_Jump 动画并拖放到视图中，这样就创建了一个已关联了动画的状态。



将该状态重命名为 Jump。
接着把 SK_Muffin_Fall 动画也拖到视图中，将其命名为 Fall。
这样我们就有了三种状态：Idle,Jump 和 Fall。



接下来需要在不同的状态之间创建关联。从初始状态拖一条线到目标状态就可以创建一个动画过渡了。

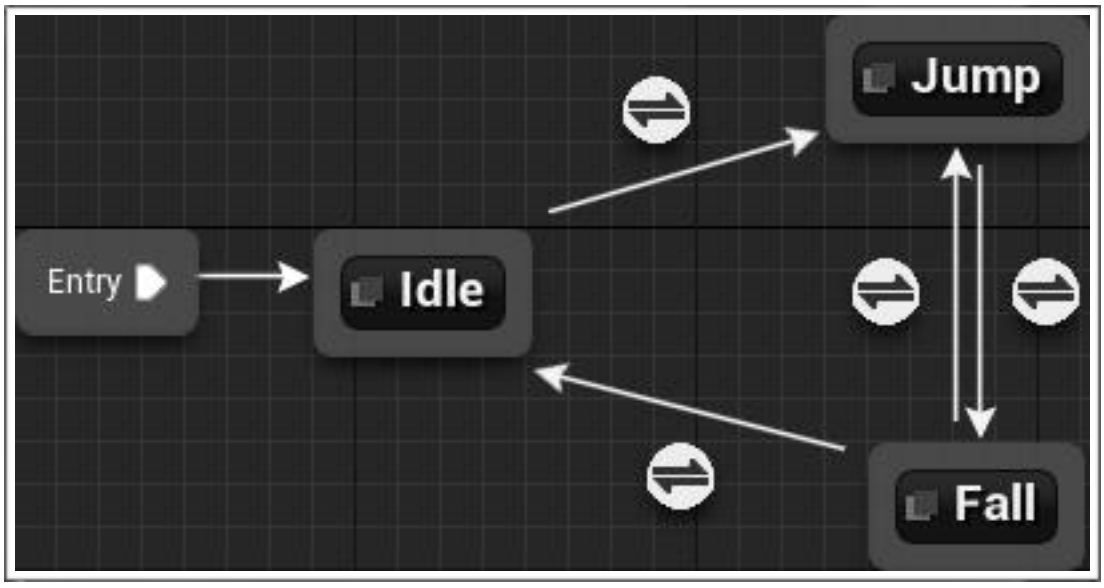
接下来让我们创建以下的几种过渡：

Idle to Jump

Jump to Fall

Fall to Jump

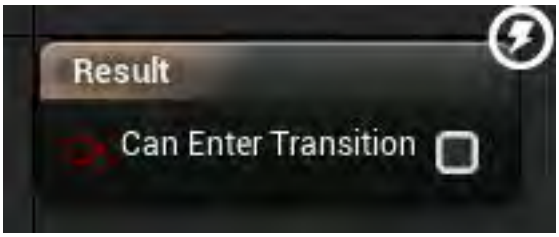
Fall to Idle



现在我们已经创建好了状态之间的过渡，接下来需要定义何时才会切换动画状态。我们将使用 Transition Rules 来实现。

Transition Rules(过渡规则）

在蓝图中，以下图标代表的就是过渡规则：

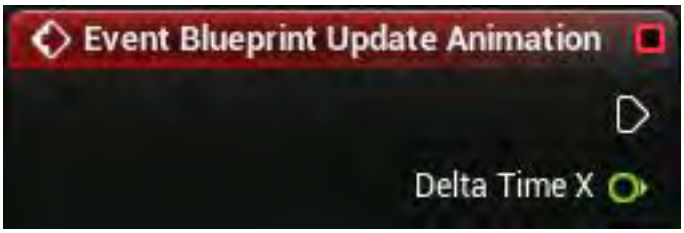


每个 Transition Rule 都包含一个 Result 节点，并带有一个单一的布尔输入值。
如果输入条件为 true，就会进行状态的切换。
接下来，我们将创建新的变量，用来获取玩家当前的状态究竟是跳跃或是跌倒。然后可以在 Transition Rules 中使用这些变量。

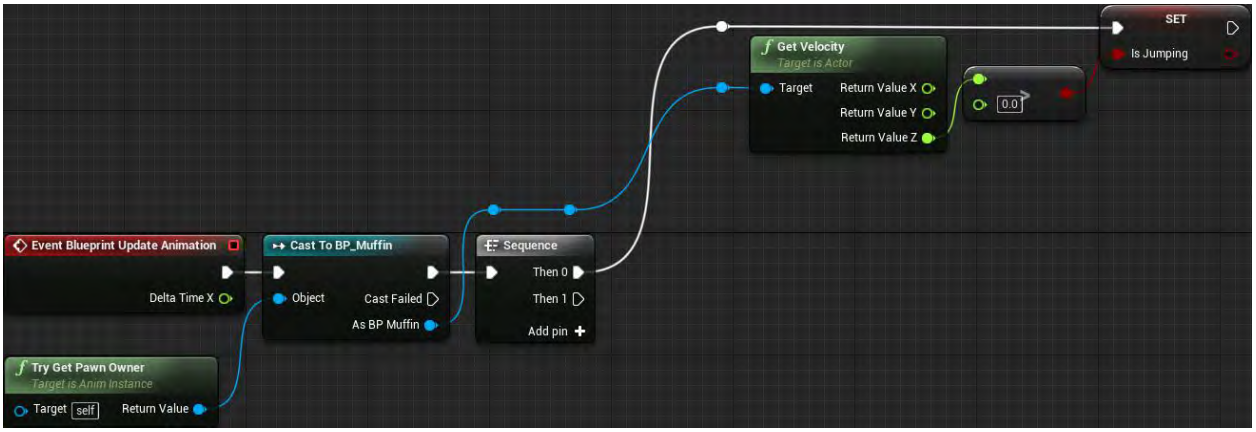
检查玩家是否在跳跃或是跌倒

创建两个布尔变量，分别命名为 IsJumping 和 IsFalling。

首先，我们将设置 IsJumping 的值。切换到 Event Graph，然后找到 Event Blueprint Update

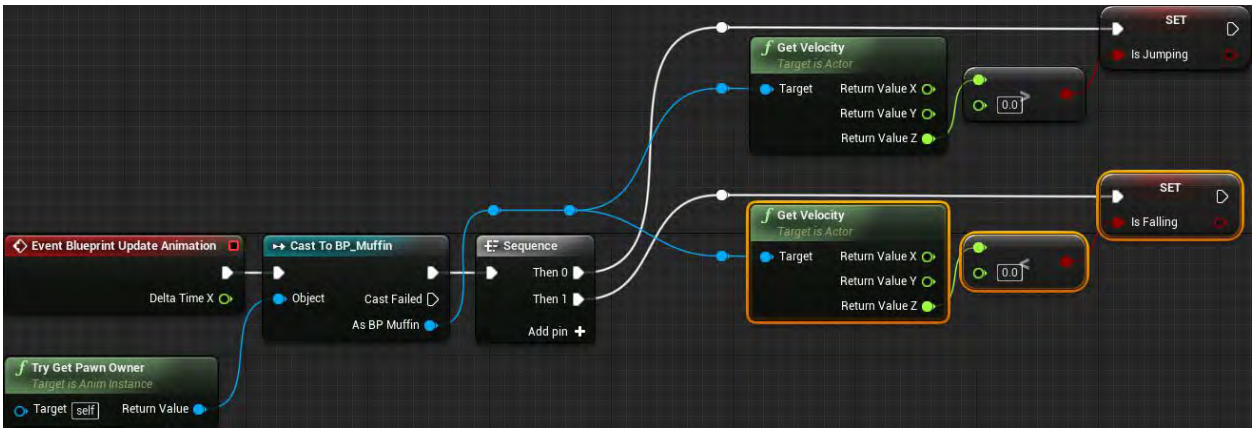


Animation 节点。该节点的作用和 Event Tick 节点类似。



为了检查玩家当前是否在跳跃，可以创建以下设置：

在以上的设置中，我们会检查玩家在 Z 轴上的的 velocity 是否大于 0。如果是，那么玩家的 IsJumping 状态就会被设置为 true。



为了检查玩家是否在跌落（falling),我们只需要做相反的检查即可，添加以下节点和连接设置：

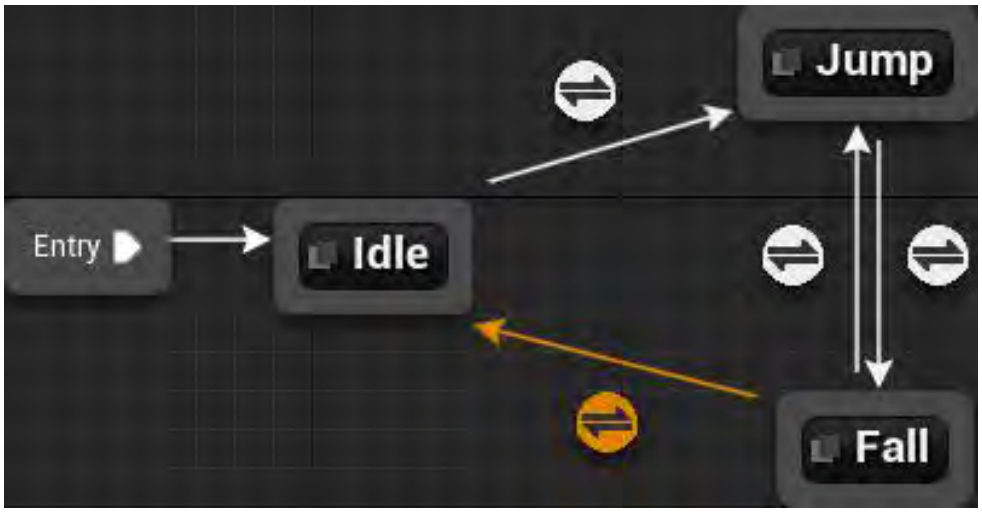
这样设置以后，当玩家在 z 轴上的速度小于 0 的时候，IsFalling 就会被设置为 true。
接下来将使用这些变量来定义动画状态切换规则。

定义动画状态切换规则

首先需要设置从 Idle 到 Jump 的过渡规则。切换回 Locomotion 状态机视图。
双击 Idle to Jump 打开该规则。

在视图中添加 IsJumping 节点，并将其连接到 Result 节点上。
此时，当 IsJumping 状态为 true 时，Idle 动画状态会切换到 Jump 状态。
对 Jump to Fall 和 Fall to Jump 做类似的操作，注意使用以下参数：
Jump to Fall: IsFalling
Fall to Jump: IsJumping

好了，现在 Jump 和 Fall 状态都可以互相切换了。
还有一个状态切换规则需要定义。打开 Fall to Idle 规则。
为了回到 Idle 状态，玩家不能跳跃或跌落。为此，我们将用到 NOR 节点。只有当两个输入都是 false 的时候，该节点才会返回 true。
创建一个 NOR 节点，然后将 IsJumping 和 IsFalling 节点连接到该节点上。然后将 NOR 节点连接到 Result 节点。



好了，现在当 IsJumping 和 IsFalling 状态都是 false 的时候，就会从 Fall 状态切换到 Idle 状态。
点击蓝图编辑器上的 Compile 按钮，然后返回主编辑器，点击 Play 可以测试状态切换。

好了，本课的内容就到这里了。
我们下一课再见~

讨论群-笨猫学编程 QQ 群：
375143733

答疑论坛：

http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

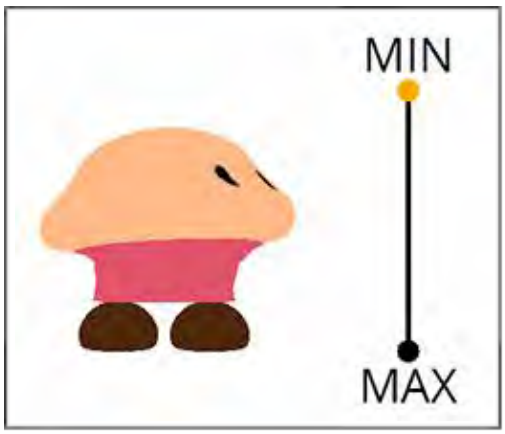
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

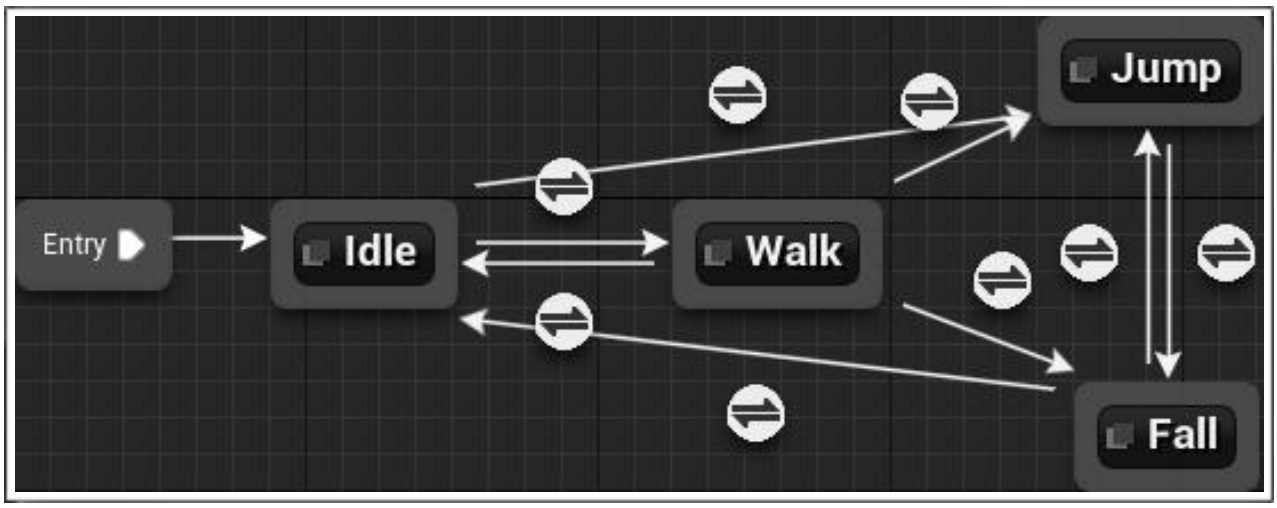
欢迎继续我们的学习。
到目前位置，松饼小人在地面上移动的时候并不自然，这是因为我们还没有使用行走动画！
不过这里我们不需要创建一个新的行走状态，而是使用 Blend Space 的方式来实现。

什么是 Blend Space？

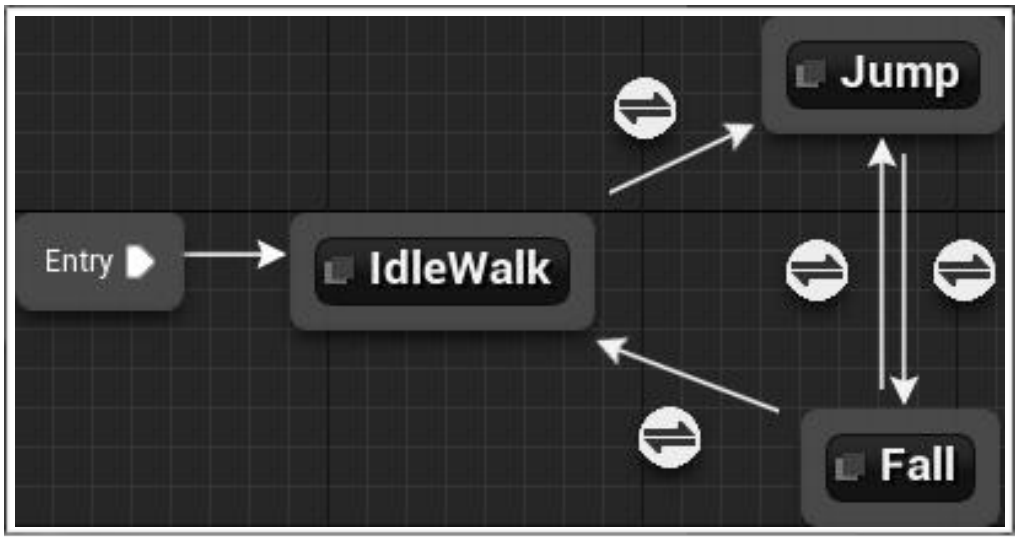
Blend Space 是一种动画集合的类型，它会基于所输入的值来插入不同的动画状态。
在本教程中，我们将使用玩家的 speed 作为输入。



使用 Blend Space 可以帮助简化状态机。
如果不适用 Blend Space，那么在添加了行走状态后的状态机可能是下面这样的：



但是在使用了 Blend Space 之后，我们只需要替代 idle 动画即可。



现在你明白 Blend Space 的强大了吧？好了，废话少说，让我们来创建一个。

创建 Blend Space

在虚幻 4 主编辑器的 Content Browser 中点击 Add New，然后选择 Animation\Blend Space 1D。

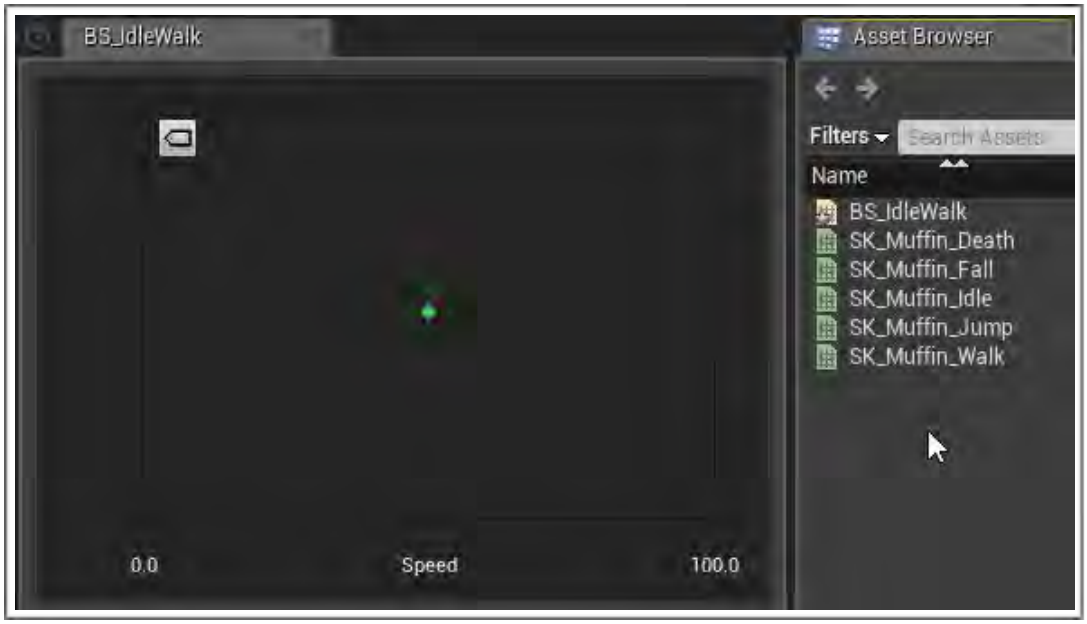
注意：
Blend Space 和 Blend Space 1D 的区别在于前者有两个输入，而后者只有一个输入。
在我们这种情况下只有速度一个输入，所以需要选择 Blend Space 1D.
从弹出菜单中选择 SK_Muffin_Skeleton。

将这个新创建的游戏资源重命名为 BS_IdleWalk，然后双击在动画编辑器中将其打开。
打开之后，在底部会看到一个面板，这就是 Blend Space 的编辑器，我们可以在这里添加所需的动画。

现在让我们给 Blend Space 添加一些动画。

给 Blend Space 添加动画

首先我们需要更改输入轴的名称。找到 Asset Details 面板，然后找到 Axis Settings 部分。将 Horizontal Axis\Name 属性更改为 Speed。
接下来让我们添加动画。
在右侧的 Asset Browser 中将 SK_Muffin_Idle 动画拖到底部左侧的 Blend Space 灰格，让其贴近



0.0 的数值处，松开鼠标添加该动画。
注意：为了显示动画的名称，请点击 Blend Space 灰格区左上的小图标。

接着将 SK_Muffin_Walk 动画添加到 100.0 的数值处。

现在 Blend Space 将会根据所输入的 speed 值来混合 idle 和 walk 动画。如果输入为 0，那么只会播放 idle 动画，如果输入为 100，那么就会播放行走动画。如果输入在 0 到 100 之间，就会播放混合动画。
好了，现在可以来尝试使用 Blend Space 的时候了。

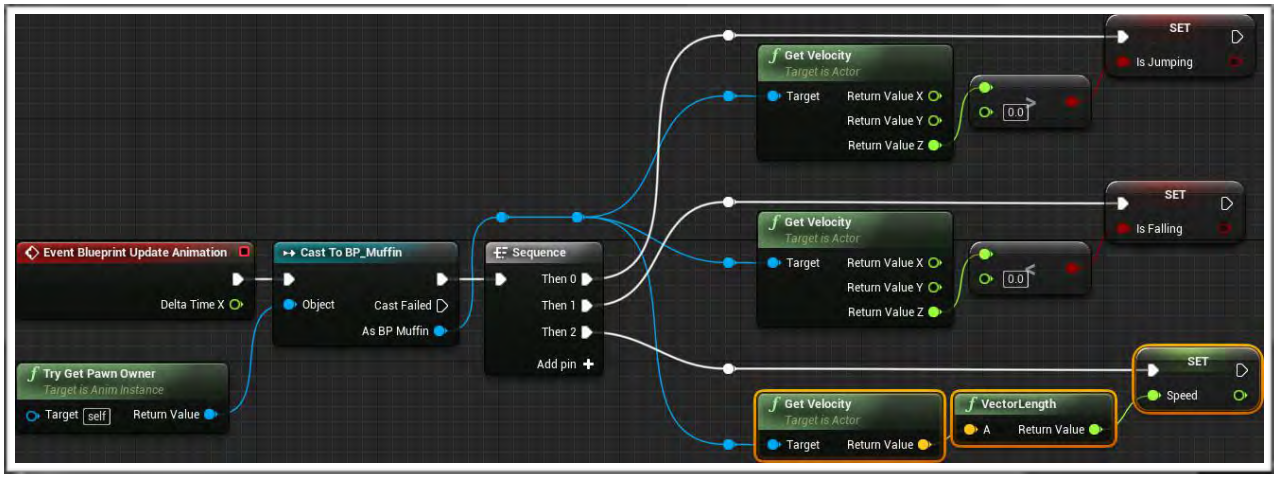
使用 Blend Space

关闭 BS_IdleWalk，然后打开 ABP_Muffin。切换到 Locomotion 状态机，然后打开 Idle 状态。
首先删掉 Play SK_Muffin_Idle 节点。
接着将 BS_IdleWalk 这个 Blend Space 添加进来，然后将 BS_IdleWalk 节点跟 Final Animation

Pose 节点连接起来。
现在 BS_IdleWalk 会自动播放，不过因为 Speed 输入始终是 0，所以只会显示 idle 动画。
为此，我们需要向它提供玩家的 speed 属性。

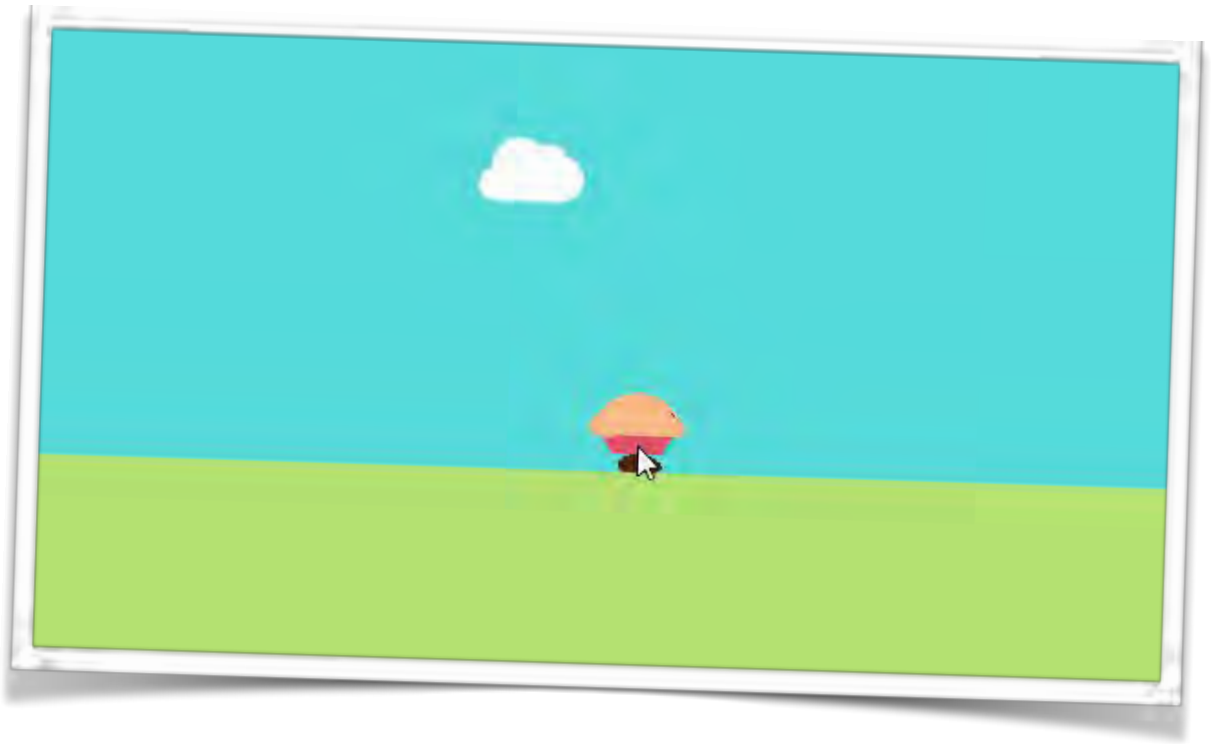
获取玩家的 Speed

创建一个新的 float 变量，将其命名为 Speed，然后切换到 Event Graph。
在 Sequence 节点上添加一个新的 pin 端口，然后使用如下方式添加高亮的节点。



以上设置就可以将 Speed 变量的值设置为玩家的速度。
现在切换回 Idle 状态的 graph 视图，将 Speed 变量连接到 BS_IdleWalk 节点的 Speed 输入。

现在，BS_IdleWalk 就可以混合 idle 和 walk 动画了。
点击工具栏上的 Compile 按钮，然后返回主编辑器。点击 Play 测试 Blend Space 的效果~



好了，本课的内容就到这里了，我们下一课再见~

讨论群-笨猫学编程 QQ 群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:

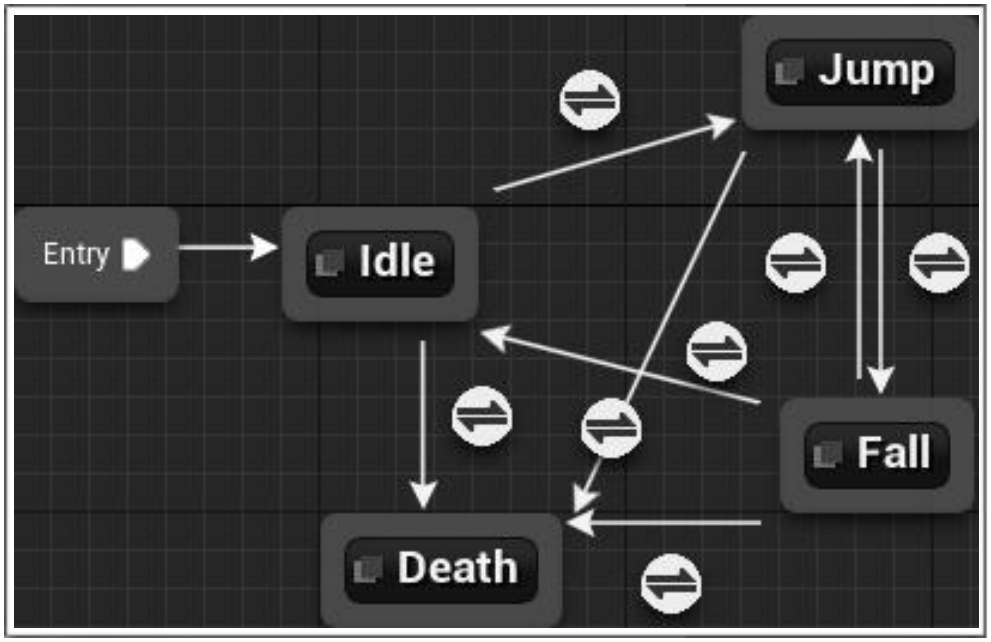
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。
接下来我们还要再添加一个动画，也就是玩家的死亡动画。

使用死亡动画

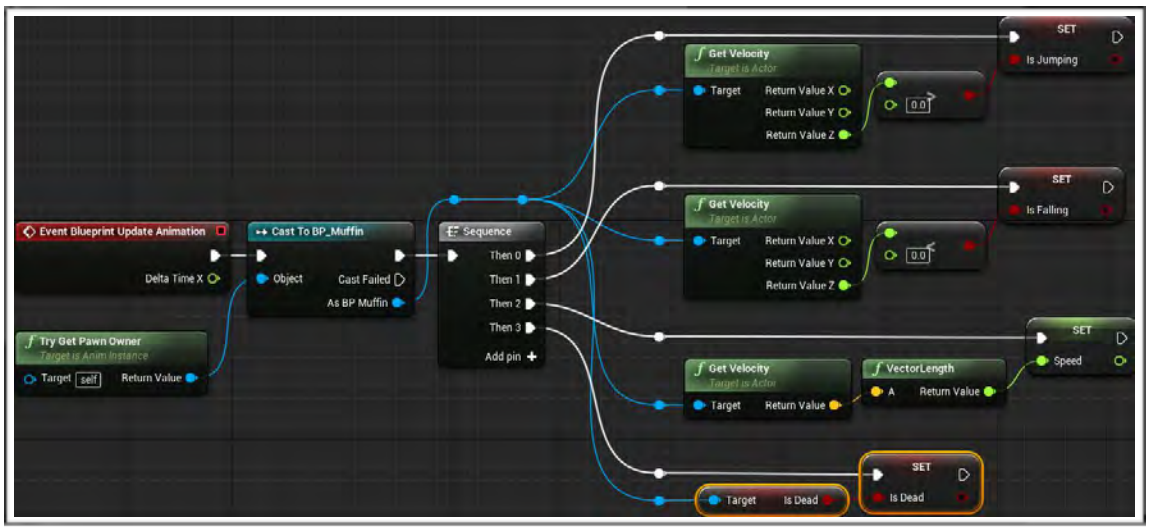
在这个游戏中，只有在 Idle 状态下才可能死亡（地面上）。不过假定玩家可以在任何状态下死亡，那么可能大家首先想到的就是创建一个 Death 状态，然后将所有的状态都和它关联在一起。虽然这也是一个选项，但是会让整个视图变得异常混乱。



解决的方法之一就是使用 Blend Poses by bool 节点。该节点可以基于输入的布尔值在两个动画之间进行切换。
在创建该节点之前，首先需要一个变量保存玩家的死亡状态。

检查玩家是否已死亡

返回 ABP_Muffin ，创建一个布尔变量，将其命名为 IsDead。
然后切换到 Event Graph，在 Sequence 节点上添加一个新的 pin 端口，然后添加如下图中的高亮节点：



这样就可以让 IsDead 变量的值跟玩家的死亡状态绑定起来。
接下来将使用 Blend Poses by bool 节点。

使用 Blend Poses by Bool Node

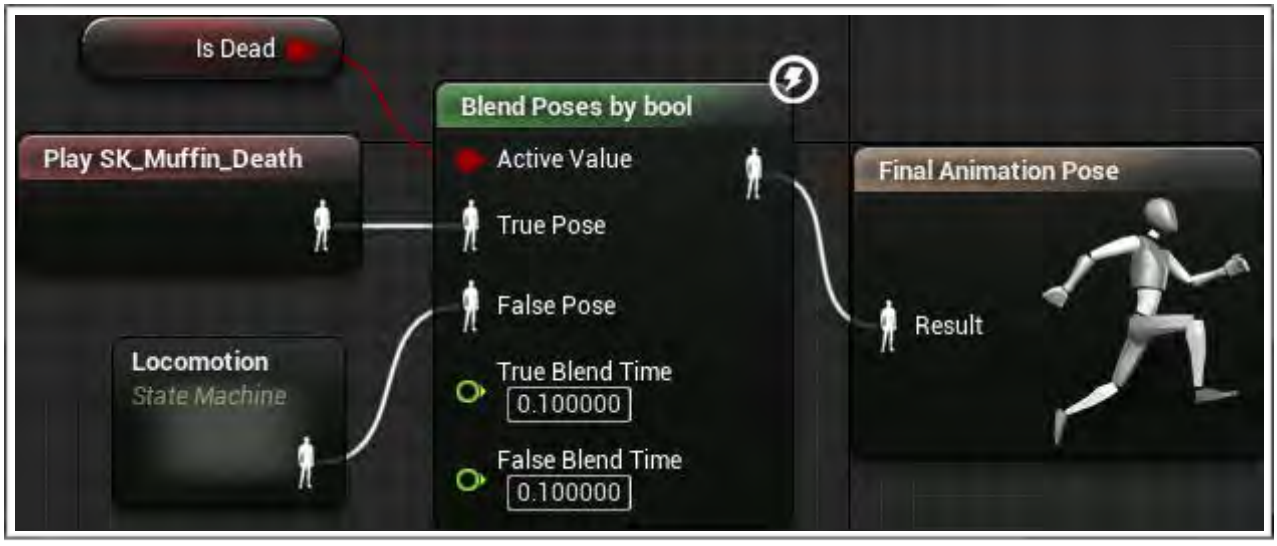
切换到 Anim Graph 视图，然后添加 SK_Muffin_Death 动画。选中该动画，然后在 Details 面板中取消勾选 Loop Animation 属性。

这样就可以保证死亡动画只会播放一次。
接下来创建一个 Blend Poses by bool 节点。

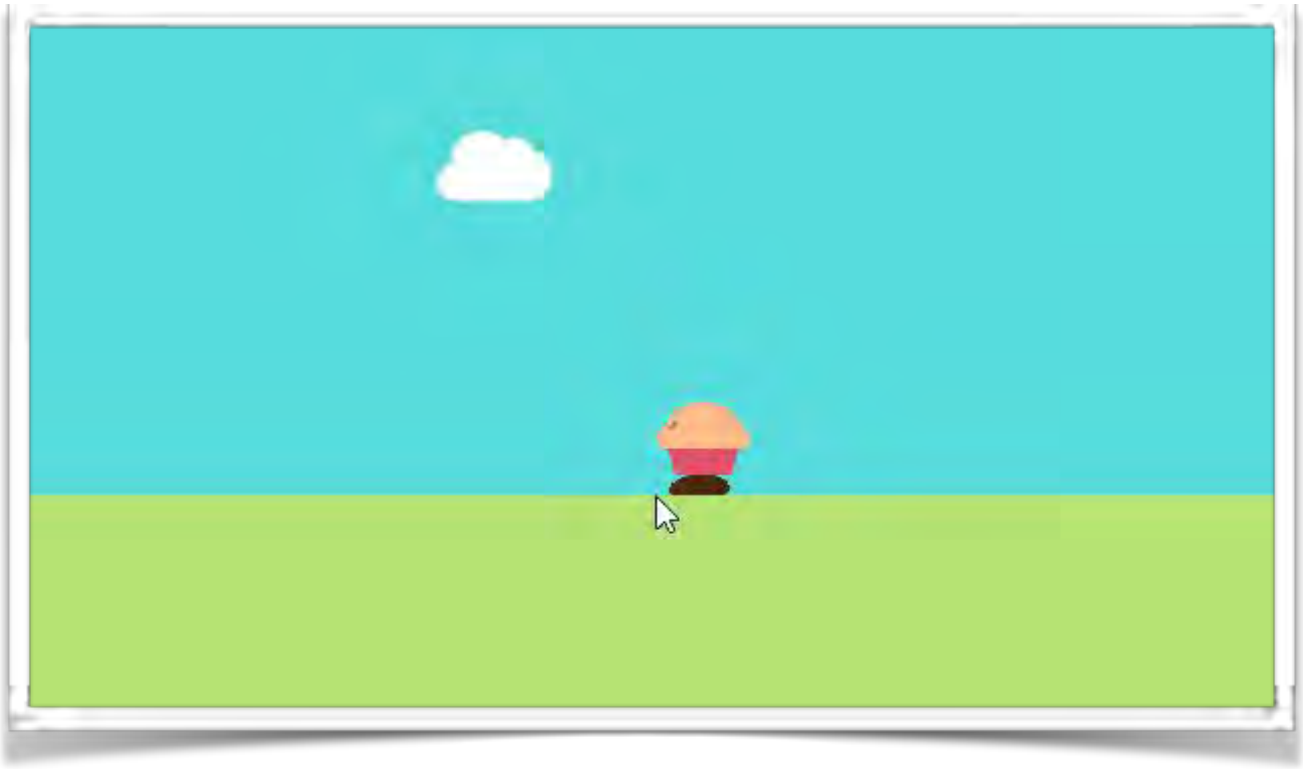
选中该节点，在 Details 面板中的 Option 部分勾选 Reset Child on Activation 属性。

因为死亡动画只会播放一次，因此该选中可以确保动画在回放前重置。

最后添加 IsDead 节点，然后使用下图的方式创建连线：



现在当 IsDead 为 true 时，就会播放死亡动画。如果 IsDead 为 false，就会播放 Locomotion 状态机的当前动画。
点击工具栏上的 Compile 按钮，然后关闭 ABP_Muffin。
回到主编辑器，点击 Play 按钮来测试新的死亡动画。



好了，关于动画的内容就到这里结束了。
完整的项目请参考这里。
链接:https://pan.baidu.com/s/1r26LQfAKbIUMIPkFvOUdRw 密码:fu9y
关于动画的更多知识，请参考官方网站：
Skeletal Mesh Animation System

在下一部分的内容中，我们将学习如何给游戏添加音乐和音效。
让我们下一课再见~

讨论群-笨猫学编程 QQ 群：
375143733

答疑论坛：

http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:

http://blog.sina.com.cn/eseedo

Github:

https://github.com/eseedo

个人网站：

http://icode.ai/

欢迎继续我们的学习。
在游戏中，Audio（声音）指的是背景音乐、对话以及各类音效。如今，任何一款游戏如果没有声音都是不可想象的。
声音可以大大提升玩家在游戏中的沉浸感。音乐可以激发玩家的情感反应，对话可以让游戏角色的形象变得更加生活，同时让游戏故事变得更加丰满。音效可以向玩家提供声音反馈机制，让游戏场景变得更加真实化。所有这一切加在一起，有可能让一个优秀的游戏变成一款伟大的游戏。
在这个教程中，我们将学习以下内容：
1.如何播放游戏背景音乐，以及如何设置是否循环播放
2.如何在动画的特定时点播放音效
3.如何在播放声音的时候调整音调
4.如何根据 3D 空间中的位置调整声音的大小。
5.使用 UI 设置界面分别调节引用和音效的音量大小。

需要注意的是，在本系列的教程中，我们仍然会使用蓝图（Blueprints）。如果你对蓝图的基础知识还是不太熟悉，建议复习一下之前的教程。
此外，强烈建议大家在学习本教程的过程中使用耳机，特别是涉及到空间音效的地方。

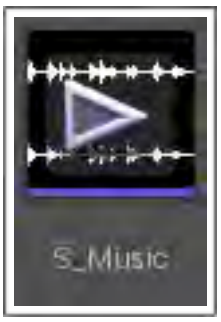
开始前的准备

在开始学习之前，先下载起始项目。
链接:https://pan.baidu.com/s/1tDdRN8FQ7JPR4NCuM9uLwQ 密码:vgt8

解压项目后，打开 SkywardMuffin.uproject 文件。

点击主编辑器工具栏上的 Play 按钮开始游戏。
游戏的目标很简单，就是在掉落天际之前触碰到尽可能多的云朵~
点击　左键可以开始登云路。
这是一款很有趣的小休闲游戏，但遗憾的是现在还没有声音。所以首先我们将添加一个让人安静的　背景音乐。

播放背景音乐
在 Content Browser 中找到 Audio 文件夹，在这里我们可以看到本教程中所需要用到的所有声音



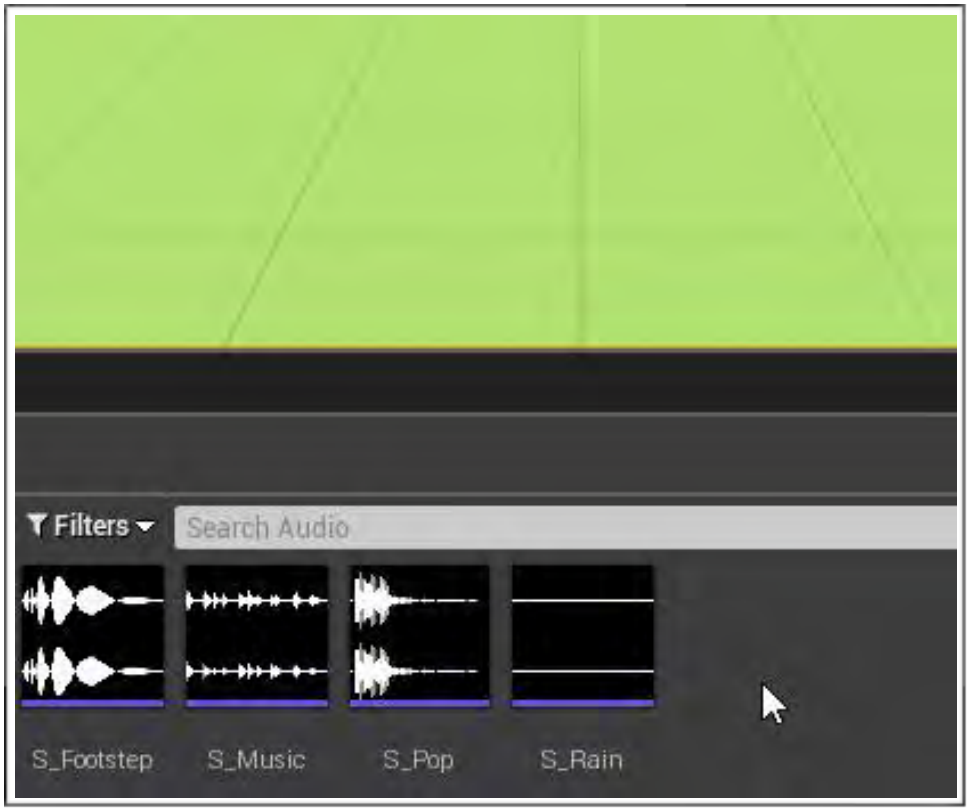
文件。在实际使用之前，大家可以把鼠标放到音乐文件的图标上，然后点击播放小图标来听一听。

在虚幻 4 中播放音乐非常简单，只需要把音乐资源文件拖进游戏场景的 Viewport 中即可。不过这样的话音乐只会播放因此，因为我们需要手动启动资源是否循环播放。

双击 S_Music 文件将其打开。

此时会打开一个新的创客，其中显示了细节信息。在 Sound Wave 部分勾选 Looping 选项。接下来返回主编辑器，然后把 S_Music 资源拖到游戏的 Viewport 中。

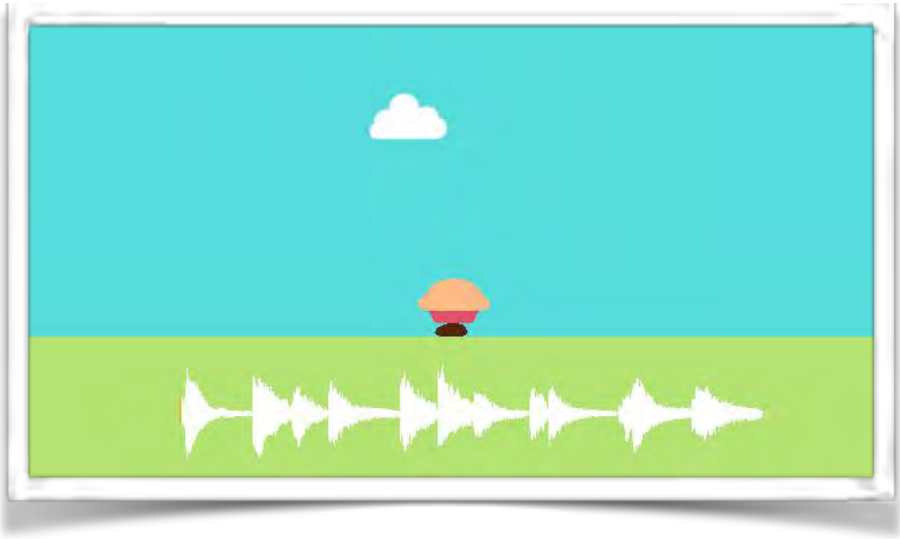
通过这种方式，就创建了一个 AmbientSound 类型的 actor 角色，它将在游戏开始之后自动播放



S_Music。

现在点击 Play 按钮，就可以听到游戏中开始播放音乐了。17 秒后，音乐会自动循环播放。

好了，本课的内容就到这里了，我们下一课再见~



讨论群-笨猫学编程 QQ 群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

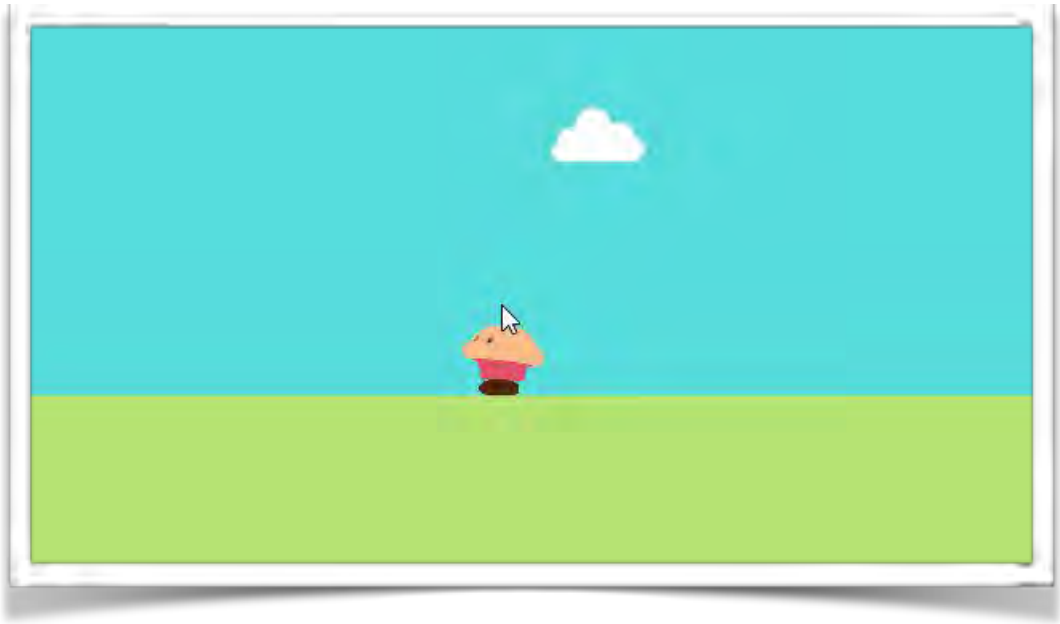
新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。
在上一课的内容中，我们给游戏添加了背景音乐。
接下来我们希望给松饼小人添加行走时的音效，为此将用到 Animation Notify.

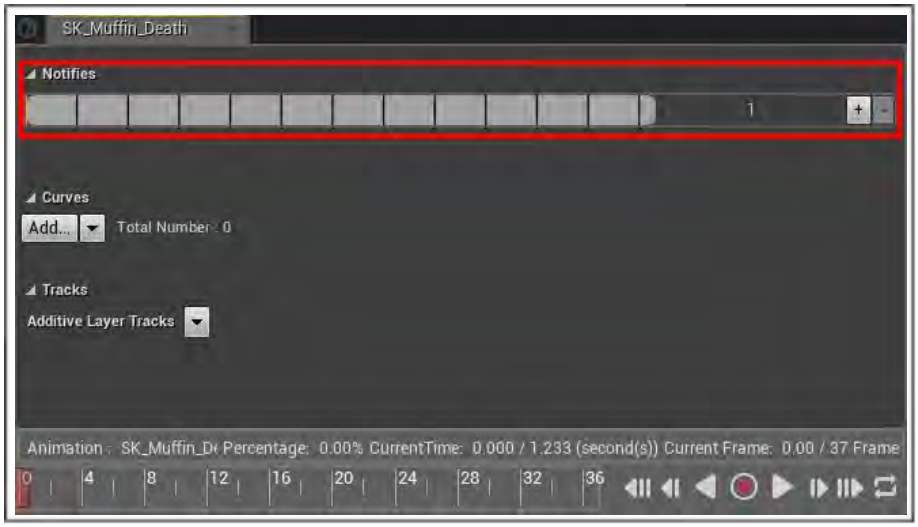
什么是 Animation Notify（动画通知）？
Animation Notify 可以在动画的某个特定节点触发事件。我们可以有多种方式来使用 Animation Notify，比如创建一个 Notify 来生成粒子特效。
在我们这个游戏中，当松饼小人碰到地面后，就会出现重新开始的按钮。但如果使用 Notify，我们可以让重新开始游戏的按钮在死亡动画的结尾处才开始显示。



在本教程中，我们将使用 Animation Notifies，从而让松饼小人的脚碰到地面的时候播放音效。

创建一个 Animation Notify

在虚幻 4 的主编辑器中找到 Characters\Muffin 文件夹，打开其中的 SK_Muffin_Walk。
在动画编辑器 Viewport 下面的面板中，可以看到一个名为 Notifies 的区域，其中的淡灰色区域就是 Notify Track。我们将在这里创建和管理 Notifies。



其中第 10 帧和第 22 帧是松饼小人的每只脚触碰到地面的时刻，因此我们需要在这两个点分别创建一个 Notify。为了创建 Notify，右键单击 Notify Track，选择 Add Notify\Play Sound。这样，我们就创建了一个名为 PlaySound 的 Notify。

接下来，我们需要将 Notify 的位置设置在第 10 帧。

移动 Animation Notify

直接在 Notify Track 上移动 Notify 比较困难，因为上面没有标注第 10 帧的位置。不过我们可以使用 timeline 来显示一个标记。
首先，找到面板底部的 Timeline。首先暂停动画，然后将红色的播放头标记拖动到 Current Frame 差不多是 10 的地方。

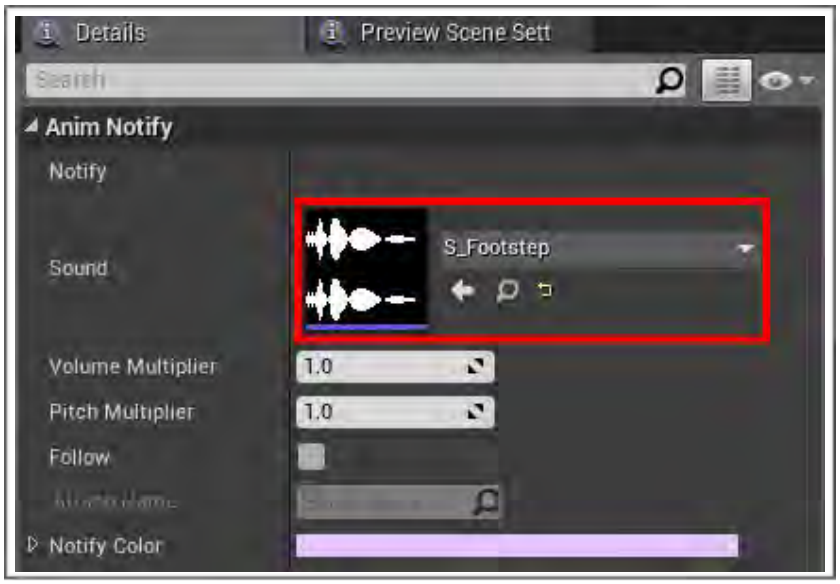
现在，Notify Track 中会生成一个红线，标记播放头所在的位置。

将 PlaySound 这个 Notify 拖动到和红线对齐的位置。

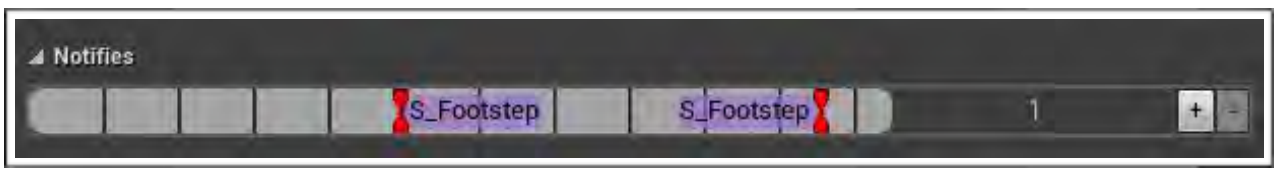
接下来，我们需要让 Notify 播放行走的脚步声。

播放脚步声

点击选中 PlaySound 这个 Notify，然后在右侧的 Details 面板中找到 Anim Notify 部分，将 Sound 设置为 S_Footstep。



接下来对另外一只脚进行相似的设置：
1.创建另外一个 Play Sound Notify
2.将 Notify 设置到第 21 帧
3.将 Notify 的音乐设置为 S_Footstep。



现在，只要行走动画运行到第 10 帧和第 21 帧，就会触发 Notify 并播放 S_Footstep 声音。
关闭 SK_Muffin_Walk，然后返回主编辑器。

点击 Play 按钮，可以体验下脚步声的音效。
好了，本课的内容就先到这里。
我们下一课再见~

讨论群-笨猫学编程 QQ 群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

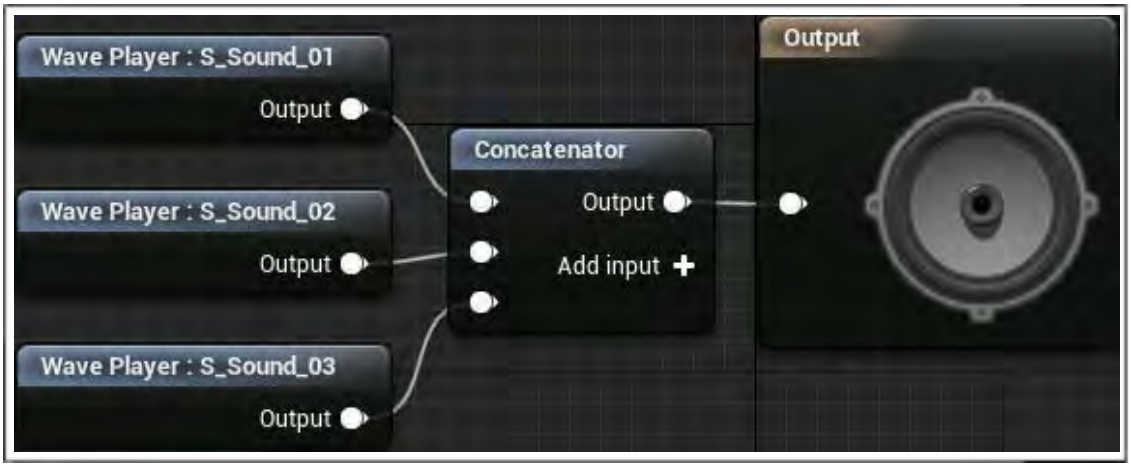
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

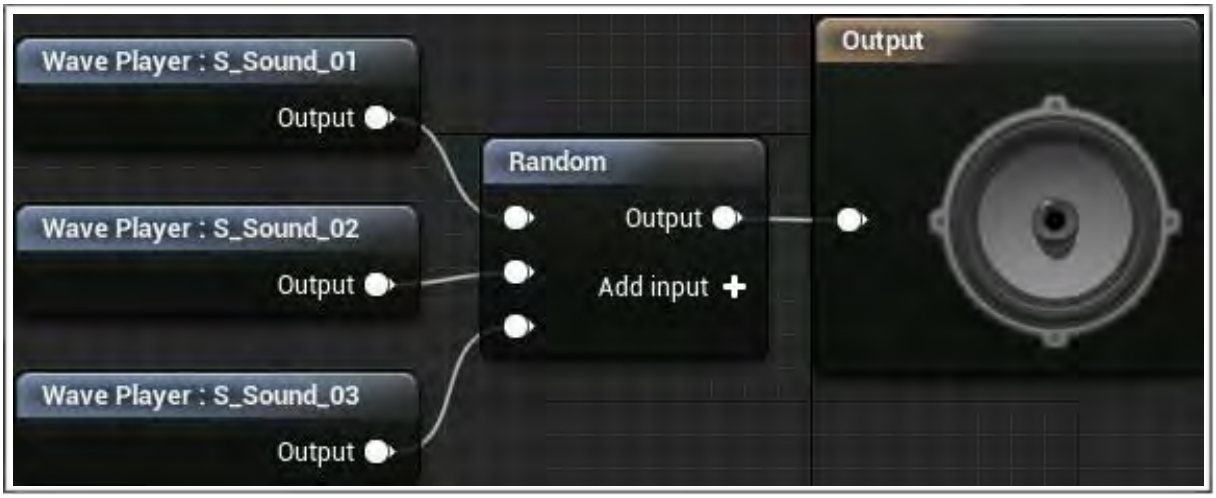
欢迎继续我们的学习。
在上一课的内容中，我们使用 Animation Notify 的方式给松饼小人添加了脚步声。
但遗憾的是，这种脚步声一直重复，几乎没有任何变化，这样似乎略有些无聊。
如果我们可以用某种方式让每次播放的音效略有不同就好了。
为此，我们需要使用 Sound Cue。

什么是 Sound Cue

Sound Cue 是一种游戏资源形式，可以让开发者控制多种声音，并将其整合在一起。然后我们可以在需要的地方使用 Sound Cue，而非原始的声音。
下面这个例子演示了如何将三种声音连接起来，以便连续播放：



如果这里使用的是 Random 随机节点，那么每次播放 Sound Cue 的时候就会选择一种随机的声音来播放。



在本教程中，我们将创建并使用 Sound Cue，以便更改声音的音调。

创建一个 Sound Cue

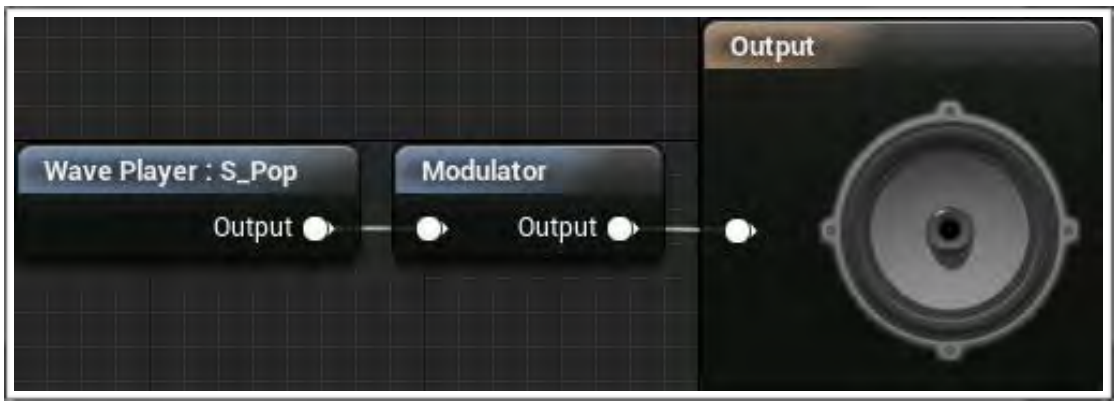
首先打开 Audio 文件夹，我们将使用 S_Pop 作为创建 Sound Cue 的基础声音。
为此，右键单击 S_Pop，然后选择 Create Cue。

这样我们就创建了一个新的名为 S_Pop_Cue 的 Sound Cue。双击在 Sound Cue 编辑器中将其

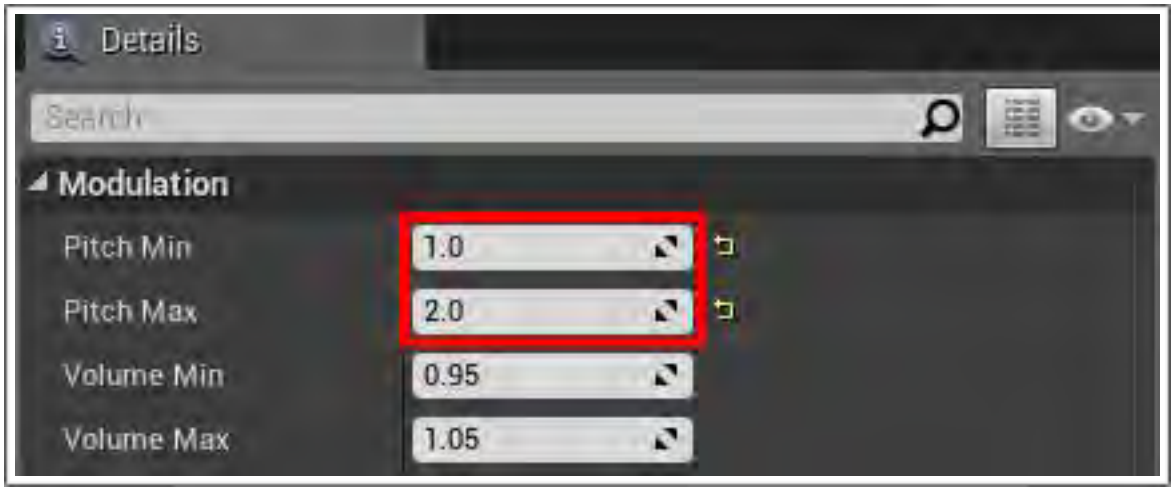
打开：
你会发现这个编辑器跟材质编辑器非常类似，这里就不再详细介绍每一个功能区的作用了。

在视图区我们可以看到两个节点：Wave Player:S_Pop 和 Output。Sound Cue 将会播放连接到 Output 节点的音乐。在这里显然会播放 S_Pop 音乐。我们可以点击工具栏上的 Play Cue 按钮预览声音效果。
接下来我们来了解如何更改声音的音调。

更改声音的音调



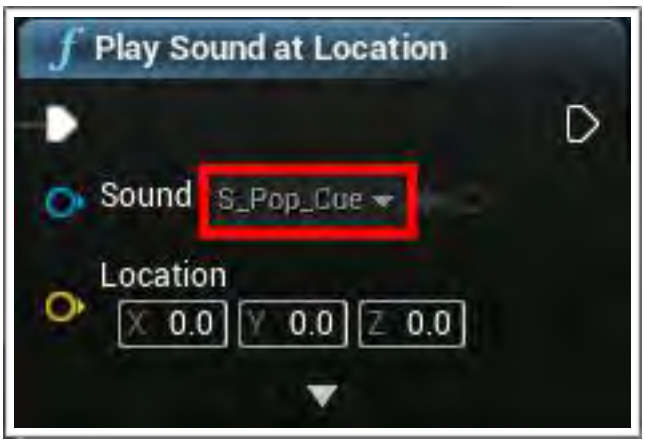
为了更改某个声音的音调，我们需要用到 Modulator 节点。在视图中创建一个 Modulator 节点，并使用如下的方式创建关联：
接下来我们需要定义声调的变化范围。选中 Modulator 节点，然后在 Details 面板中会看到跟声调相关的两个参数：Pitch Min 和 Pitch Max。大于 1 的数值表示声调可以被降低，等于 1 则表示声



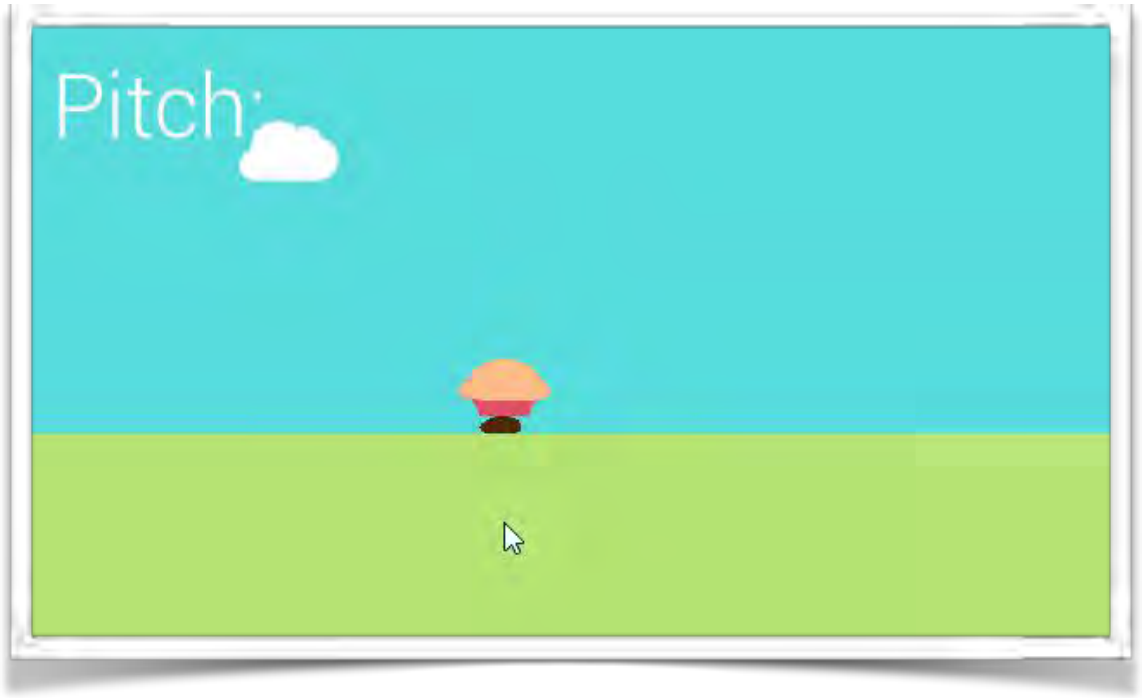
调将保持不变。
对于本教程来说，声调应该被调高。为此，我们将这里的 Pitch Min 设置为 1.0，将 Pitch Max 设置为 2.0。
现在，每当我们播放 Sound Cue 的时候，最终所播放声音的音调会介于原始声调和双倍声调之间
接下来我们需要在玩家触碰到某个云朵的时候播放 Sound Cue。

播放 Sound Cue

返回主编辑器，找到 Blueprint 文件夹。打开 BP_Cloud，然后双击打开 CloudTouched 函数。
该函数将在玩家触碰到云朵的时候执行，因此也是播放 Sound Cue 的理想时点。
我们可以使用两类节点来播放声音：
1.Play Sound 2D:简单来说，就是不考虑任何衰减和空间变化，对于背景音乐和 UI 界面音效适合这一类节点。
2.Play Sound at Location:播放 3D 场景中特定位置的音效。如果我们希望基于玩家角色的位置和朝向来播放音乐，就需要选择这一类节点。



因为云朵存在于游戏世界之中，因此对应的音效也应该存在于游戏世界之中。为此，我们需要在节点链的最后选择 Play Sound at Location 节点。



接着将节点中的 Sound 设置为 S_Pop_Cue。
好了，现在每当玩家触碰到云朵的时候，都会播放 S_Pop_Cue。
点击工具栏上的 Compile 按钮，然后返回主编辑器。点击 Play 按钮开始玩游戏。
每当你碰到某个云朵的时候，都会听到随机声调的声音。

好了，本课的内容就到此结束了。
我们下一课再见~

讨论群-笨猫学编程 QQ 群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

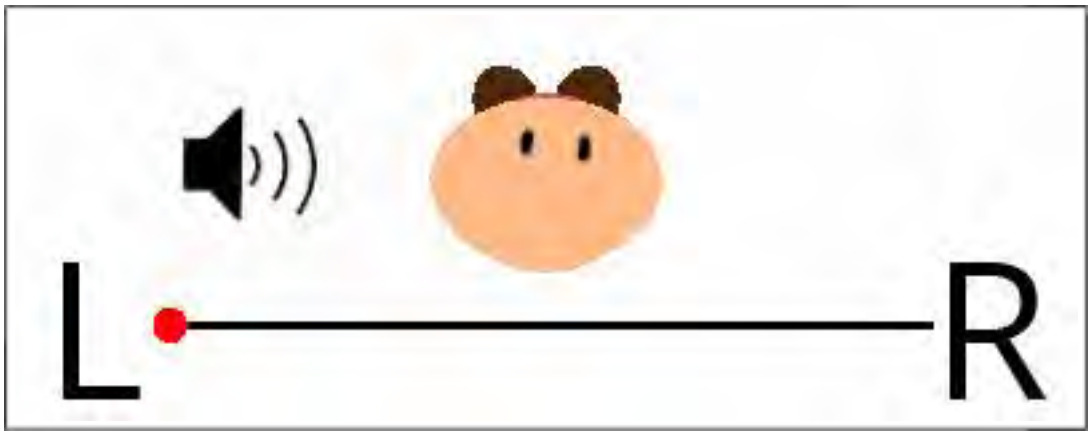
新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。
现在声调已经发生了变化，但是听起来不像是 3D 空间里的声音。为此，我们需要将当前声音空间化（spatialize)。

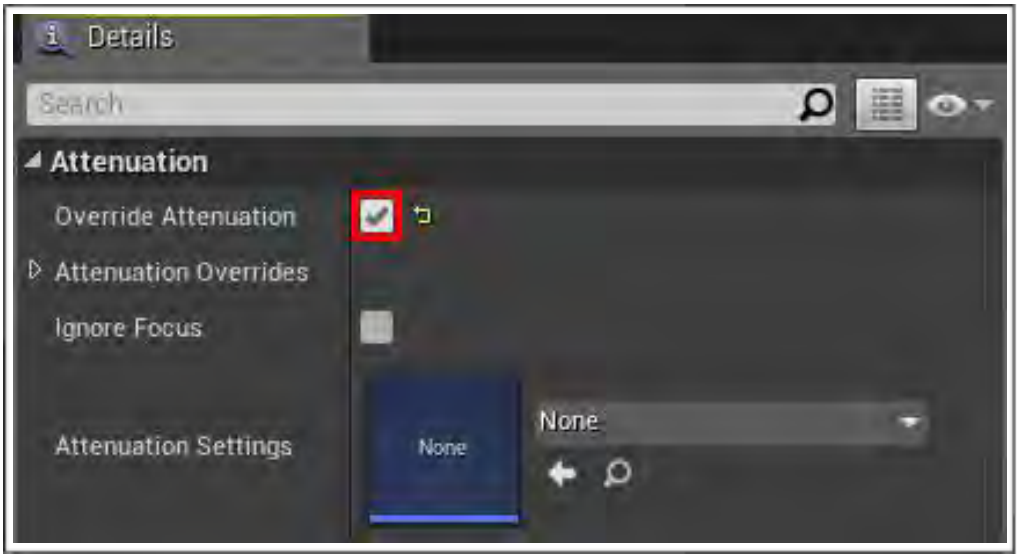
什么是 Spatialization（空间化）
使用 spatialization，可以让声音听起来像是在真实的 3D 空间里。
从左侧发出的声音会被左耳接收，反之则会被右耳接收（前提是戴上耳机~）。



除了增加游戏的沉浸感，空间化还可以提供游戏内的辅助。在竞技类游戏如守望先锋和 CS 中，空间化的音效可以帮助玩家识别其它玩家的位置。
在本教程中，我们将利用空间化，根据云朵的位置来调整音效。

启用空间化

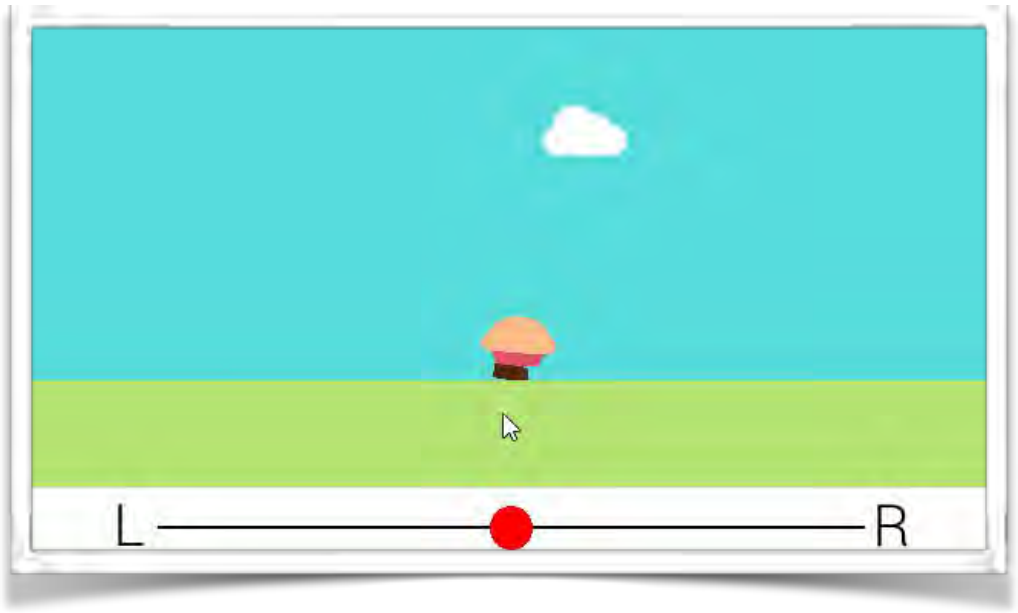
有两种方式可以启用对 Sound Cue 的空间化：
1.Sound Attenuation（声音衰减）
该资源中包含了对衰减和空间化的相关设置。开发者可以将该资源设置给不同的声音，从而确保它们有相同的设置。
2.Override Attenuation(覆盖衰减）
除了使用 Sound Attenuation，我们还可以在 Sound Cue 中指定设置。使用 Override Attenuation 可以为不同的 Sound cue 创建设置。
在本教程中，我们将使用第二种方式。
打开 S_Pop_Cue，然后在 Details 面板中找到 Attenuation 部分，并启用 Override Attenuation。这样将启用 Attenuation Distance 和 Attenuation Spatialization 部分。



在 Attenuation Spatialization 中确保勾选 Enable Spatialization。
接下来我们需要需要指定声音在 3D 空间中的位置。

在 3D 空间中播放声音
打开 BP_Cloud，然后创建一个 GetActorLocation 节点。然后使用如下的方式将其连接到 Play Sound at Location 节点的 Location 端口。

好了，现在声音将在云朵的相同位置播放。
点击 Compile 按钮，然后返回主编辑器。
点击 Play 按钮，并触碰云朵。
此时，你可以听到来自不同位置的声音~



需要注意的是：
默认情况下，摄像机是 audio listener。也就是说我们所听到的声音是以摄像机的视角为依据。
如果我们希望更改 audio listener，可以使用 Set Audio Listener Override 节点。

好了，本课的内容就到此结束了。
我们下一课再见~

讨论群-笨猫学编程 QQ 群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

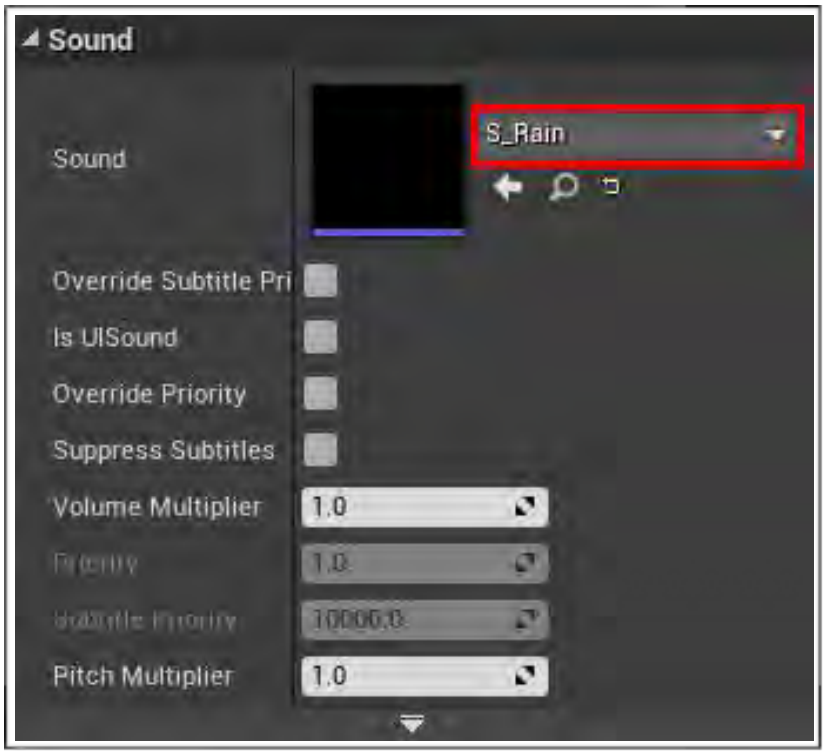
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。
你可能已经注意到了，有些云朵上面有雨，但是如果听不到下雨的声音，就很难让人感觉到在☂ 。
接下来，我们将添加雨声，并使用 attenuation，根据距离的远近来更改雨声大小。

添加雨声

为了播放下雨的声音，不一定要使用节点，这里我们可以考虑使用 Audio 组件。使用组件的好处是，它会自动在云朵的位置播放声音。
打开 BP_Cloud，然后在 Components 面板中添加一个新的 Audio 组件，将其命名为 RainAudio。
选中 RainAudio 组件，在右侧的 Details 面板中找到 Sound 部分，将 Sound 更改为 S_Rain。
对于普通的云朵，显然不应该播放雨声。为此，我们需要对普通的云朵禁用 RainAudio。



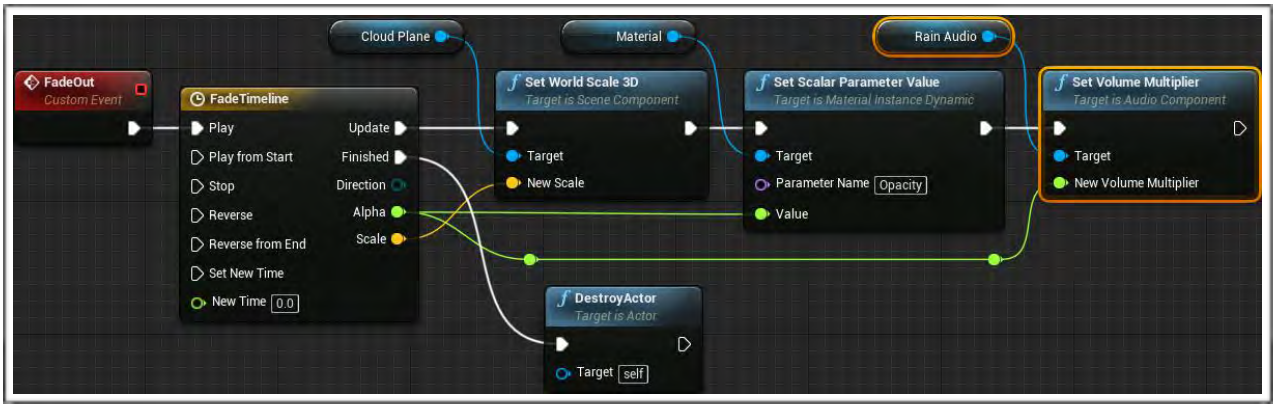
为此，在 Details 面板中找到 Activation 部分，并取消勾选 Auto Activate。
接下来，我们需要对雨云手动启用 RainAudio。

我们可以在 EnableRain 函数中进行 giant 设置。当某个云朵是雨云的时候，就会执行该函数。
打开 EnableRun 函数，并添加下图中的高亮节点：

接下来我们需要启用 attenuation(衰减），并进行相关的设置。

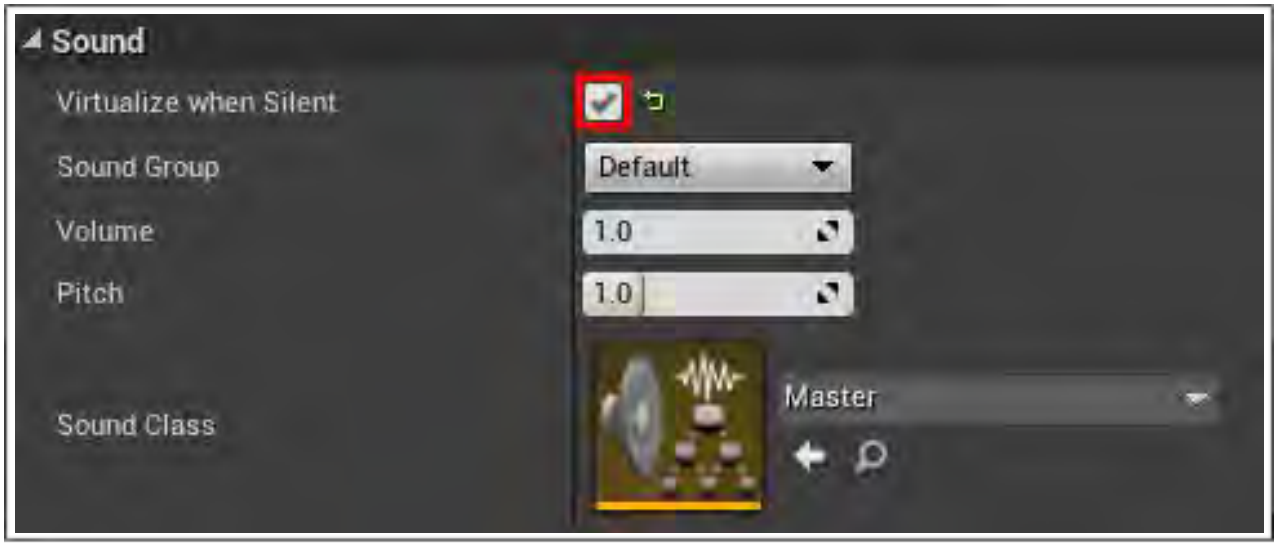
设置 Attenuation
在蓝图编辑器的 Components 面板中找到并选中 RainAudio。在 Details 面板中找到 Attenuation 部分，启用 Override Attenuation 设置。
在 Attenuation Distance 部分，有两个参数需要设置。
其中一个是 Radius,也就是声音开始衰减之前的最大距离。
另外一个是 FalloffDistance，也就是声音衰减到零的距离。
在这里，将 Radius 设置为 300，Falloff Distance 设置为 400。
这就意味着当玩家离音源的距离在 300 个单位以内时，音量是 100%。当玩家离音源的距离在 700 个单位（300+400）时，音量会降到 0。
如果当云朵消失的时候玩家还没有走出衰减范围，那么声音就会突然断掉。为此，我们需要设置声音的逐渐消失。

让声音逐渐消失
切换到 Event Graph 视图，然后找到 FadeOut 事件。操作的方式是在 My blueprint 面板中找到 Graphs 部分，然后双击 EventGraph 下列出的 FadeOut。
在节点链的最后添加下图中的高亮节点：

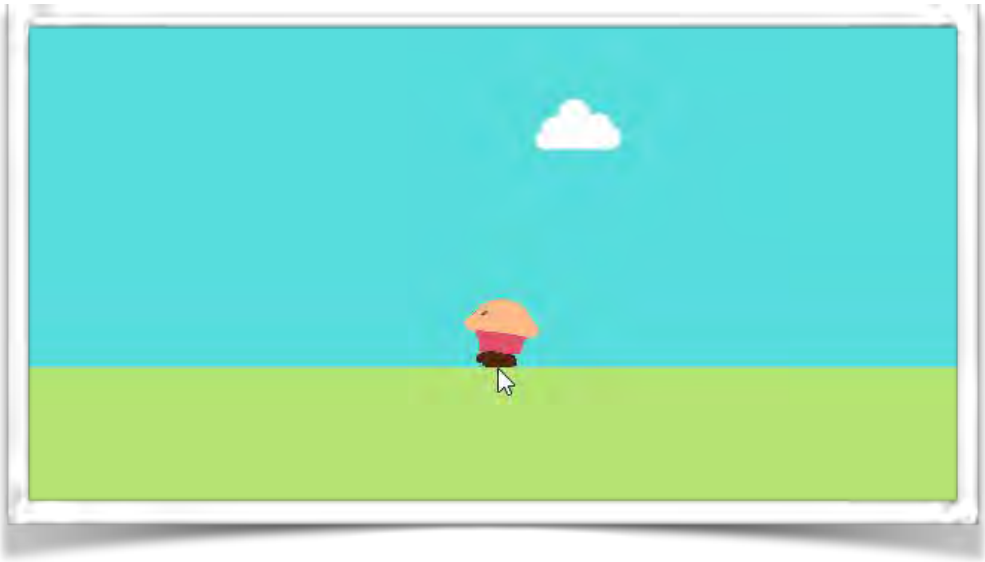


此时，当玩家触碰云朵的时候，就会执行 FadeOut。Timeline(FadeTimeline)节点会输出一个介于 0 和 1 之间的数值。通过使用该数值，就可以让 RainAudio 的音量产生渐变。

最后还有一处需要设置的。当某个声音的音量是 0%时，就会停止播放。而因为在声音开始的时候玩家在可以听到声音的范围之外，S_Rain 的音量就是 0%。此时，当我们进入可以听到声音的范围内时，也不会听到声音。
我们可以使用 Virtualize when Silent 设置来修复这个问题。使用该设置，不管音量如何，都会播放声音。
点击 compile 按钮，然后关闭 BP_Cloud。



找到 Audio 文件夹，并打开 S_Rain。在 Sound 部分启用 Virtualize when Silent。
关闭 S_Rain，返回主编辑器。
点击 Play 按钮，然后移动到雨云的范围内，看能否听到下雨的声音~



好了，本课的内容就到这里了。
我们下一课再见~

讨论群-笨猫学编程 QQ 群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

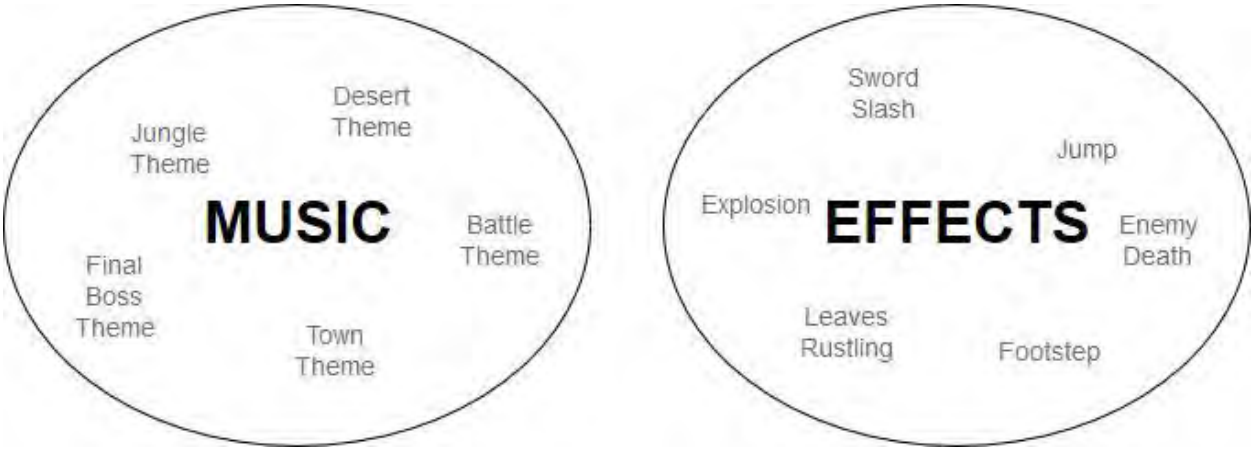
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。
在这部分课程的最后，我们将学习如何使用 Sound Classes 和 Sound Mixes 来控制音量。

Sound Classes 和 Sound Mixes

使用 Sound Class 可以轻松管理多个声音。比如，我们可以把所有的背景音乐放到一个类中，然后所有的音效放到另外一个类中。



为了在游戏的过程中调节 Sound Class 的属性（音量，音调等），我们需要使用 Sound Mix。Sound Mix 其实是一个表格，其中的每一条记录都是一个 Sound Class。每一条记录中都包含了 Sound Class 应该有的属性调整，比如下图就是一个示例：

Sound Class	Volume	Pitch
Music	0.5	1.0
Effects	1.0	2.0

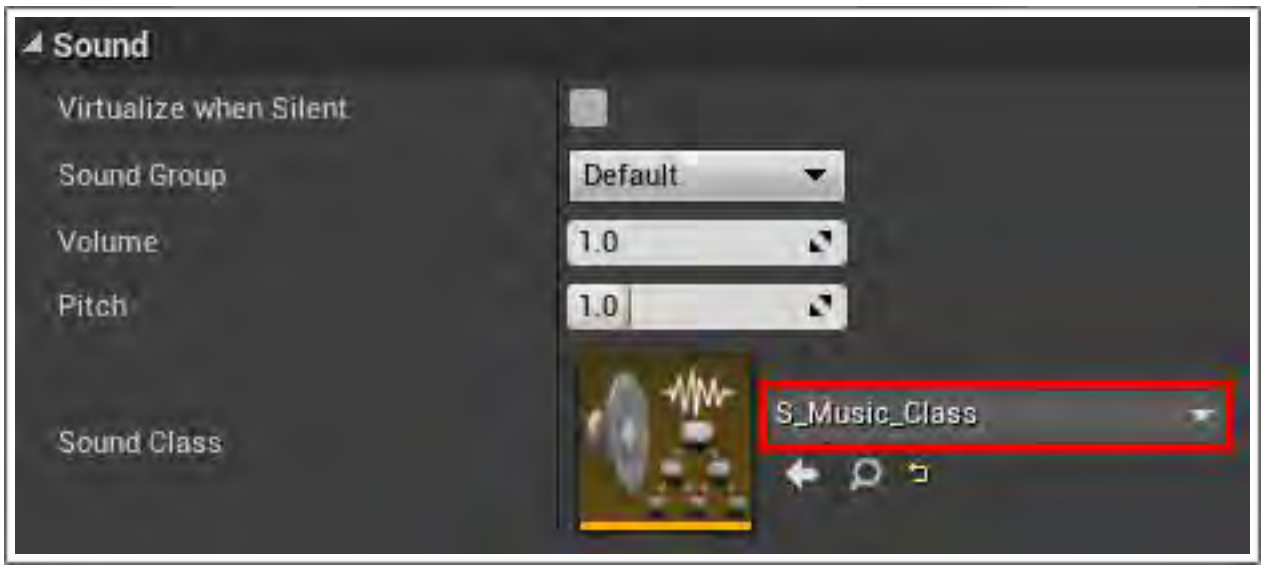
通过使用上面的 Sound Mix，Music 类中的每个声音都将以一半的音量进行播放，而 Effects 类中的声音的声调都会调整为 2 倍。

接下来首先我们需要创建 Sound Class

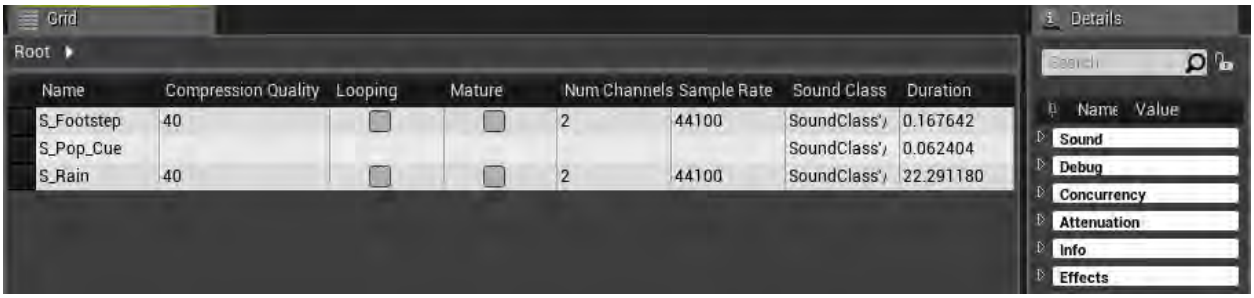
创建 Sound Class
在本教程中，我们将分别调整背景音乐和音效的音量。为此我们需要创建两个 Sound Class。在 UE4 主编辑器的 Content Browser 中，点击 Add New，选择 Sounds\Sound Class。将新创建的 Sound Class 命名为 S_Music_Class。

使用同样的方式创建另一个 Sound Class，并将其命名为 S_Effects_Class。

接下来我们需要给每个声音设置为一个 Sound Class。
首先让我们来设置背景音乐。
打开 S_Music，然后找到 Sound 部分，将 Sound Class 更改为 S_Music_Class。



完成后关闭 S_Music。
接下来要给音效分配 Sound Class。不过我们没必要一个个打开音效文件，在 Content Browser 中选中以下资源：
S_Footstep
S_Pop_Cue
S_Rain
右键单击所选资源之一，然后选择 Asset Actions\Bulk Edit via Property Matrix。
此时将打开类似下图的属性编辑器：

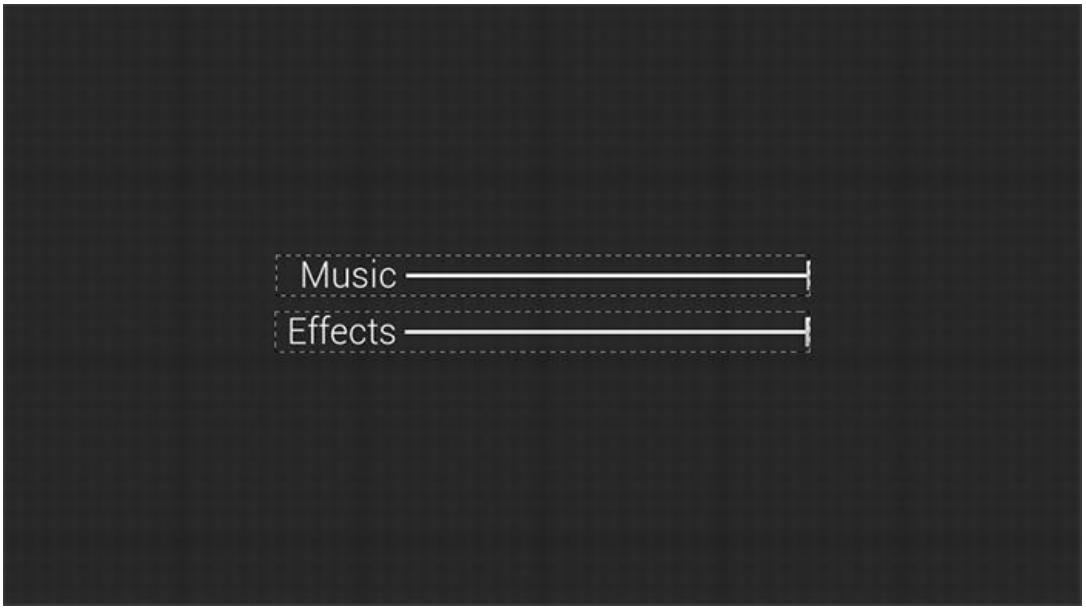


使用以上的属性编辑器可以让我们同时编辑多个音效文件的属性。
在 Details 面板中找到并展开 Sound，点击 Sound Class 右侧的灰格图标。

选择 S_Effects_Class，然后关闭属性编辑器。
现在所有的声音都被分配到了合适的 Sound Class。
接下来我们需要创建一个 Sound Mix，并使用蓝图来调整它。

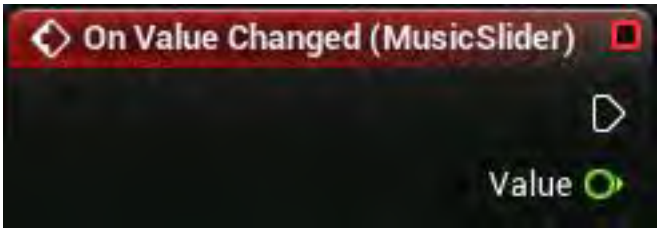
创建和调整 Sound Mix

在 Content Browser 中，点击 Add New，然后选择 Sounds\Sound Mix，将新创建的 Sound Mix 命名为 S_Volume_Mix。
为了控制每个 Sound Class 的音量，我们可以使用滑动条。这里已经提前创建了一个带有两个滑动条的 widget。
在 UI 文件夹中打开 WBP_Options。



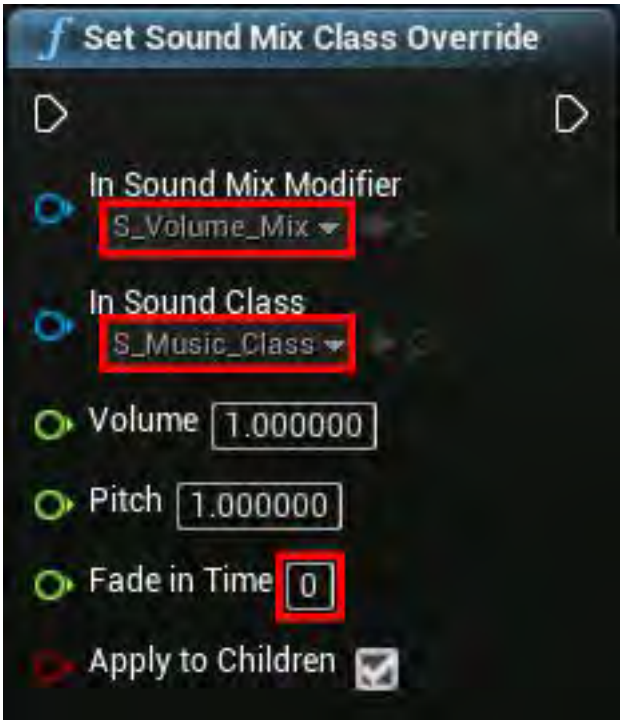
为了调整音量，我们需要用到滑动条的数值，并反馈给 Sound Mix。
首先让我们来使用这种方式尝试控制背景音乐。
切换到 Graph 模式，然后在 My Blueprint 面板中的 Variable 部分选中 MusicSlider。
在 Details 面板中点击 On Value Changed 旁边的按钮。

这样就创建了 On Value Changed(MusicSlider)事件，每当我们移动滑动条的时候都会触发该事件。

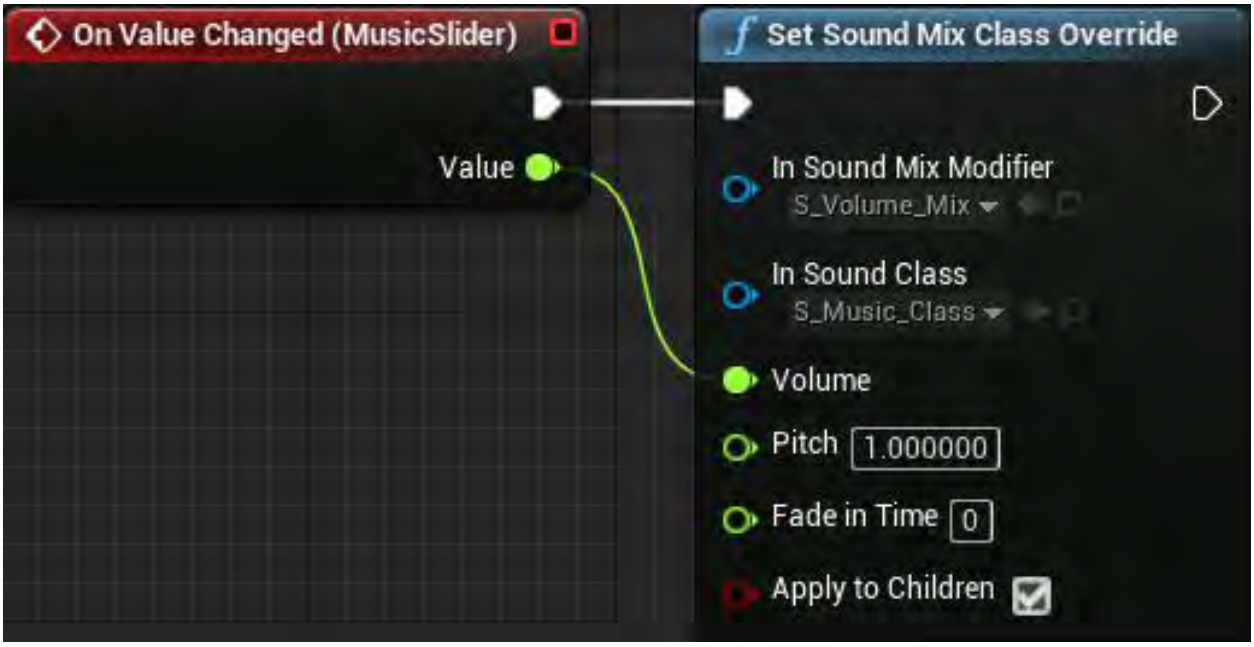


接下来我们需要在 S_Volume_Mix 中设置 S_Music_Class。为此，我们需要使用 Set Sound Mix Class Override 节点。该节点允许我们指定 Sound Mix 和 Sound Class。如果 Sound Class 不在 Sound Mix 中，就会添加该 Sound Class。如果已经在 Sound Mix 中，那么就会更新 Sound Class。
在视图中添加 Set Sound Mix Class Override 节点，并设置以下选项：

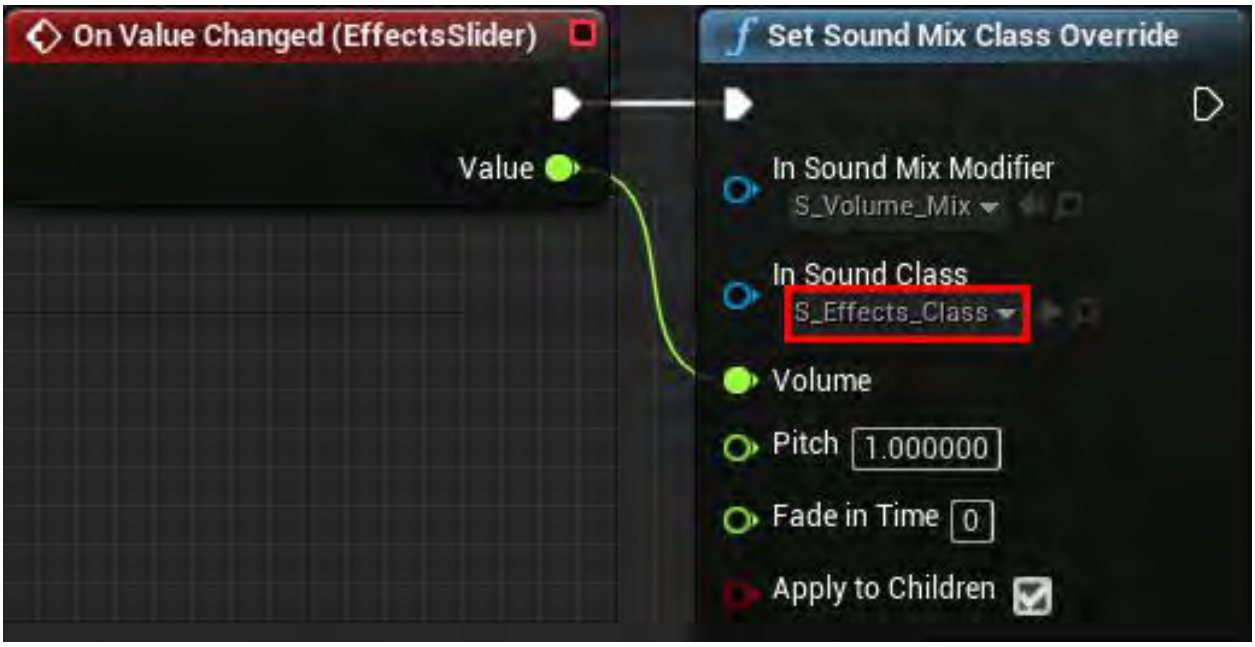
In Sound Mix Modifier: S_Volume_Mix
In Sound Class: S_Music_Class
Fade in Time: 0（确保音量调节实时生效）



接下来使用如下方式连接节点：



接下来对 EffectsSlider 做类似的操作，将 In Sound Class 端口中的值更改为 S_Effects_Class。

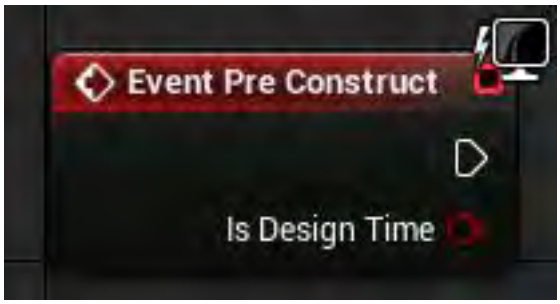


现在只要滑动条的数值发生变化，S_Volume_Mix 就会自动调节所对应的 Sound Class 的音量。不过在一切生效之前，我们还需要激活 Sound Mix。

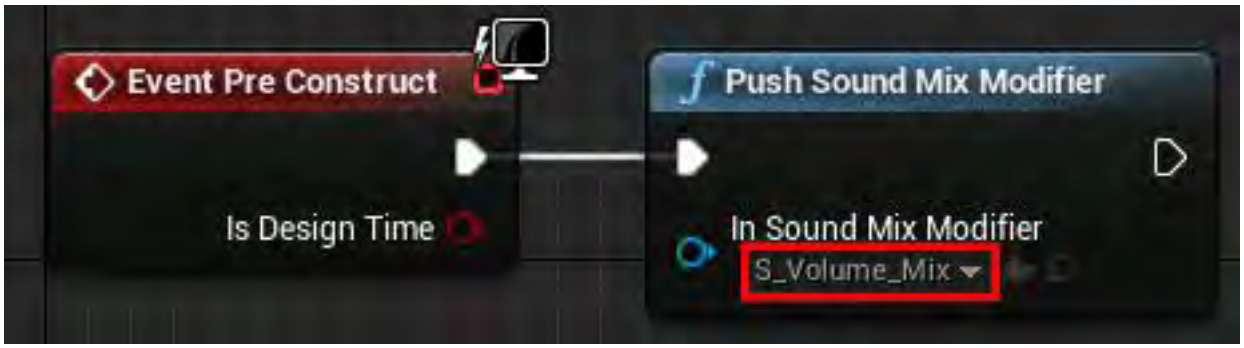
激活 Sound Mix

对于使用 UI 界面调节音量大小的游戏，最好在游戏开始的时候就激活 Sound Mix。这样 Sound Class 就会自动使用 Sound Mix 中的音量调节结果。不过为了简化起见，这里我们在 widget 中激活 Sound Mix。

在刚才的蓝图中创建一个 Event Pre Construct 节点，该节点和 Event BeginPlay 节点类似：



为了激活 Sound Mix，我们需要使用 Push Sound Mix Modifier 节点。创建一个该节点，并将其



连接到 Event Pre Construct 节点上。然后将 In Sound Mix Modifier 设置为 S_Volume_Mix。
这样就会在打开 WBP_Options 时激活 S_Volume_Mix。
点击 Compile,然后关闭 WBP_Options。
点击 Play 按钮启动游戏，然后按下 M 键启用滑动条。试着调节滑动条来影响音量的大小。

好了，本系列的教程就到此结束了。
完整的项目可以参考这里：
链接:https://pan.baidu.com/s/1DLaNj1PgcY5Hq_ql4Bcf6w 密码:9fmx

如果你想了解虚幻 4 引擎中关于声音系统的更多知识，可以参考官方文档：
https://docs.unrealengine.com/en-us/Engine/Audio/Overview#reverbeffects

在下一个系列的教程中，我们将继续带领大家学习如何创建粒子特效。

我们下一课再见~

讨论群-笨猫学编程 QQ 群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

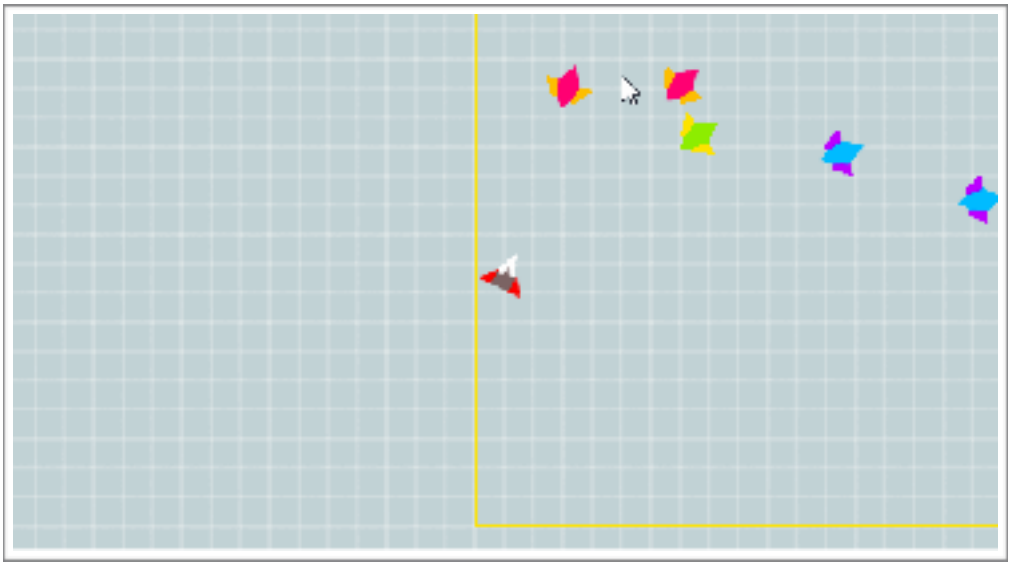
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。
对于游戏的视觉效果来说，粒子系统是非常重要的。使用粒子系统，游戏美术设计师可以轻松创建各种酷炫的效果，比如爆炸、烟雾和下雨等。
虚幻4中提供了一个简单易用且强大的粒子特效制作系统，被称之为Cascade。使用Cascade系统可以创建模块化的效果，并轻松控制粒子的行为。
在本教程中，我们将学习以下内容：
1.创建粒子系统
2.设置粒子的速度和大小
3.调整粒子的生成速度
4.使用曲线设置粒子大小随时间的变化
5.使用Cascade设置粒子颜色
6.使用蓝图激活和停用粒子系统
7.使用蓝图设置粒子颜色

开始前的准备

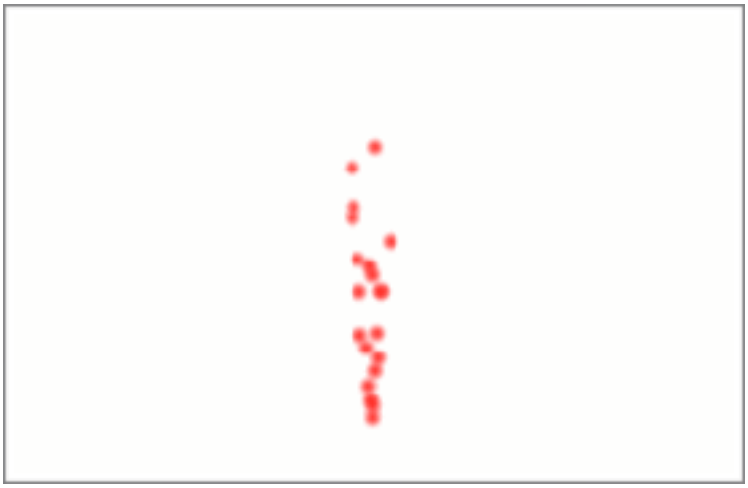
先从这里下载起始项目（链接:https://pan.baidu.com/s/1lPqRqkxHwELibAEY6NF3hA 密码:ny32），然后将其解压缩。打开项目文件夹，并双击打开SpaceshipBattle.uproject。
点击Play体验游戏，使用鼠标左键可以射击，使用W,A,S和D可以四处移动。



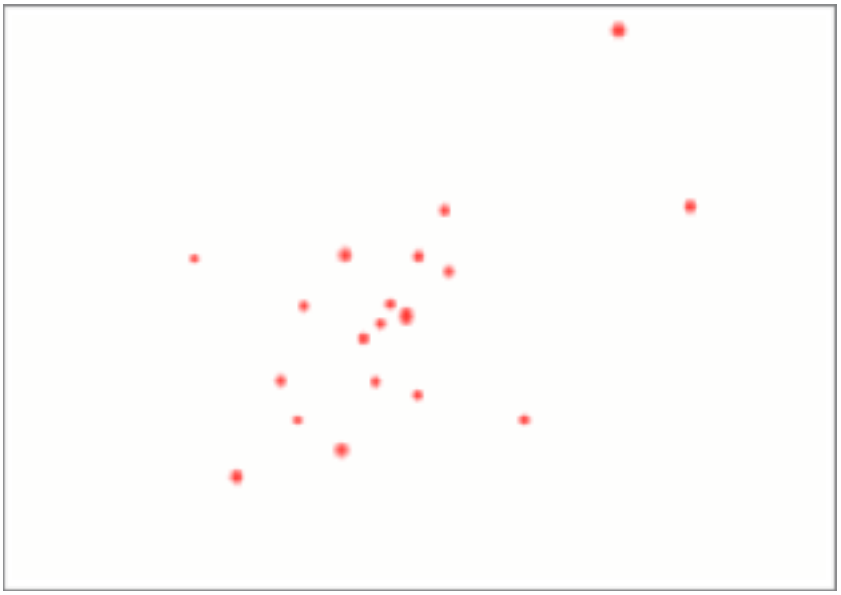
在本教程中，我们将创建两个粒子特效。其中一个针对飞船推进器，而另一个则是当飞船爆炸的时候。为此，我们将需要用到粒子系统。

什么是粒子系统

顾名思义，粒子系统就是用来创建和管理粒子的系统。那么什么是粒子呢？显然这里我们不需要掌握高能物理里面的玄奥知识，这里的粒子可以简单看做空间中的一个点。使用粒子系统，我们可以控制粒子的外观和行为。
粒子系统中包含了一个或多个名为emitters（发射器）的组件，emitter的作用是生成粒子。



emitter中也有组件，被称之为module。使用Module可以控制由emitter生成的例子的各种属性，比如材质和初始速度。在下面的例子中，使用两个module给每个粒子一个红色的圆形材质和随机速度。



我们也可以在粒子的生命周期中更改粒子的颜色。
好了，现在我们已经知道了什么是粒子系统，接下来将为飞船的推进器创建粒子系统。

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

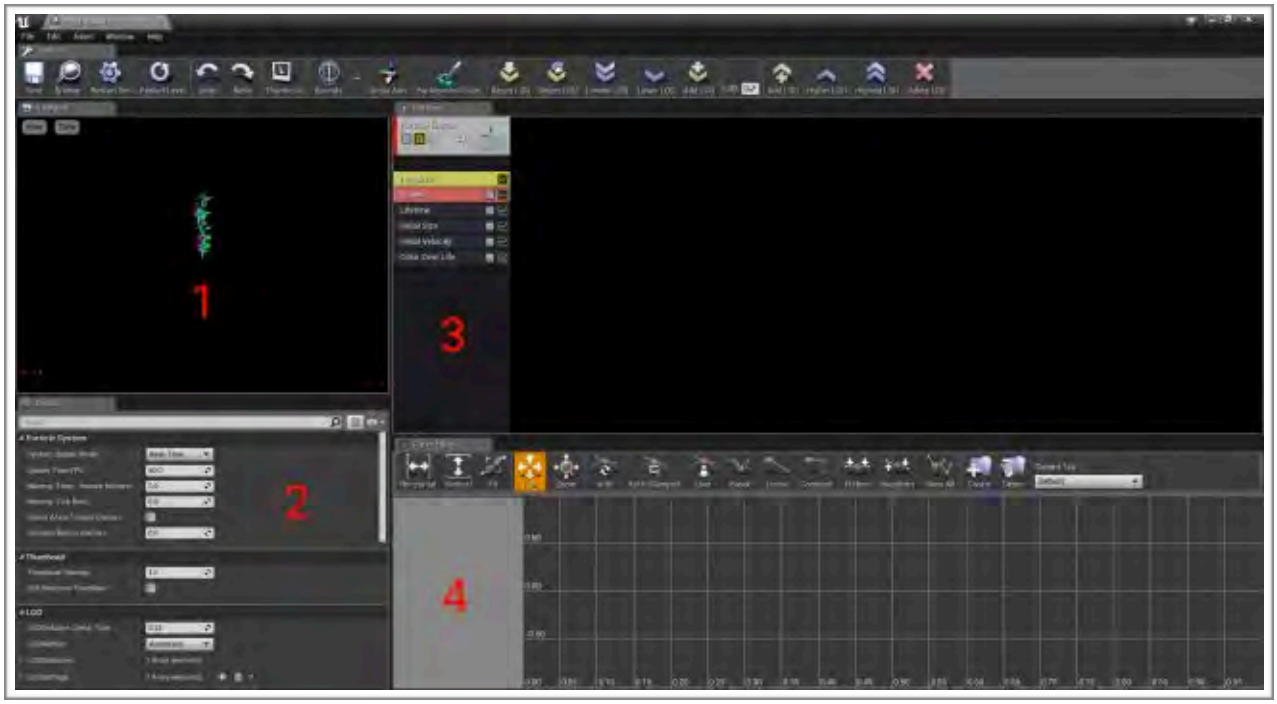
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。
在本课的内容中，我们将一起创建飞船推进器的粒子特效。

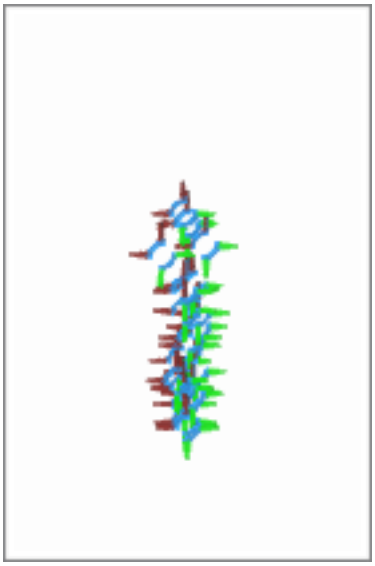
创建粒子系统
打开ParticleSystems文件夹，点击Add New\Particle System，将新建的粒子系统命名为PS_Thruster，然后将其打开。

Cascade:粒子系统编辑器
Cascade是虚幻4引擎中的粒子系统编辑器，它包含了四个主要的面板：



1.Viewport视口：
该面板中会显示粒子系统的预览。开发者可以按住鼠标右键移动鼠标以便查看。按住鼠标右键，使用WASD键即可。
2.Details：
在这里，会显示所选中的任何组件（emitter,module）的具体属性。如果没有选中任何东西，此处会显示粒子系统的属性。
3.Emitters:
该面板显示从左到右的emitter列表。每个emitter都会有一系列的module。
4.Curve Editor:
使用Curve Editor可以可视化的调整module的曲线。需要注意的是，并非所有的module属性都支持曲线调节。

默认情况下，粒子系统将使用默认的粒子材质。

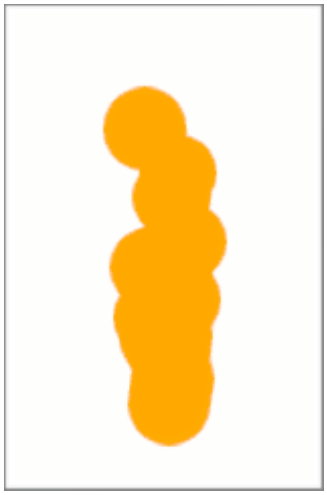


首先我们将使用圆形材质替换默认的粒子材质。

将Material运用到粒子上

在Emitter面板中选中Required module。

Required module中包含了一些必需的属性，比如粒子材质和emitter的生命周期。每个emitter都必须有一个Required module。
为了更改材质，在Details面板中将Material设置为M_Particle。此时粒子的外观会变成橙色的圆形。

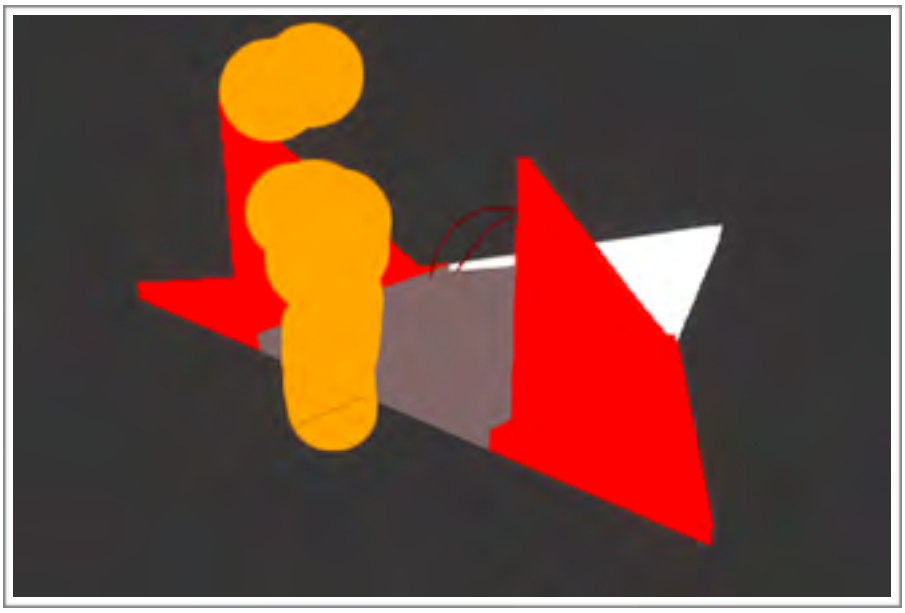


接下来，我们需要将粒子系统关联到玩家的飞船上。

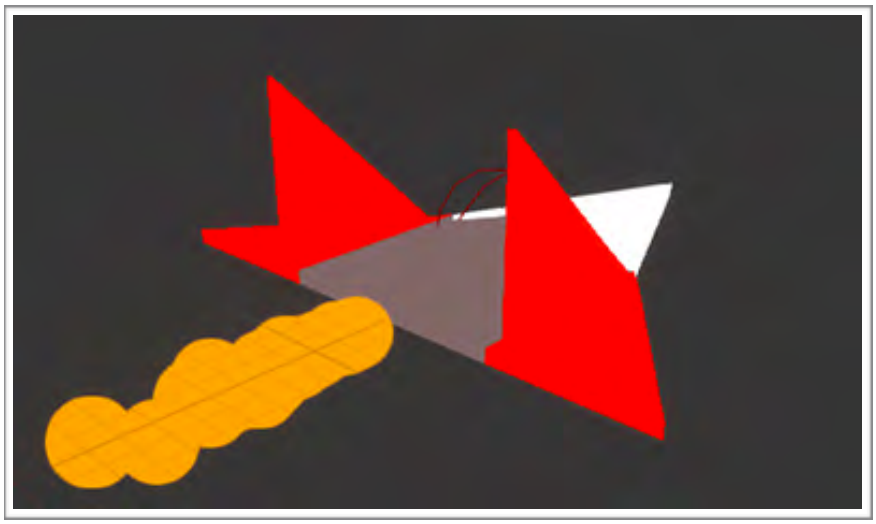
关联粒子系统

返回主编辑器，找到Blueprints文件夹。打开BP_Player，然后找到Components面板。
为使用粒子系统，我们需要使用Particle System组件。在Components面板中创建一个Particle System组件，将其更名为ThrusterParticles。注意需要将其关联到Collision组件上。

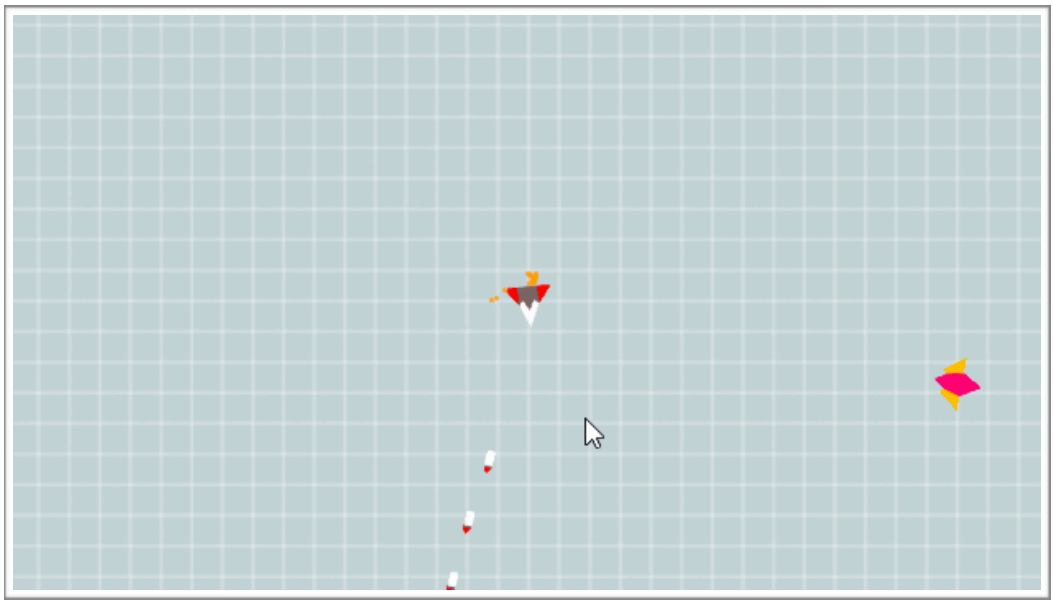
接着在Details面板中找到Particles部分，然后将Template设置为PS_Thruster。
接着将ThrusterParticles的Location设置为（-80,0,0），这样会让粒子显示在飞船的后面。



最后，将Rotation设置为（0，90，0），这样就可以让粒子系统的朝向更加合理。



点击Compile按钮，然后返回主编辑器。点击工具栏上的Play按钮，就可以看到粒子系统生效了。



好了，这一课的内容就到这里了。
我们下一课再见~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

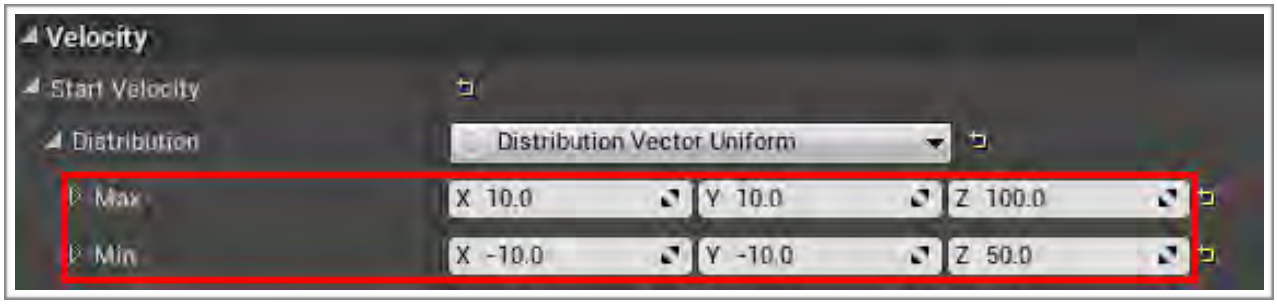
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

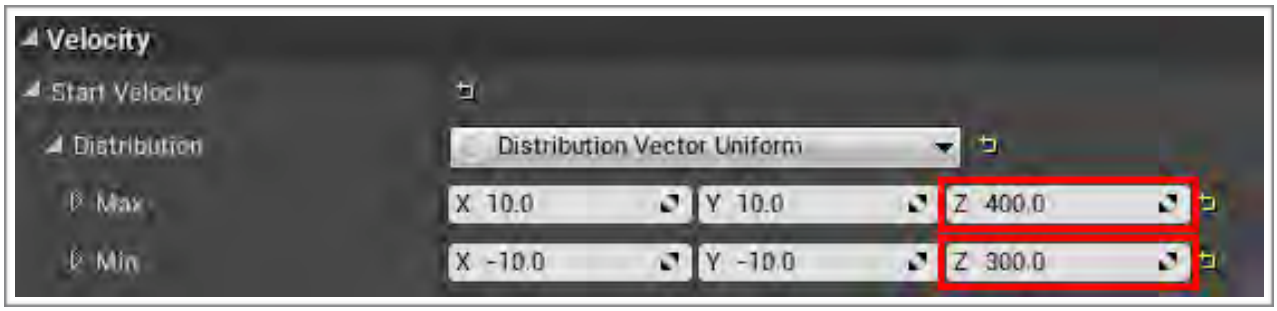
欢迎继续我们的学习。
在上一课的内容中，我们已经实现了基础的粒子特效，但是目前的速度很慢，而且尺寸也很小。为此，我们需要设置粒子的初始速度和大小。

设置粒子的速度和大小

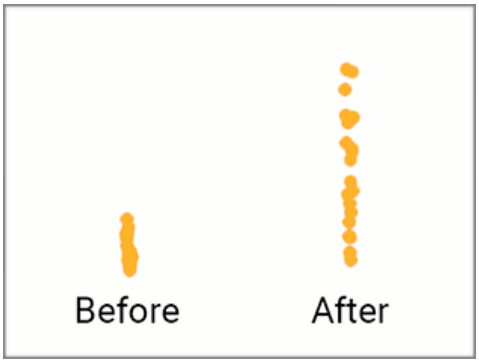
首先，我们一起来设置粒子的初始速度。打开PS_Thruster，选择Initial Velocity。随后展开Start Velocity\Distribution。
默认情况下，粒子的初始速度在(-10,-10,50)到(10,10,100）之间。



为了让粒子从飞船的尾部以更快的速度出现，我们只需要增加Z轴上的速度。为此，将Min Z设置为300，将Max Z设置为400。



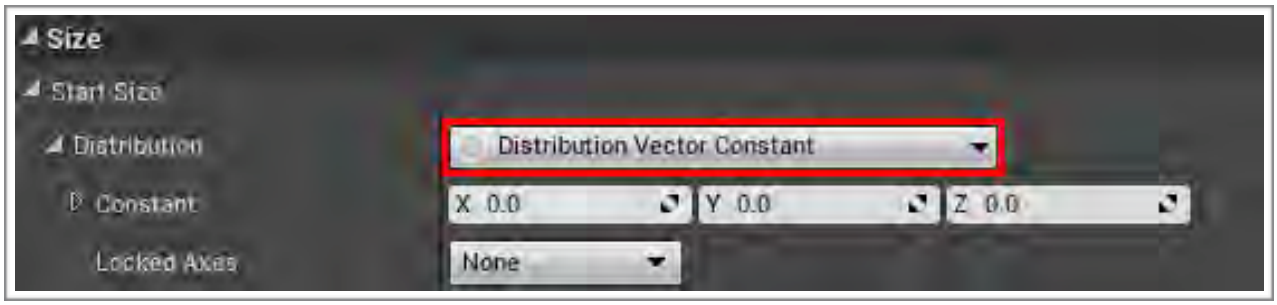
下面是初始速度与调整后的粒子速度对比：



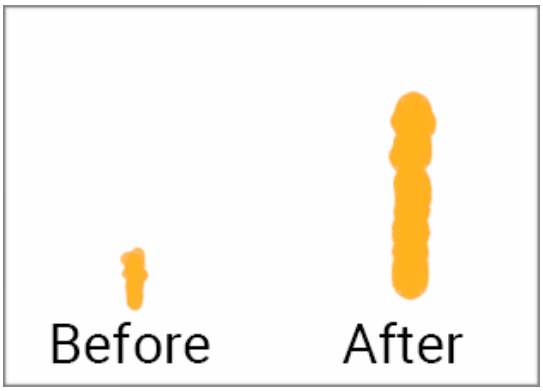
接下来我们来设置粒子的初始大小。

设置粒子的大小

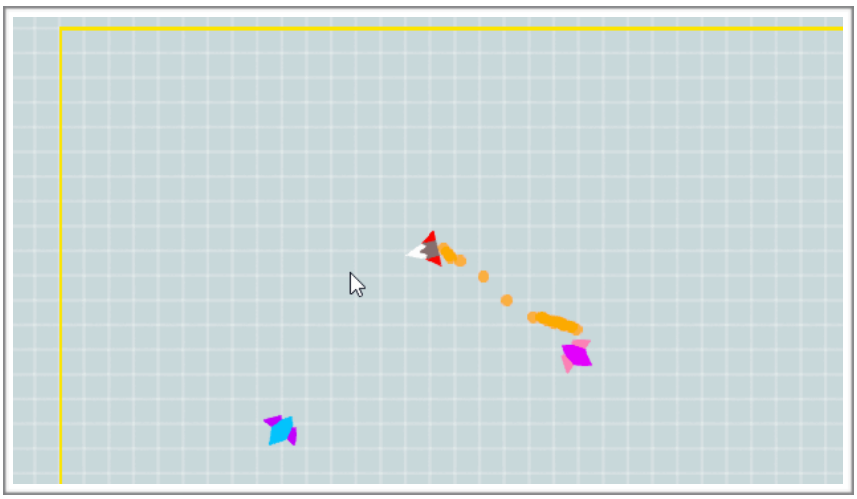
选中Initial Size，然后在Details面板中展开Start Size\Distribution。
和初始速度模块一样，初始大小模块也有个最小和最大范围。不过对本教程来说，我们需要将粒子大小设置为一个常数。为此，将Distribution设置为Distribution Vector Constant。



接着将Constant设置为（70，70，70），下面是粒子大小的前后对比：



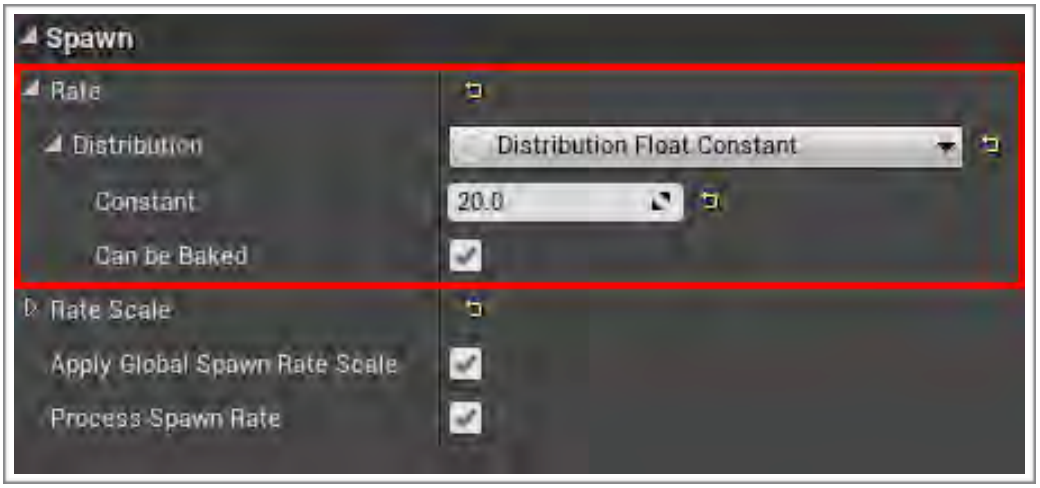
返回主编辑器，点击Play按钮预览效果。



现在粒子效果看起来好一点了，不过还是有点分散。这是因为粒子生成之间的间隔太长，为此我们需要增加粒子的spawn rate。

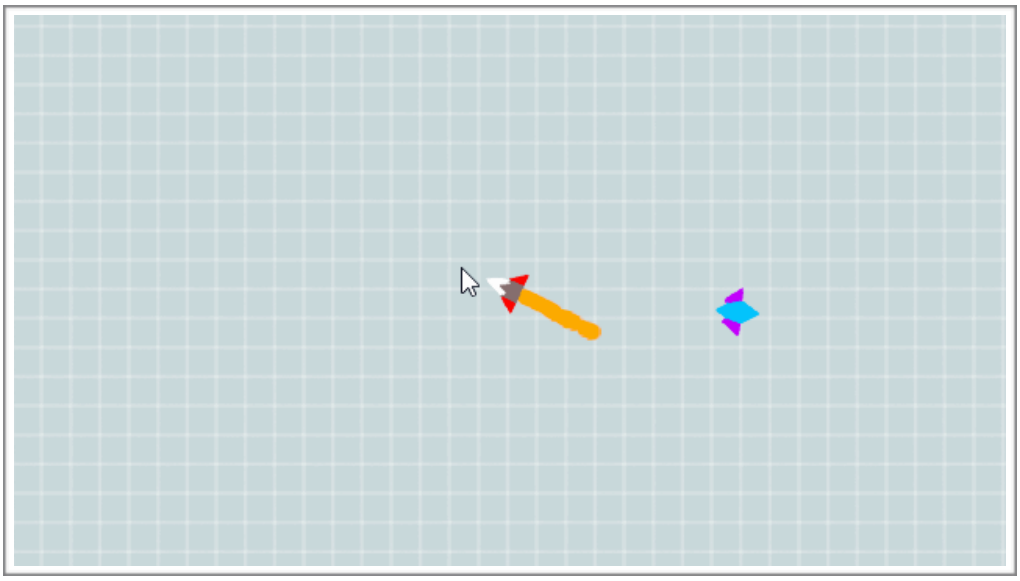
增加粒子的Spawn Rate

为了增加spawn rate，我们需要使用Spawn模块，它可以控制粒子发射器生成粒子的速度。而每个粒子发射器都有一个Spawn模块。
打开PS_Thruster，选择Spawn。在Details面板中展开Spawn\Rate部分。



将Constant常数设置为50，这样就可以让spawn rate增加到每秒生成50个粒子。

返回主编辑器，点击Play按钮预览效果。



可以看到，此时的粒子特效看起来更像是一个轨迹。为了让粒子看起来更像是推进器的火焰，我们需要让粒子生成后逐渐变小。

让粒子生成后逐渐变小

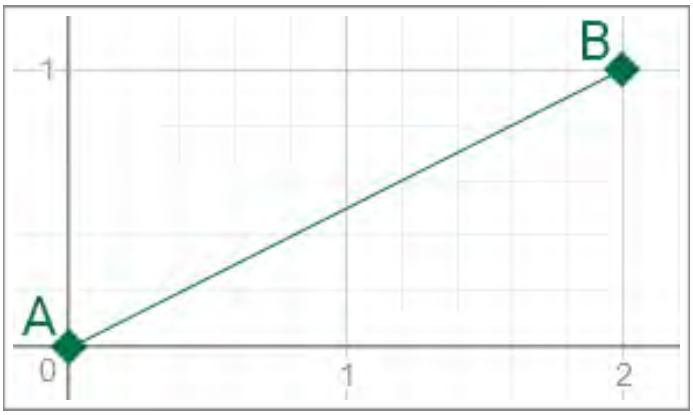
打开PS_Thruster，然后找到Emitters面板。

为了让粒子缩小，我们可以使用Size By Life模块。使用该模块可以让粒子大小在其生命周期中乘以一个系数。
右键单击emitter的空白区域，然后选择Size\Size By Life。

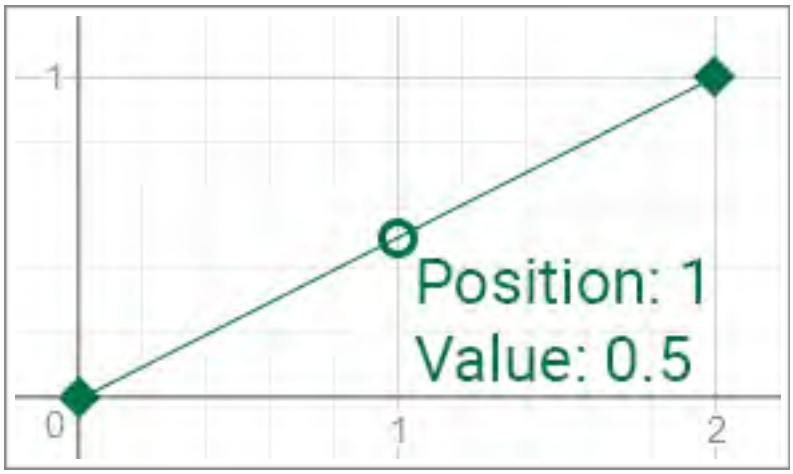
默认情况下，该操作不会影响粒子大小的视觉效果。这是因为当前的系数始终设置为1.为了让粒子变小，我们需要调整模块的曲线，从而让粒子大小在其生命周期中乘以不同的系数。
问题来了，什么是曲线?

什么是曲线

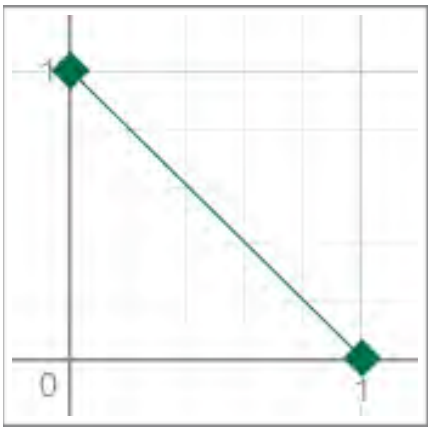
学过中小学数学的同学都知道直线和曲线。
这里的曲线也没什么区别，是点的组合。只不过每个点都有两个属性：位置和数值。
当我们有两个或多个点的时候，就形成了一条线。



如果我们在直线之间的任一位置取样，那么就叫作”线性插值“。听起来很恐怖，其实很简单。
比如在上面的曲线中，如果我们在位置1取样，那么就会得到数值0.5。



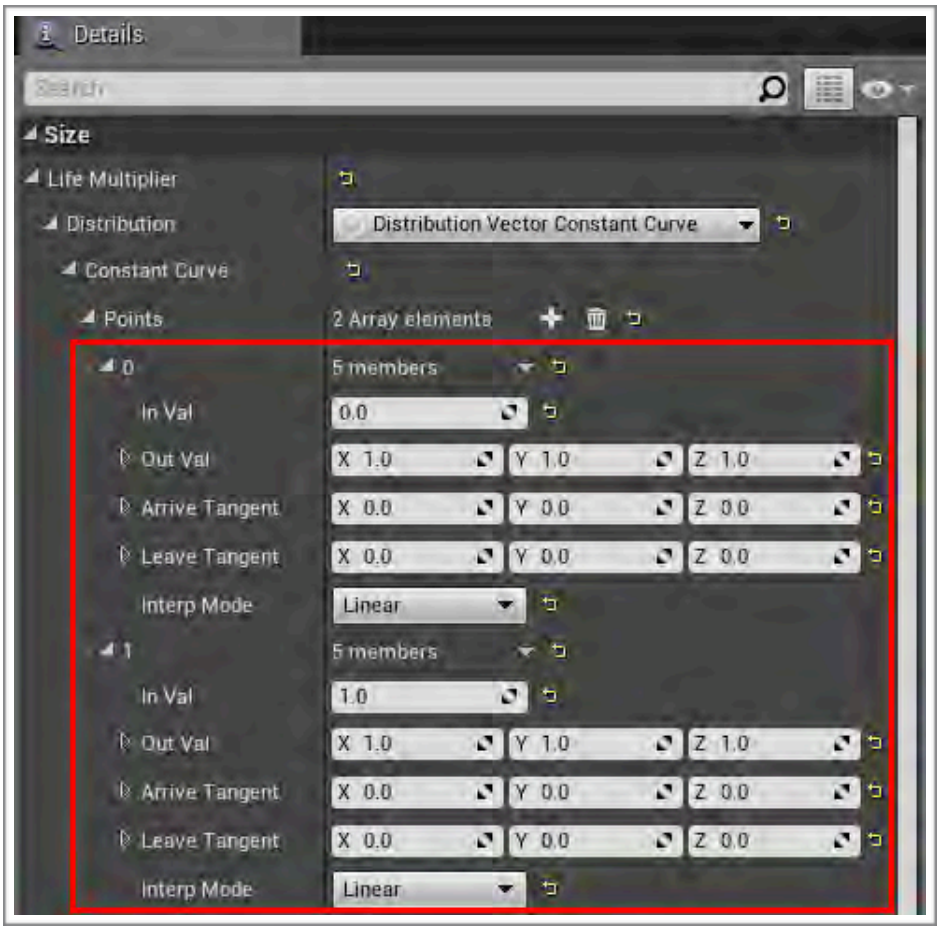
对于Size By Life模块来说，我们需要创建一个随着位置增加数值减少的曲线。



接下来我们将在Cascade中创建以上曲线。

调整模块的曲线

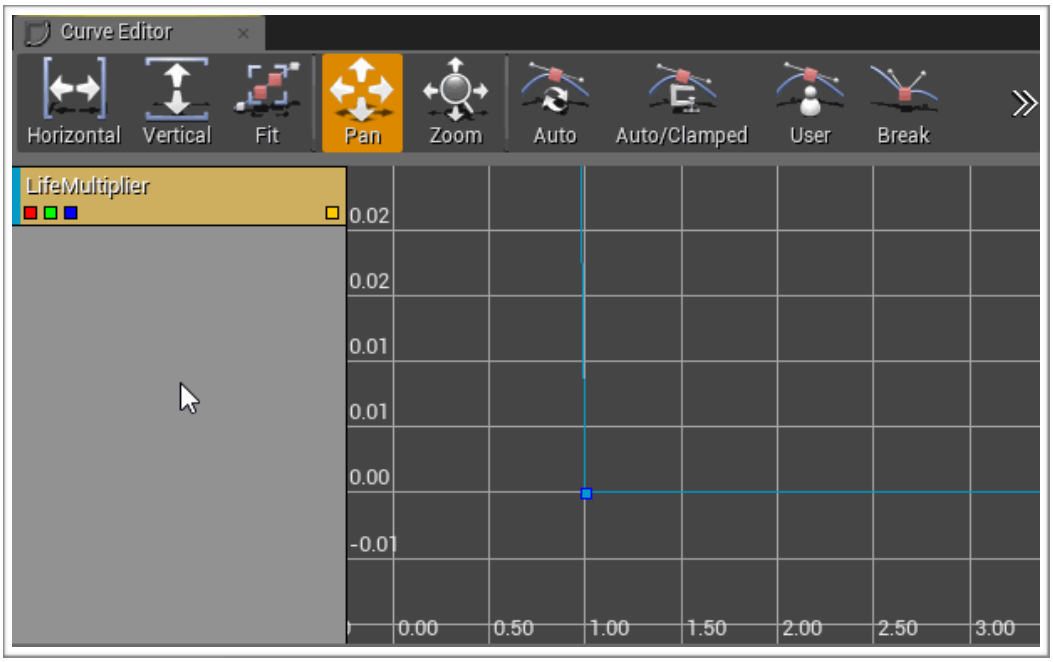
选择Size By Life，然后在Details面板中展开Life Multiplier\Distribution\Constant Curve\Points。这里我们将看到Life Multiplier曲线的一系列点。



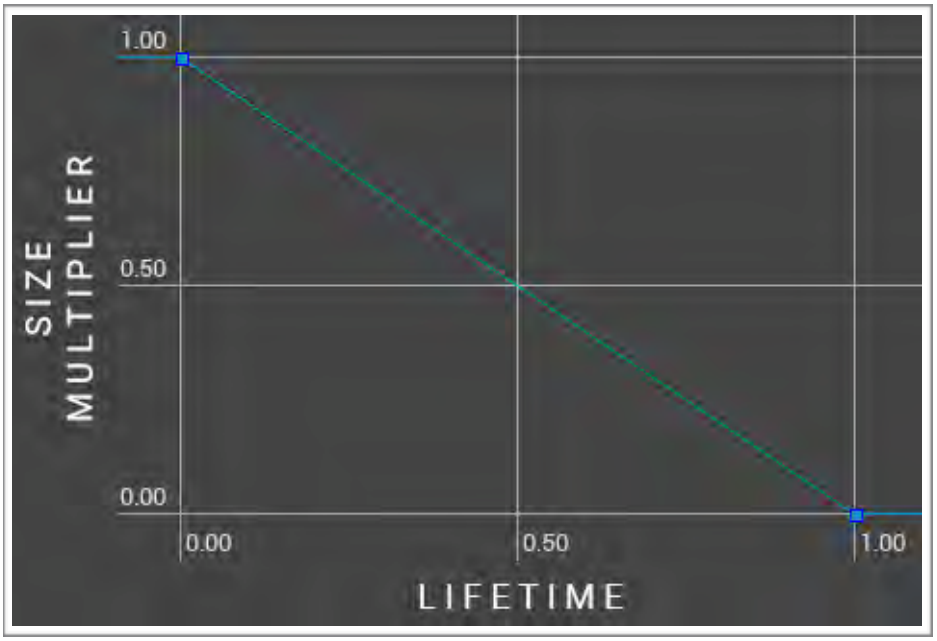
In Val就是曲线上点的位置。对于Size By Life模块来说，0代表粒子生命周期的起点，而1则代表粒子生命周期的终点。
为了让粒子的大小系数随时间变小，我们需要减少第二个点的Out Val值。将点1的Out Val设置为（0，0，0）。这样就可以让粒子大小随生命周期逐渐变小到消失。

我们可以使用Curve Editor来可视化Life Multiplier。为此，点击Size By Life模块的graph图标。

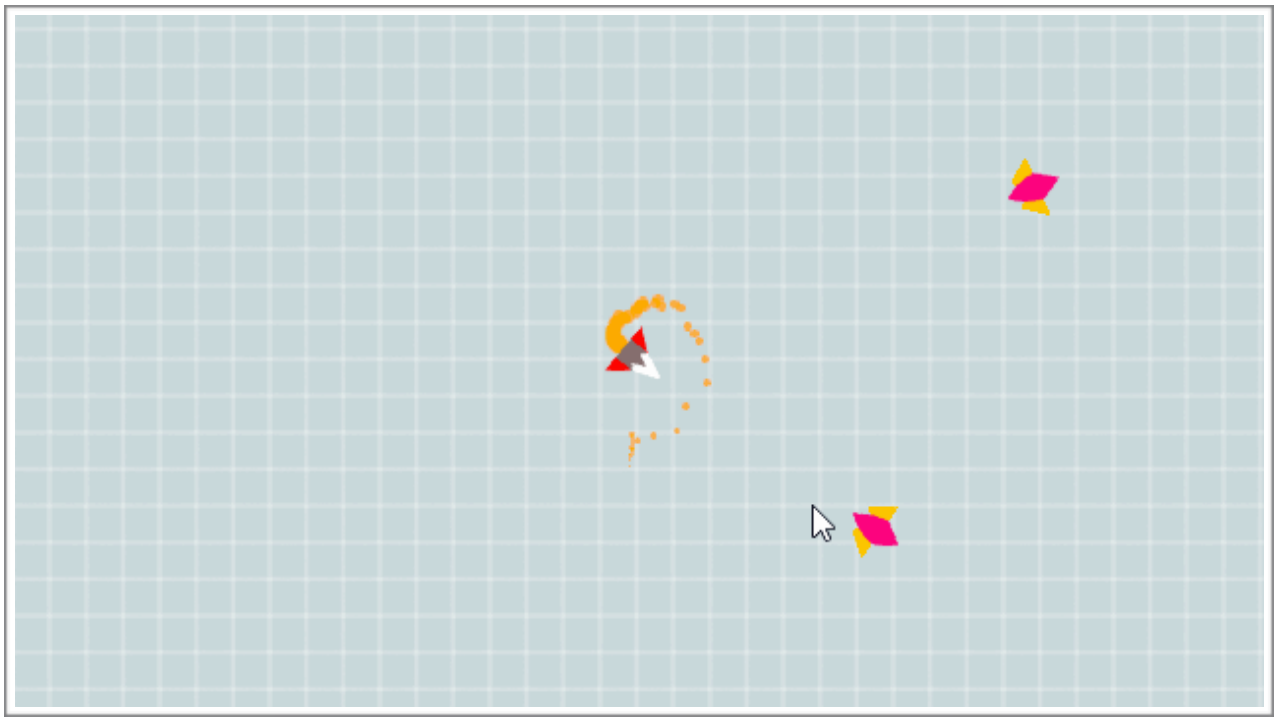
这样就可以将Life Multiplier添加到Curve Editor。为了让曲线显示合理，点击Curve Editor工具栏上的Fit按钮。



可以看到，粒子大小的系数size multiplier随着粒子生命周期从1逐渐变为0。



返回主编辑器界面，然后点击Play按钮。



好了，现在粒子效果看起来有点像是🔥了。
这一课的内容就到这里了，我们下一课再见~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

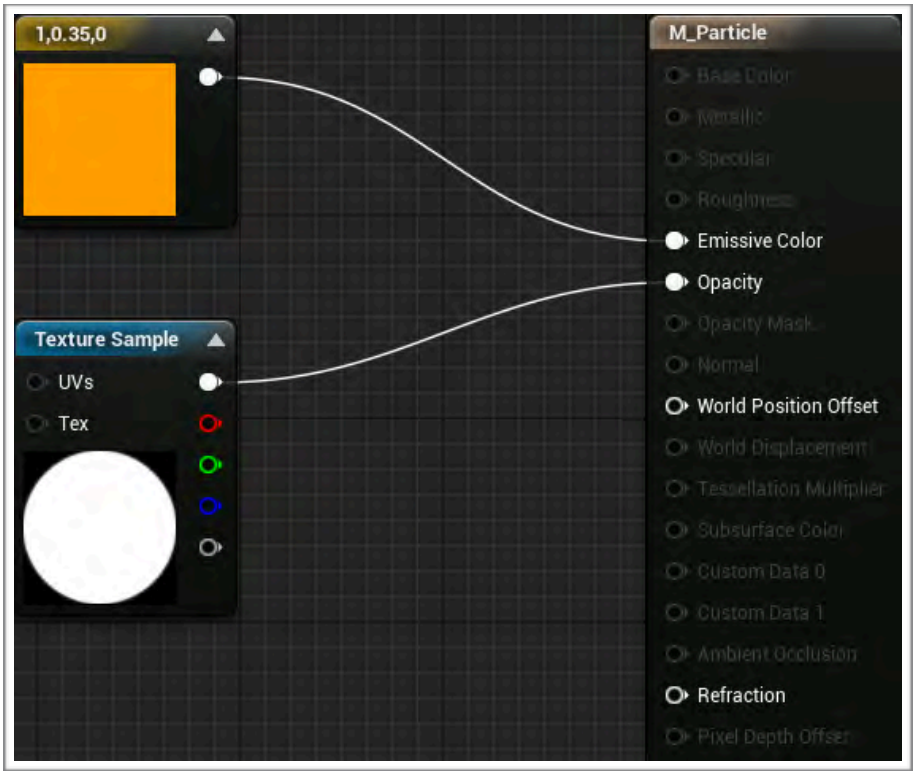
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

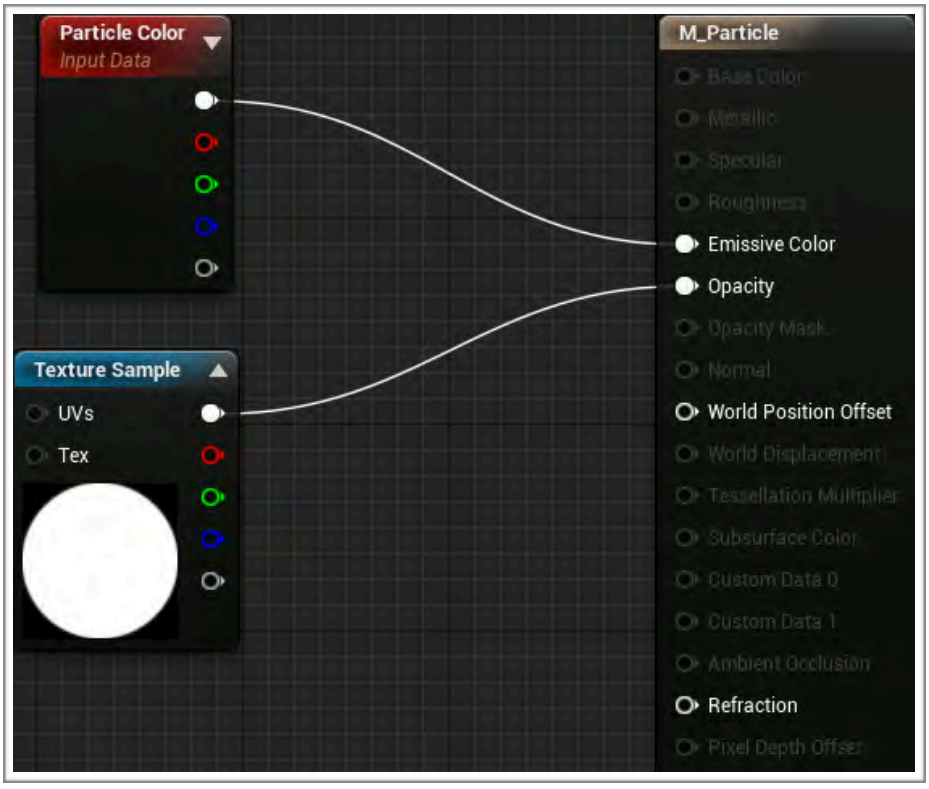
欢迎继续我们的学习。
在本课的内容中，我们将继续完善粒子特效。
首先我们将要给粒子系统添加一些色彩变化。

添加色彩变化

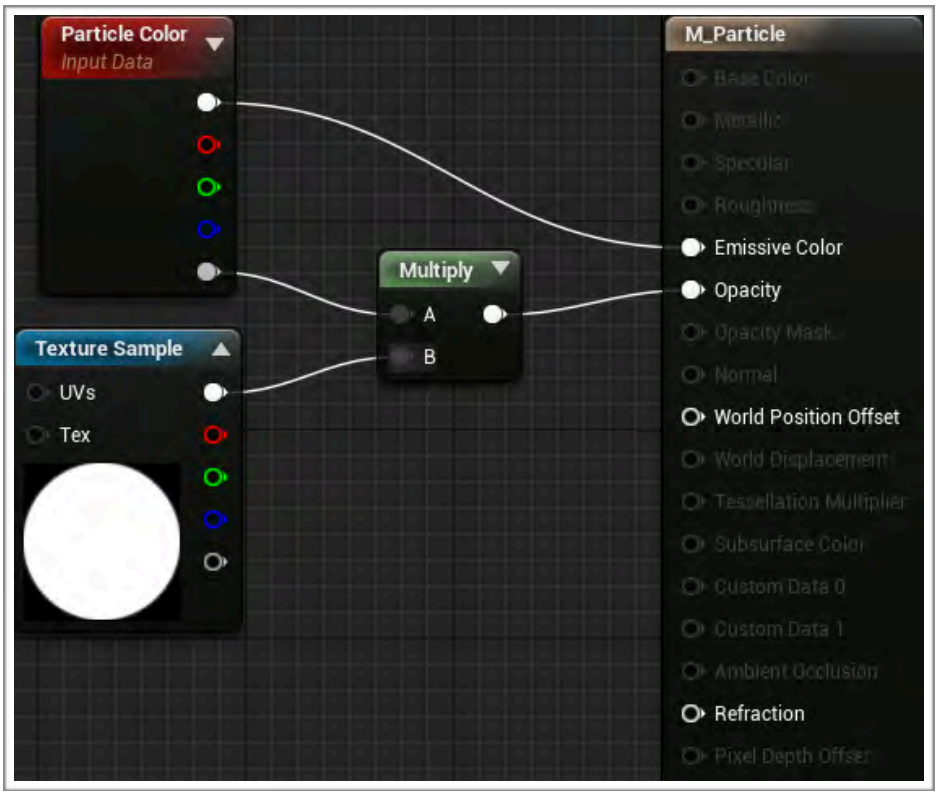
为使用Cascade设置粒子的色彩，我们需要正确设置粒子的材质。找到Materials文件夹，然后打开M_Particle。



当前的色彩是直接在材质中设置的。为使用粒子系统的色彩，我们需要用到ParticleColor节点。
首先删除连接到Emissive Color的节点，然后添加一个新的ParticleColor节点，并使用以下方式连线：



如果我们需要同时控制粒子的透明度，那么需要添加一个Multiply节点，并使用如下方式连线：



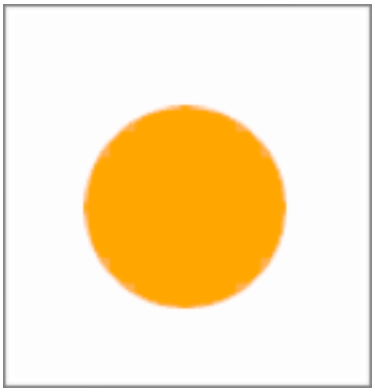
点击Apply按钮，然后关闭M_Particle。
为了设置粒子的色彩，我们还需要使用Initial Color模块。

Initial Color模块
打开PS_Thruster，然后添加一个Initial Color模块，该模块可以在Color分类中找到。

为了添加色彩变化，我们需要指定一个色彩变化范围。为此我们需要用到distribution。
选中Initial Color，然后在Details面板中展开Start Color部分，并将Distribution更改为Distribution Vector Uniform。这样我们就可以为每个色彩通道指定范围。

对本教程来说，色彩的变化范围应该是从橙色到红色。为此，将Max设置为（1.0,0.0,0.0），Min设置为（1.0,0.35,0.0）。

此时注意观察Viewport视口，可以看到色彩的变化很奇怪，有点像星际能量跃迁~



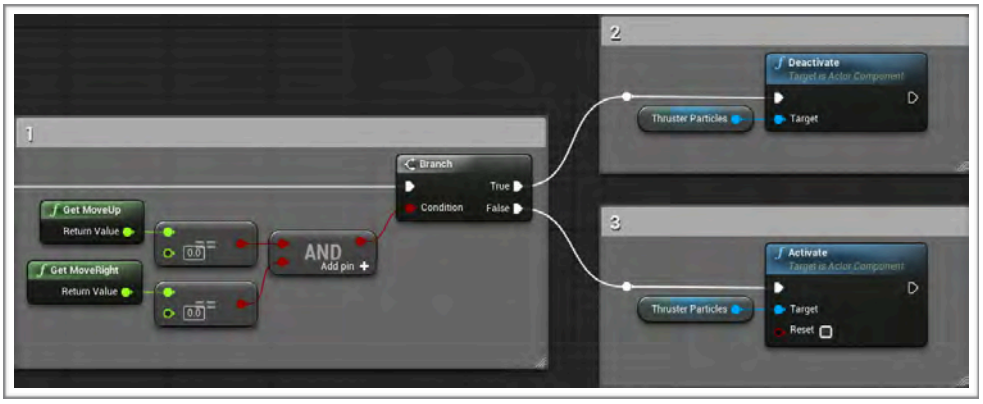
出现这种现象的原因是Color Over Life模块在持续将色彩更新到白色。为此，让我们选中Color Over Life，然后按下Delete。此时模块列表显示如下：

关闭PS_Thruster，然后点击主编辑器工具栏上的Play按钮，可以看到此时的推进器🔥视觉效果更酷了。

接下来我们需要根据飞船的运行状态来开启或关闭粒子特效。

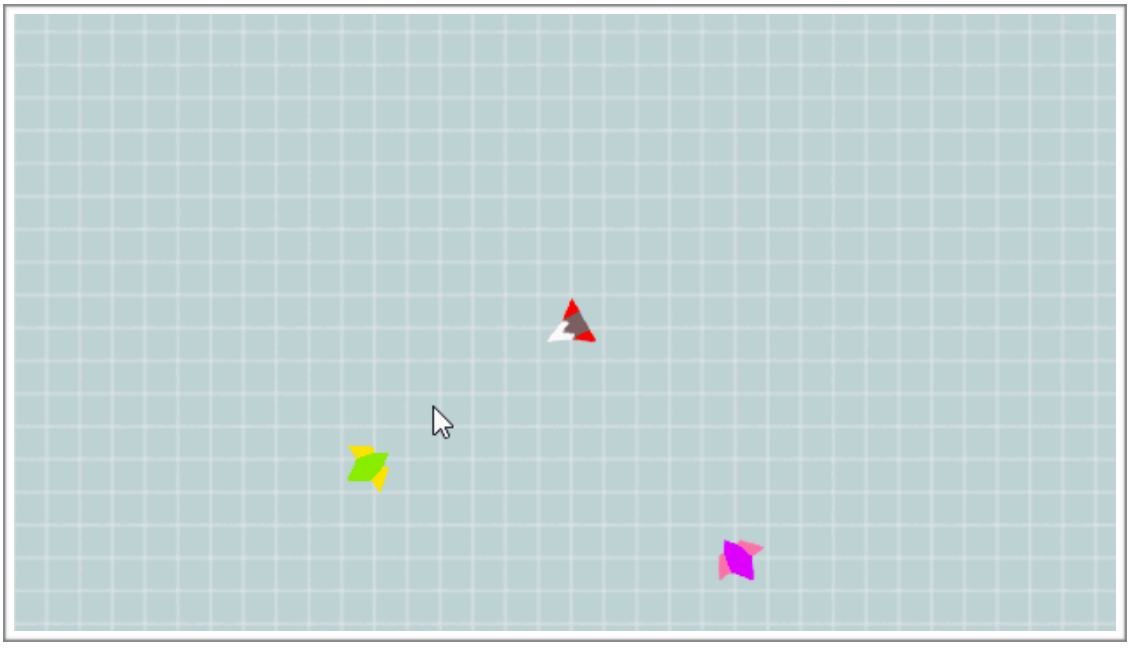
开启和关闭粒子特效

为了检查飞船是否在运行，我们需要检查玩家是否按下了任一运动控制按钮。
打开BP_Player，找到Event Tick节点。在节点链的最后添加以下设置。



让我们看看以上设置所完成的事情：
1.检查MoveUp和MoveRight的映射，如果返回值是0，就代表玩家没有按下任何运动按钮。
2.如果Branch分支节点返回true，代表玩家没有按下任何运动按钮，此时需要禁用ThrusterParticles
3.如果Branch分支节点返回false，代表玩家按下任何运动按钮，此时需要启用ThrusterParticles

点击Compile按钮，然后关闭BP_Player。点击工具栏上的Play按钮，然后切换运动状态，查看粒子效果的开启和关闭是否生效。



好了，关于飞船推进器粒子特效的创建就到这里了，我们下一课再见~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。

在之前的内容中，我们成功创建了飞船推进器的尾焰效果。接下来我们来一起学习创建爆炸💥的粒子特效。

创建爆炸特效

为了方便起见，我们无需创建一个新的粒子特效，只需要复制出飞船推进器的尾焰效果，然后在它的基础上调整就好了。

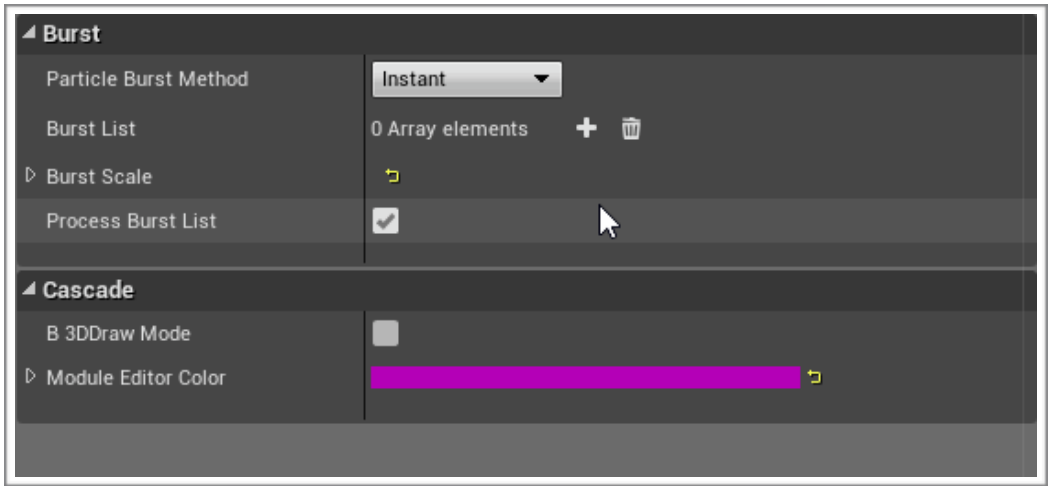
找到ParticleSystem文件夹，右键单击PS_Thruster，选择Duplicate。将其重命名为PS_Explosion，然后打开。

对爆炸效果来说，所有的粒子都应该在同一时间出现，而不是一个接一个的出现，这就是所谓的burst-emitting。

创建burst

首先，我们需要将粒子的生成速率spawn rate设置为0，因为无需使用默认的生成行为。选中Spawn模块，然后将Spawn\Rate\Distribution\Constant设置为0。

接下来我们需要通知emitter，接下来希望创建一个burst。找到Burst部分，然后在Burst List中添加一个新的记录。直接点击+小图标就可以完成。



每条记录都包含了三个字段：

1.Count

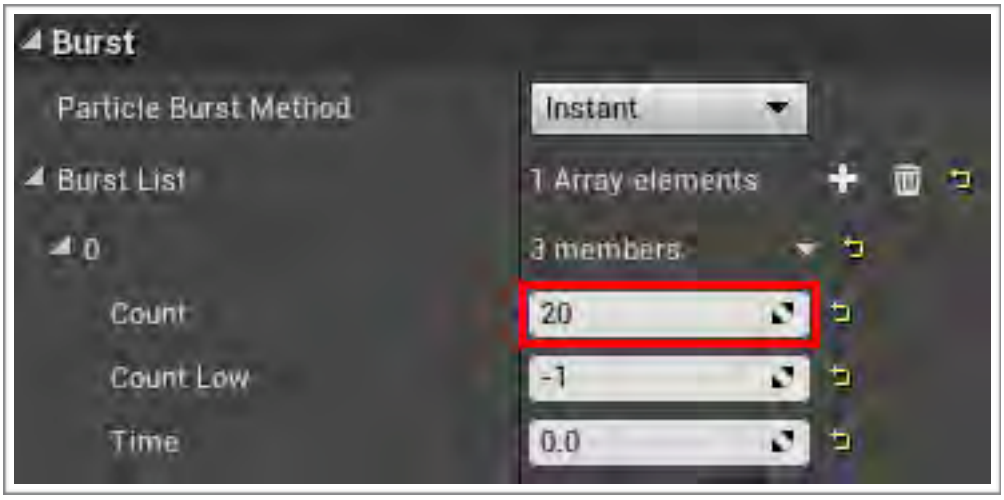
代表要生成的粒子数量，这里将其设置为20。

2.Count Low

如果大于或等于0，那么生成的粒子数量范围将在Count Low到Count之间。这里保留-1的默认值。

3.Time

代表何时生成粒子。0表示emitter生命周期的开始，1表示emitter生命周期的结束。这里保留默认的0.0

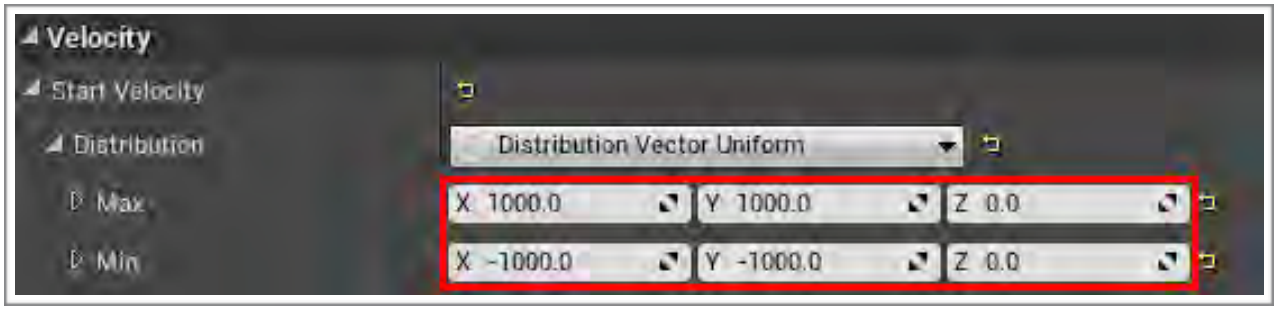


这就意味着emitter粒子发射器将在生命周期的开始生成20个粒子。

为了让粒子特效看起来像爆炸的效果，我们还需要设置粒子向外运动的速率。

让粒子向外运动

因为这是一个简单的2D游戏，所以我们只需要设置X和Y轴的速度即可。
选择Initial Velocity模块，展开Start Velocity\Distribution。将Max设置为（0，1000，0），将Min设置为（-1000，1000，0）。



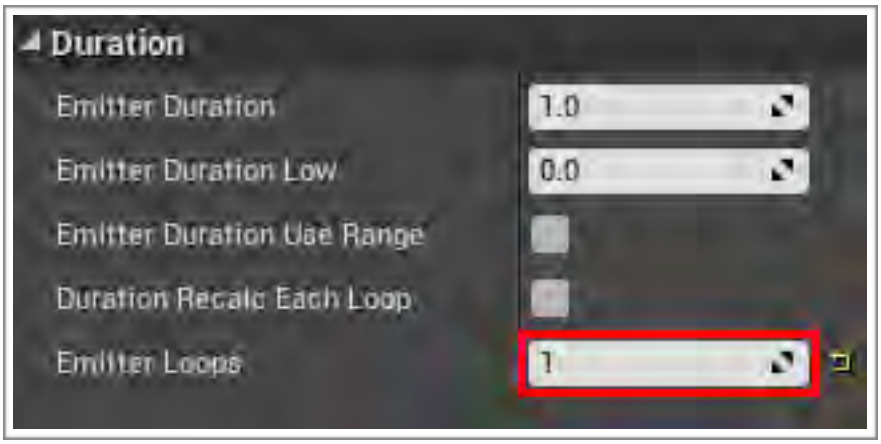
通过指定以上的范围，可以让粒子向发射器的外部运动。

接下来我们需要设置粒子发射器的循环时间

设置emitter loop

默认情况下，粒子发射器会一直循环下去。对于🔥或是烟雾之类的效果，这种设置是合理的。但是💥效果只需要播放一次。因此，我们需要让发射器只循环播放一次。

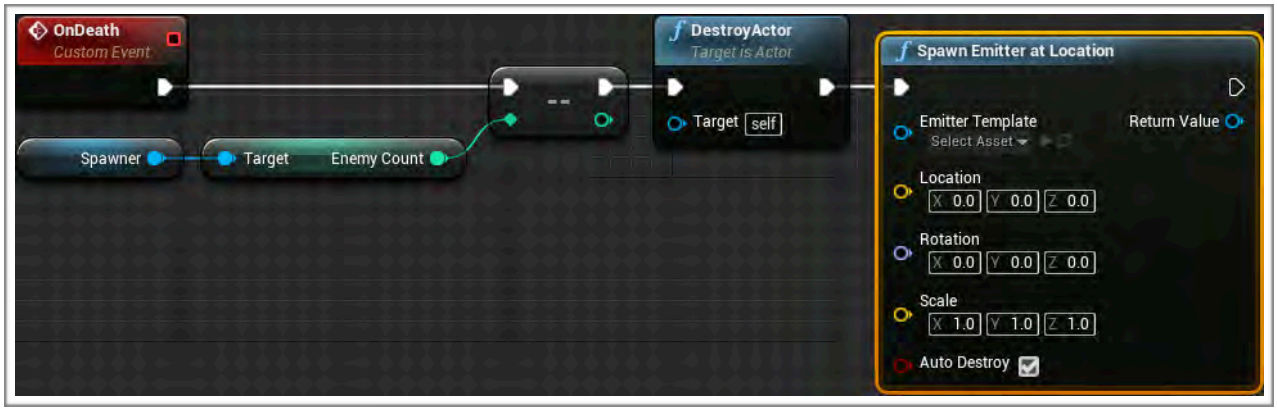
选中Required模块，然后找到Duration部分。将Emitter Loops设置为1。



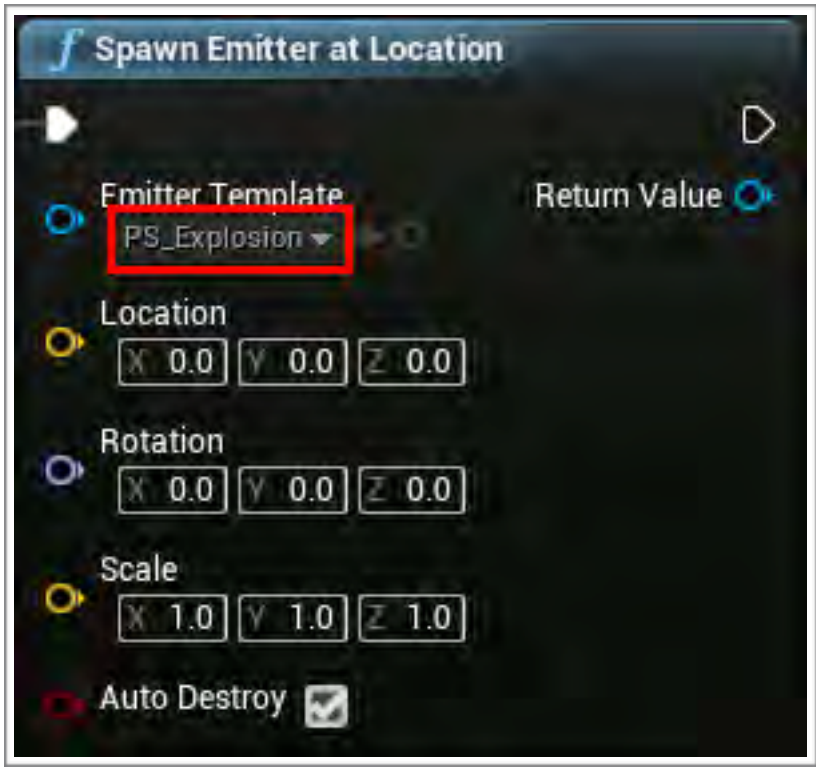
接下来，我们需要在敌人飞船受到攻击的时候播放💥效果。

在敌军飞船受攻击时生成粒子特效

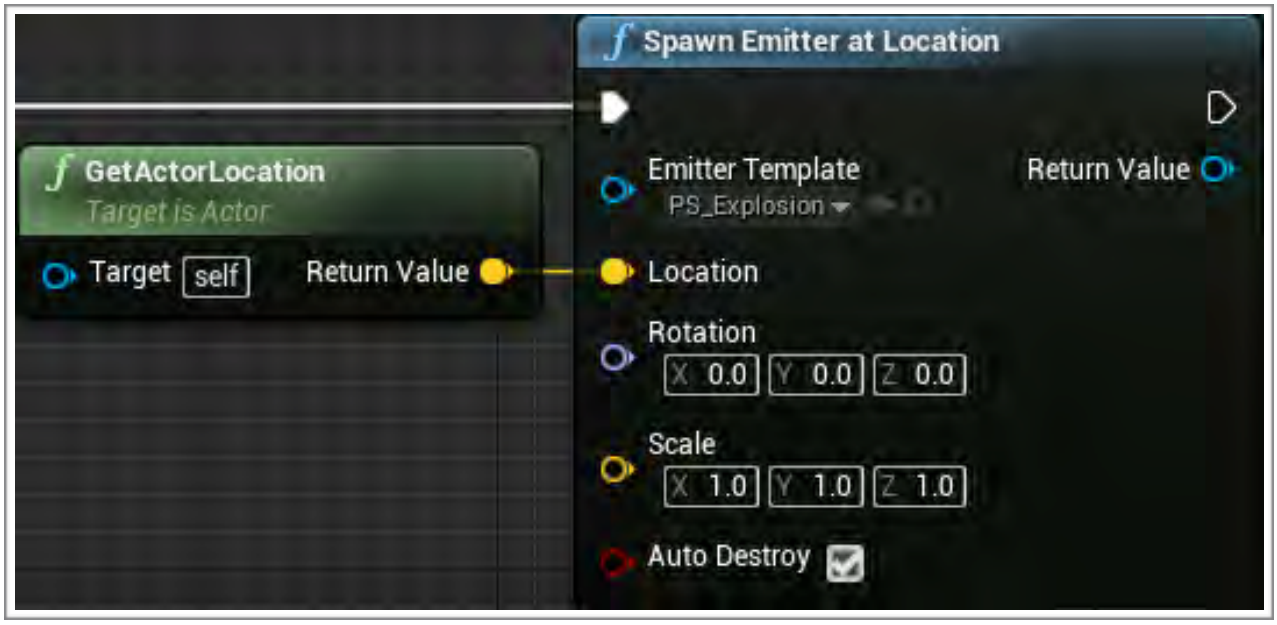
返回主编辑器，打开Blueprints文件夹。打开BP_Enemy，然后找到OnDeath事件。
为了生成粒子特效，我们可以使用Spawn Emitter at Location节点。创建该节点，并将其连接到Destroy Actor节点上。



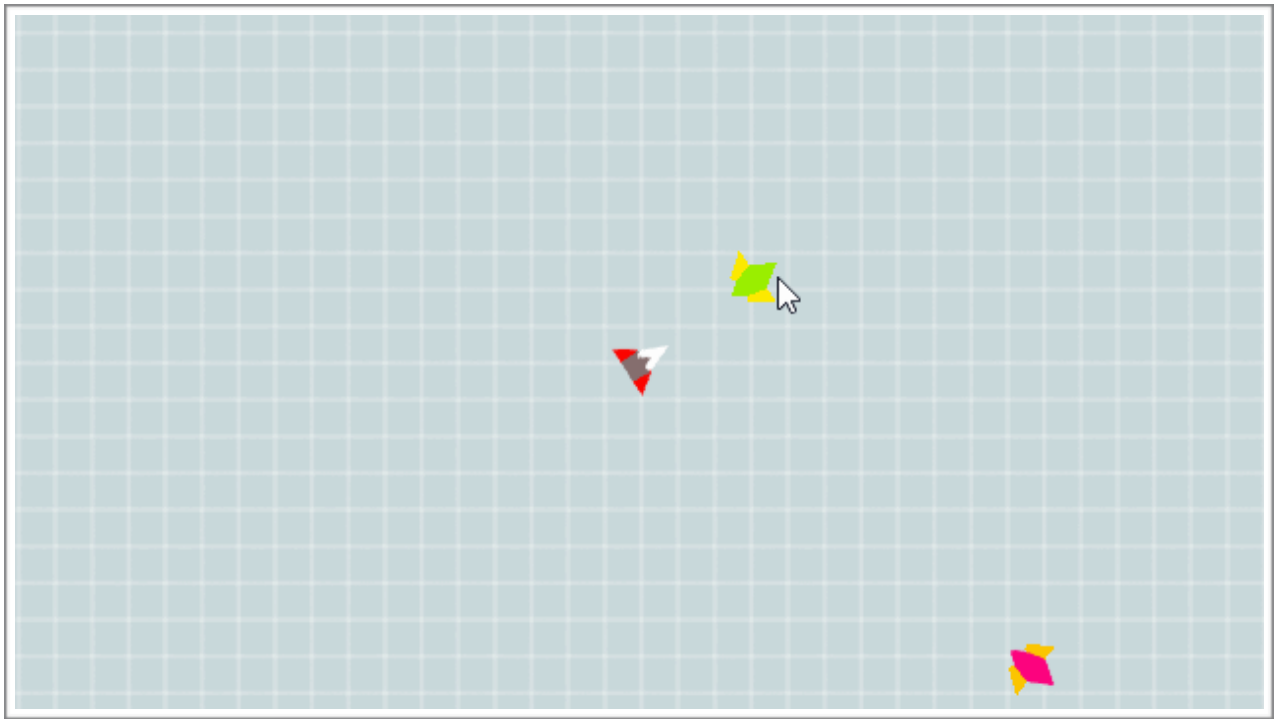
接下来设置Emitter Template为PS_Explosion。



最后，创建一个GetActorLocation节点，并将其连接到Location端口上。



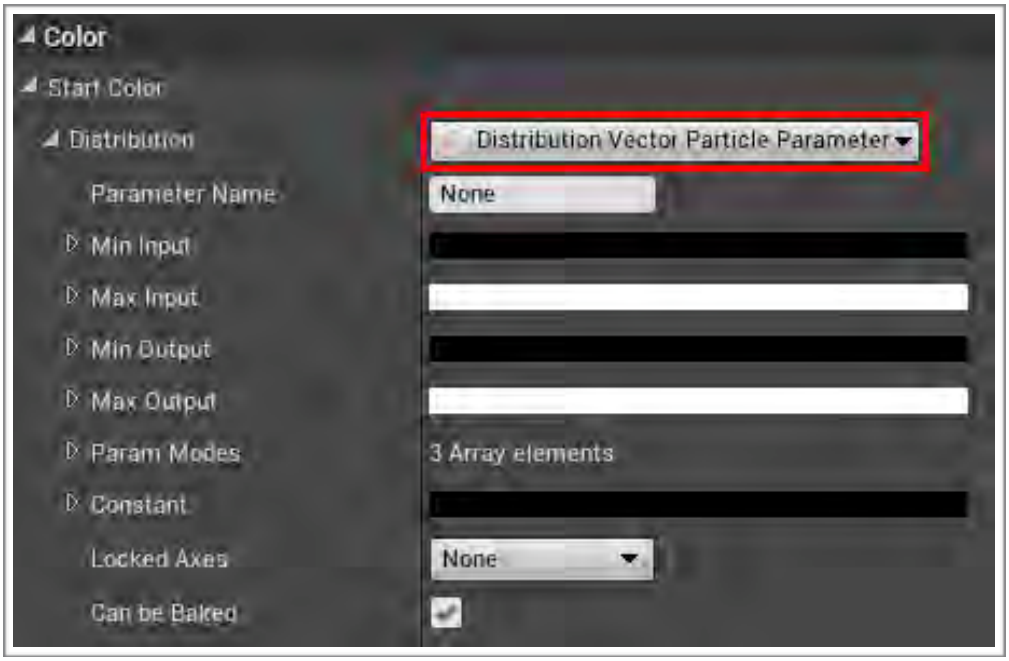
现在，当敌军受到攻击时，就会在对应的位置生成PS_Explosion的实例。
点击Compile按钮保存，然后返回主编辑器。
点击工具栏上的Play按钮预览游戏效果。



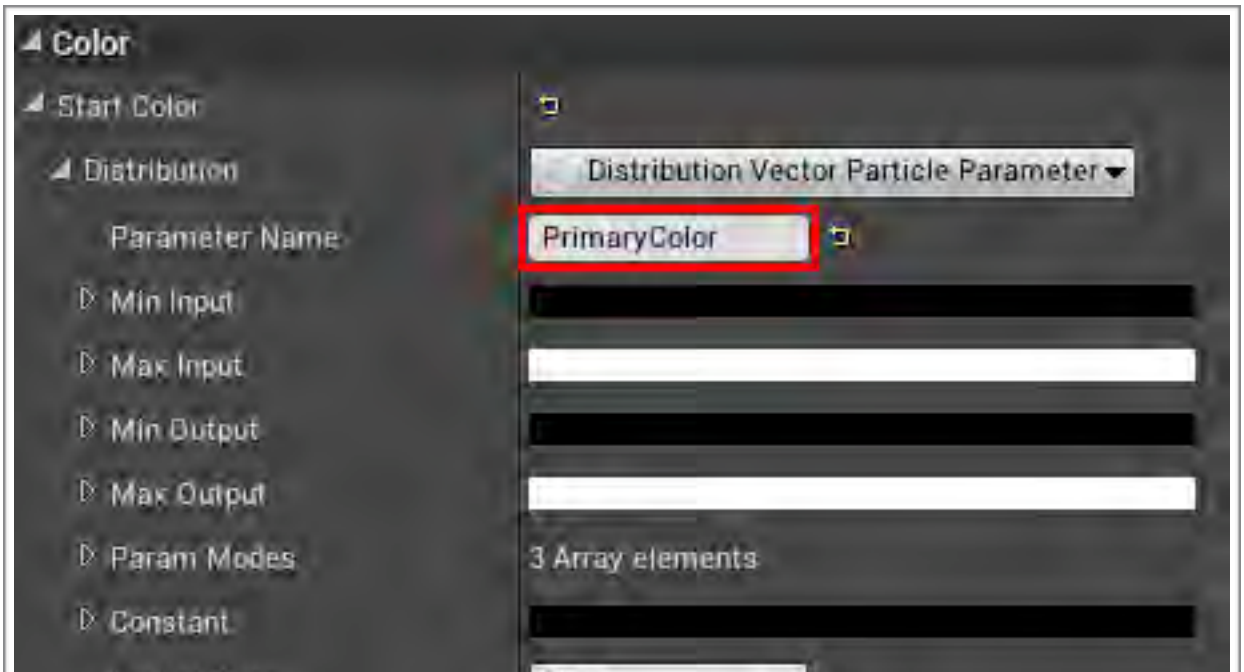
接下来，让我们将粒子的颜色设置为跟敌军相同。

将爆炸粒子的色彩设置为敌军飞船的色彩。

为了使用敌军的色彩，我们需要想办法从蓝图中接收信息。幸运的是Cascade种提供了distribution，可以完成这种操作。
打开PS_Explosion，选择Initial Color模块。将Start Color\Distribution设置为Distribution Vector Particle Parameter。



这样我们就获得了一个参数，可以使用蓝图进行设置。
将Parameter Name设置为PrimaryColor。



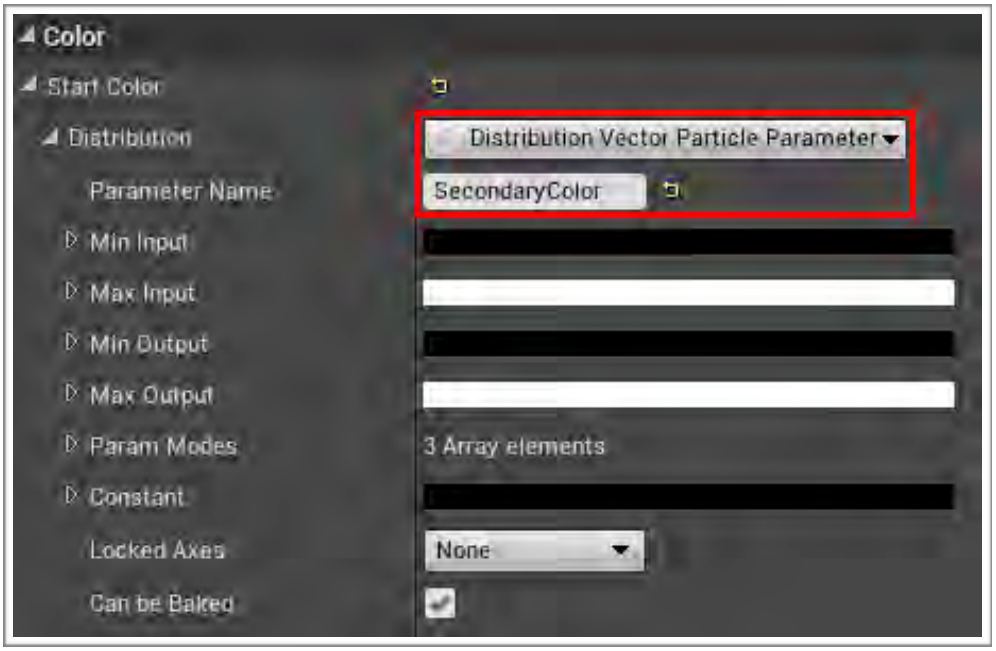
对爆炸效果来说，我们可以使用两种敌军的颜色。为了使用第二种颜色，我们需要另外一个emitter。右键单击emitter的空白区域，选择Emitter\Duplicate and Share Emitter。这样就可以复制出一个emitter。

可以看到，每个模块上都有了一个+标记。通过使用Duplicate and Share Emitter而非Duplicate，我们就可以将模块关联在一起，而不是简单的复制。接下来我们对任一模块的修改都会反应在另外一个emitter的同一模块上。如果我们希望同时更改所有emitter的相同属性，这样做就很方便了。

这里我们只需要更改Initial Color模块。不过正如刚才所说的，当我们作出任何更改时，两个emitter都会发生同样的变化。对当前的游戏来说，我们不需要模块关联在一起，因为它们需要不同的参数名。为此，我们需要删除复制出来的Initial Color模块，并创建一个新的模块。

注意：到目前为止，虚幻4引擎还没有提供内置的方法来取消对模块的关联。

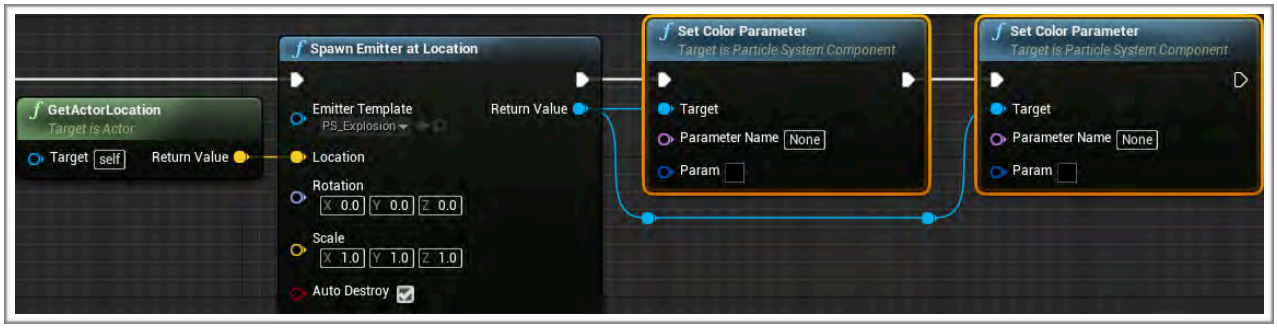
选中新的Initial Color，将Start Color\Distribution 设置为Distribution Vector Particle Parameter。
接下来，将Parameter Name设置为SecondaryColor。



到这里，整个粒子系统已经完成了。关闭PS_Explosion。
接下来，我们需要使用蓝图设置相关的参数。

使用蓝图设置粒子参数

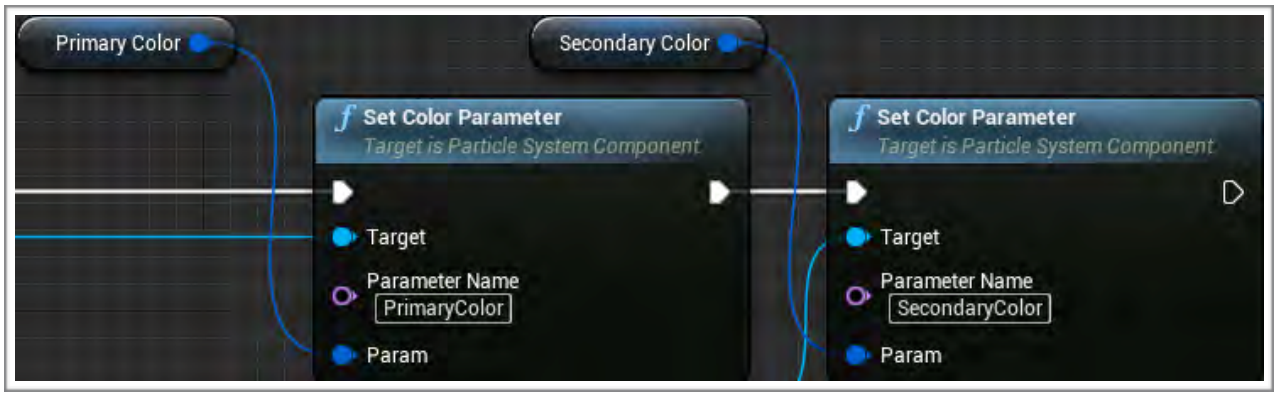
打开BP_Enemy，并在Spawn Emitter at Location节点之后添加下图中所高亮的节点：



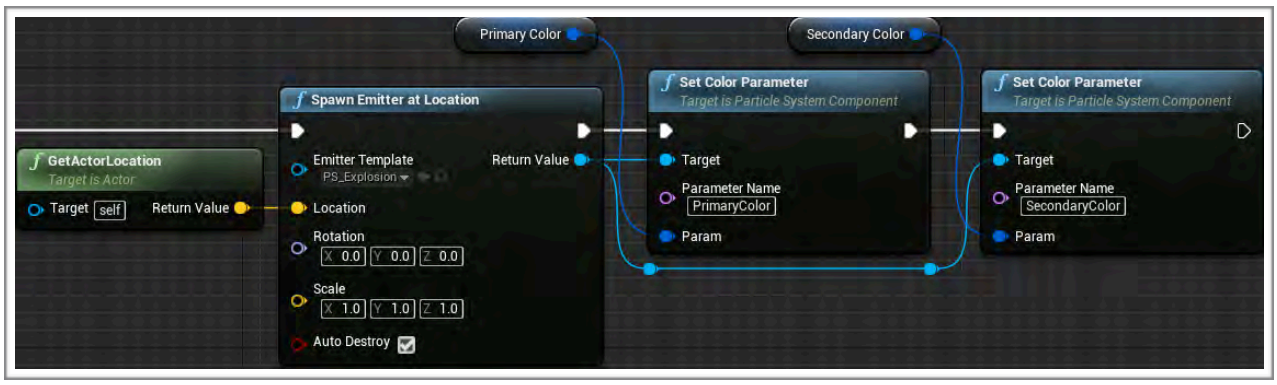
这样我们就可以在PS_Explosion中设置两个参数了。

现在我们需要设置正确的参数名。将第一个Set Color Parameter节点中的Parameter Name设置为PrimaryColor。将第二个节点中的Parameter Name设置为SecondaryColor。

最后我们还需要提供颜色。为了简化起见，颜色已经存储在变量PrimaryColor和SecondaryColor中。使用以下方式将变量和对应的节点关联在一起。



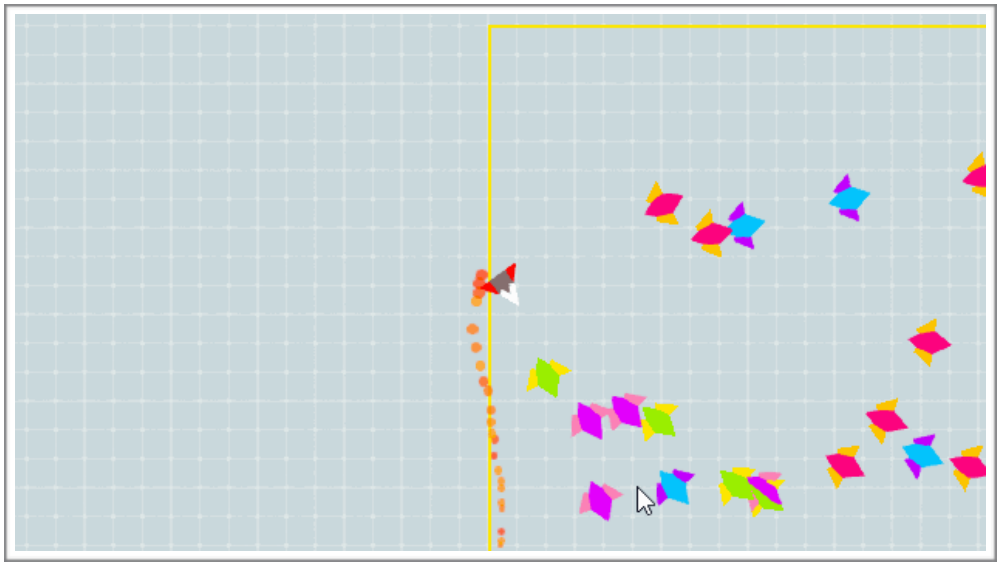
最后的节点连线图如下图所示：



让我们来看看事件的执行顺序：

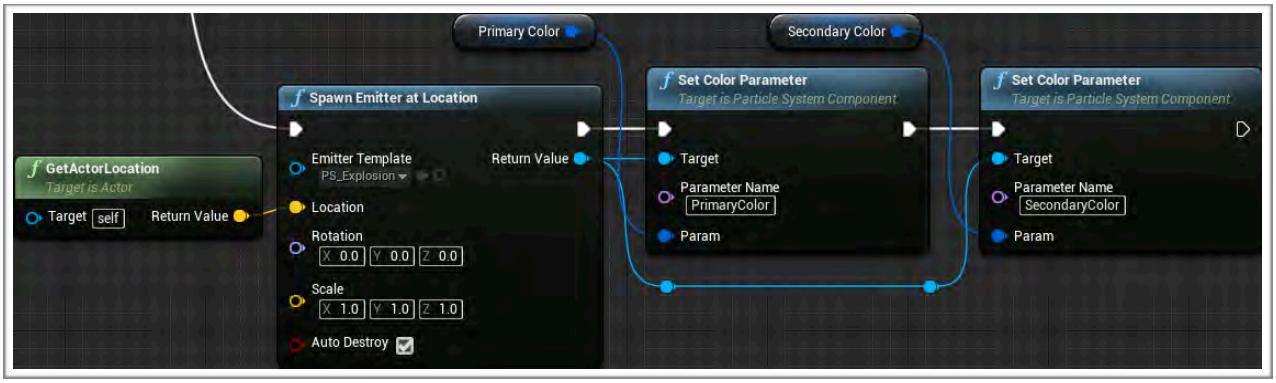
1.当敌军飞船受到攻击时，将在对应的位置生成一个PS_Explosion的实例。
2.设置PS_Explosion的PrimaryColor参数
3.设置PS_Explosion的SecondaryColor参数

点击Compile按钮，然后关闭BP_Enemy。点击Play按钮，享受消灭敌军飞船的爽快粒子效果吧~



小练习：
添加一个新的粒子特效，当玩家飞船受到攻击的时候播放。

练习答案：
1.打开BP_Player，找到OnDeath事件
2.向Sequence节点的Then 1端口添加Spawn Emitter at Lcoation节点。将Emitter Template设置为PS_Explosion。
3.创建一个GetActorLocation节点，将其连接到Spawn Emitter at Location节点的Location端口
4.创建一个Set Color Parameter节点，并将其关联到Spawn Emitter at Location节点。将Parameter Name设置为PrimaryColor，并将PrimaryColor变量关联到Param。
5.创建另外一个Set Color Parameter节点，并将其关联到第一个Set Color Parameter节点上。将Parameter Name设置为SecondaryColor，并将SecondaryColor变量关联到Param。



好了，关于创建游戏中的粒子特效部分内容就到此结束了~
完整的项目请在这里下载：
链接:https://pan.baidu.com/s/1J5lDLz_QKobnV6XKVv_4AA 密码:7geq

从下一课开始，我们将一起来学习虚幻4中的人工智能~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

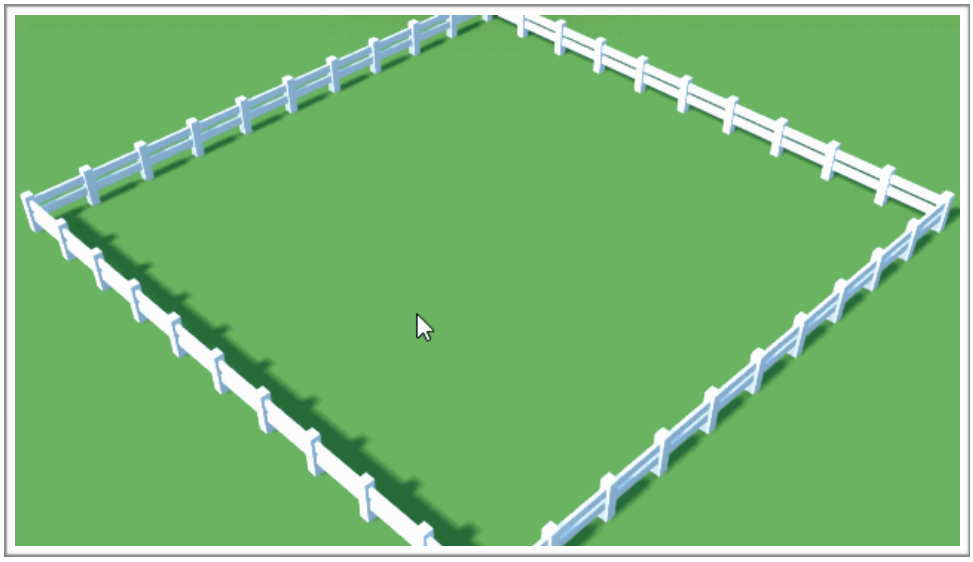
个人网站：
http://icode.ai/

欢迎继续我们的学习。
从这一课的内容开始，我们将一起了解游戏中的人工智能。
大家都知道，人工智能的概念这两年超级火爆。遗憾的是，目前游戏中所使用的人工智能技术并没有达到足够的高度。不过从去年开始，暴雪和Google DeepMind开始合作在《星际争霸》等游戏中使用最新的深度学习技术，以提升机器的对战能力。
在游戏中，AI指的是非玩家角色如何做出决策。简单的决策包括发现玩家并开始攻击，而复杂的决策可能是在RTS实时战略游戏中如何让AI控制的角色跟人类对战。
在虚幻4引擎中，我们通过behavior tree(决策树)来创建AI。顾名思义，决策树用于判断AI应该执行何种决策。例如，在某个游戏中我们设定了游戏角色可以有战斗和逃跑的行为。我们可以创建一个决策树，当AI在自己的生命值在50%以上时持续战斗，而当生命值低于50%时则选择逃跑。
在这部分的内容中，我们将学习以下内容：
1.创建一个AI实体，从而可以控制一个Pawn角色
2.创建并使用决策树和黑板
3.创建AI Perception，从而让Pawn具备视力
4.创建行为，从而让Pawn角色可以漫游并攻击敌人

好了，废话不多说，让我们进入正题。

开始前的准备

在开始之前，首先下载起始项目(链接:https://pan.baidu.com/s/1x_dgtd_qnwL30CREaRWb7g 密码:n7vn)
打开项目文件夹，并打开MuffinWar.uproject。
点击Play开始体验游戏。左键点击在篱笆区域内生成一个松饼。



在这部分的课程中，我们将创建一个可以自由行走的AI。当作为敌军角色的松饼进入AI角色的视野范围内时，AI角色就会向松饼靠近，并开始攻击。
为了创建AI角色，我们需要三个东西：
1.身体：
这是角色的物理形态，在这款游戏中松饼就是身体。
2.灵魂：
灵魂就是控制角色的实体，在游戏中通常代表玩家或AI。
3.大脑：
大脑是指AI任何作出决策。我们可以使用不同的方式来创建，比如C++代码，蓝图或是行为树。

现在我们已经有了AI角色的身体，接下来只需要创建灵魂和大脑。

我们这一课的内容就到这里了，下一课再见~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

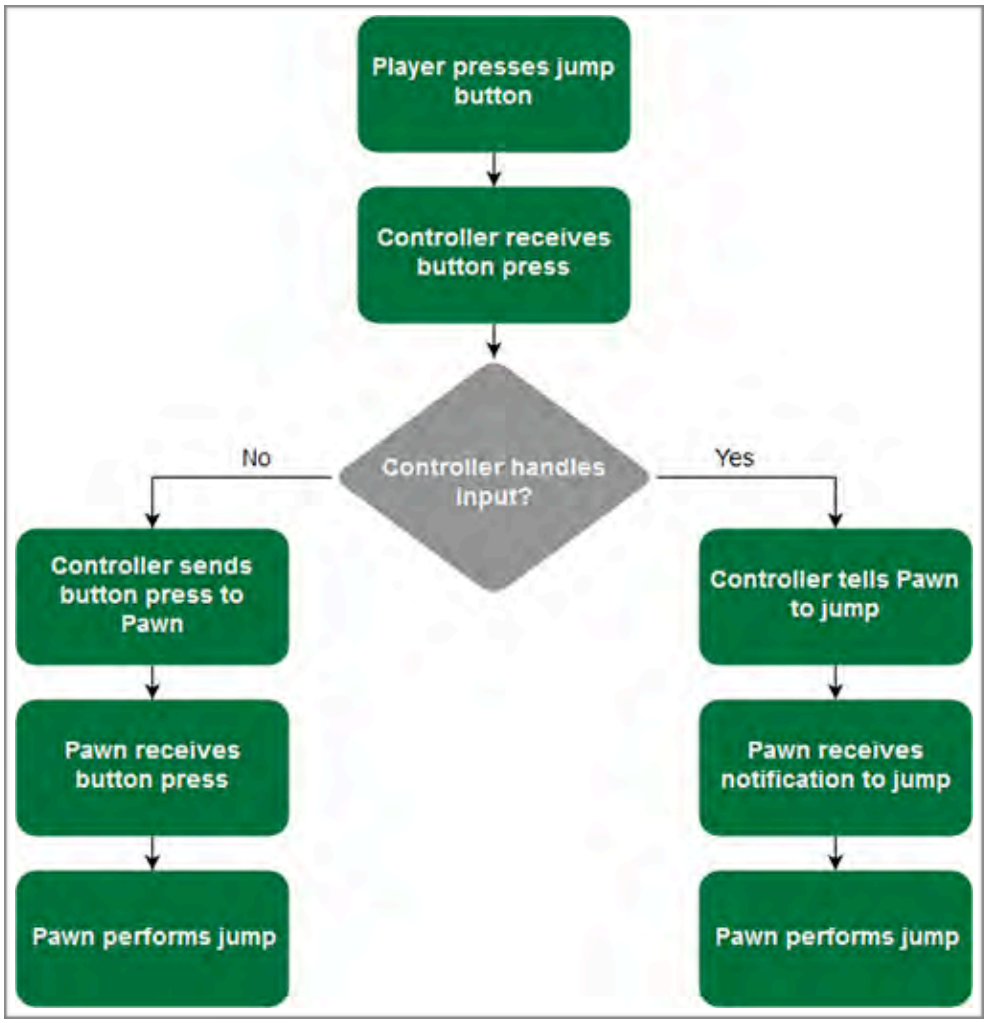
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。
在这一课的内容中，我们将创建AI的灵活，也就是controller（控制器）。

什么是控制器?

在虚幻4中，控制器属于非物理的角色，可以possess一个Pawn，进而control(控制）一个Pawn。
那么这里的”control”控制指的是什么呢?
对玩家来说，意味着当按下某个按键时，Pawn就会去做某些指定的事情。控制器将从玩家那儿接收输入，然后发送到Pawn。此外，控制器还可以处理输入，然后通知Pawn执行某个动作。

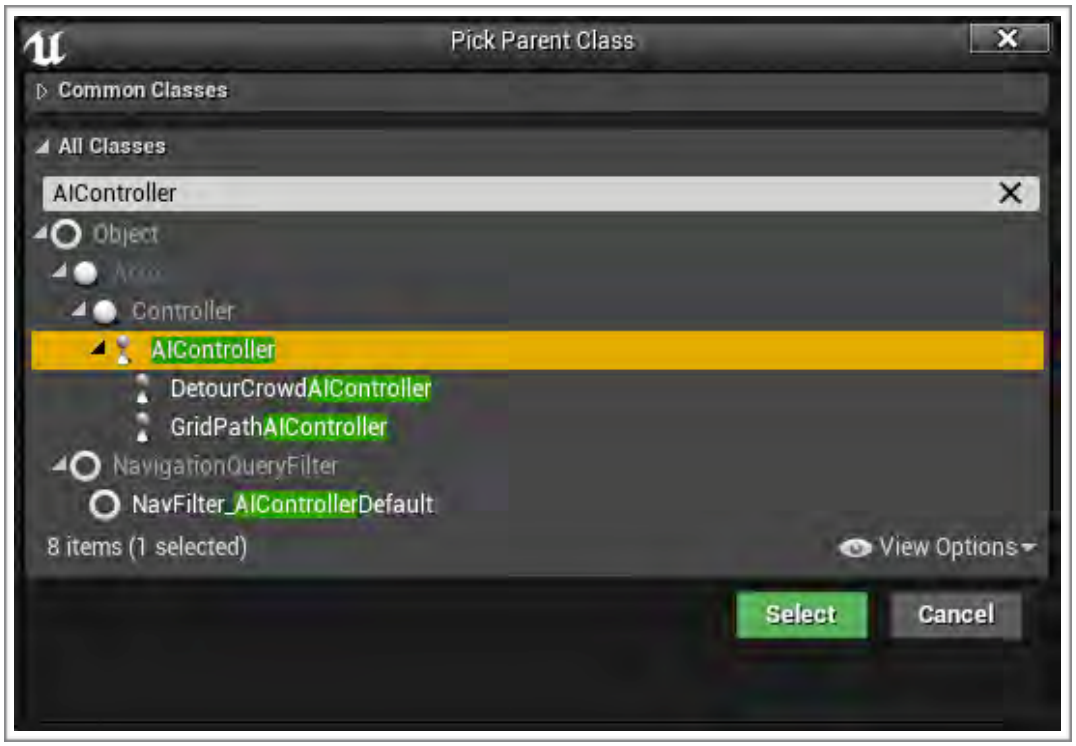


对AI来说，Pawn可以从控制器或“大脑”中接收信息（取决于我们如何编码~）。

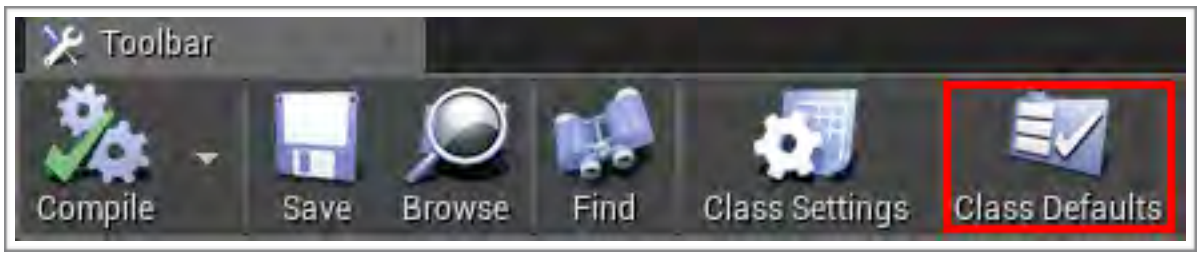
为了使用AI来控制松饼，我们需要创建一种特殊的控制器，也就是AI Controller。

创建一个AI Controller

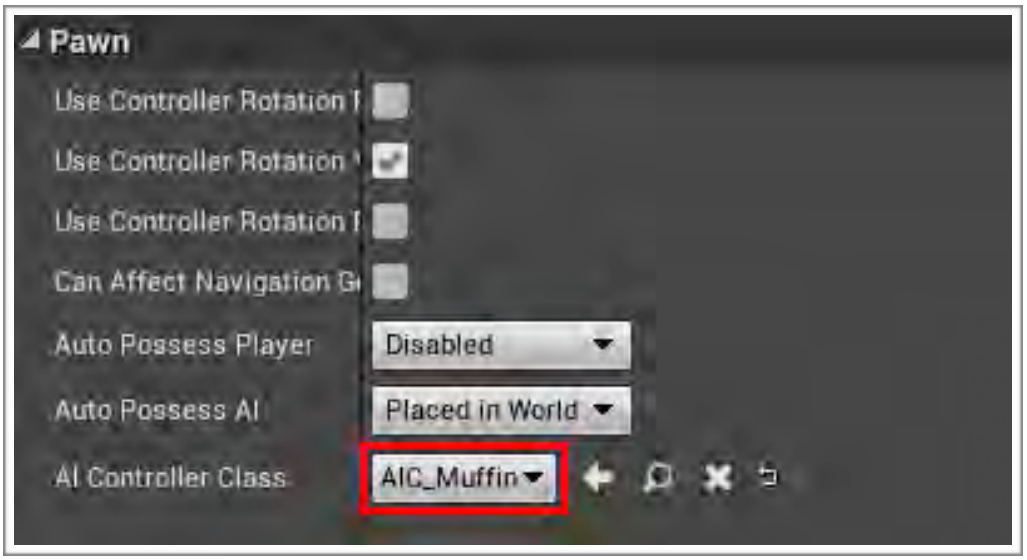
在虚幻4中打开Characters\Muffin\AI，然后创建一个新的Blueprint Class。选择AIController作为parent class，并将其命名为AIC_Muffin。



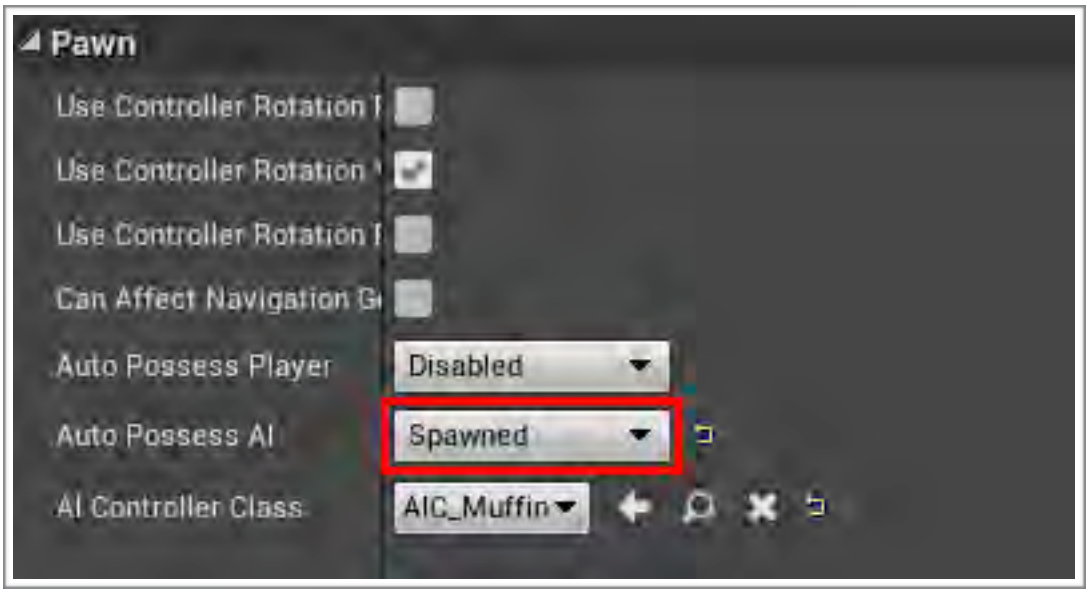
接下来，我们需要通知松饼使用新的AI Controller。打开Characters\Muffin\Blueprints，然后打开BP_Muffin。
默认情况下，Details面板中应该显示了蓝图的默认设置。如果没有，可以手动点击工具栏上的Class Defaults。



在Details面板中找到Pawn部分。将AI Controller Class设置为AIC_Muffin。这样，当生成松饼的时候就会生成一个控制器的实例。



因为我们要生成松饼，所以还需要将Auto Possess AI设置为Spawned。这样就可以确保生成松饼时让AIC_Muffin自动拥有BP_Muffin。



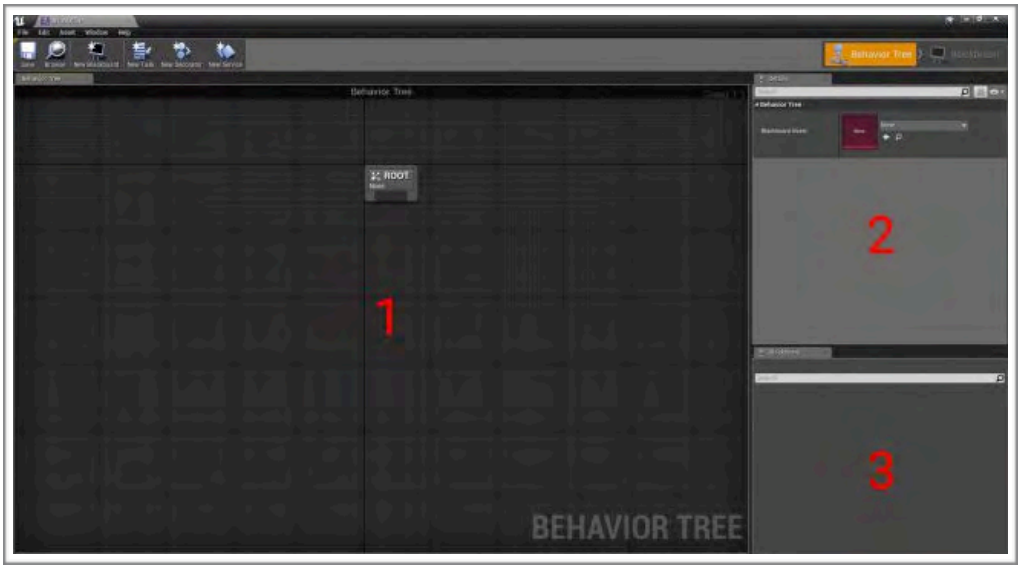
点击工具栏上的compile按钮，然后关闭BP_Muffin。
接下来，我们可以创建用于驱动松饼行为的逻辑。为此，我们需要用到行为树（behavior trees)>

创建一个行为树

找到Characters\Muffin\AI，选中Add New\Artificial Intelligence\Behavior Tree。将其命名为BT_Mufin，然后将其打开。

行为树编辑器（Behavior Tree Editor)

行为树编辑器中包含了两个新的面板：



1.Behavior Tree:

该视图中用来创建和行为树相关的节点

2.Details:

所选中的节点将在这里显示出具体的属性

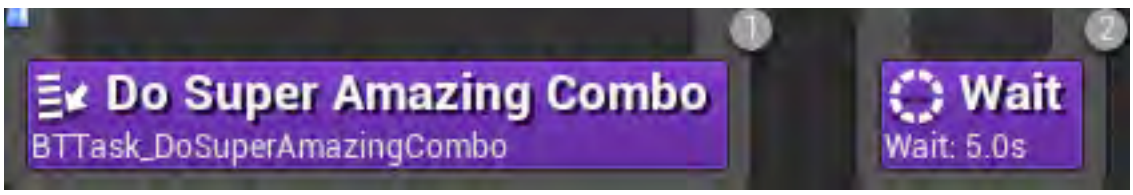
3.Blackboard:

该面板将显示Blackboard key及其对应的值，不过只有在游戏运行时才会显示。

和蓝图类似，行为树中也包含了各种节点。行为树中有四种不同类型的节点，头两个是tasks和composites。

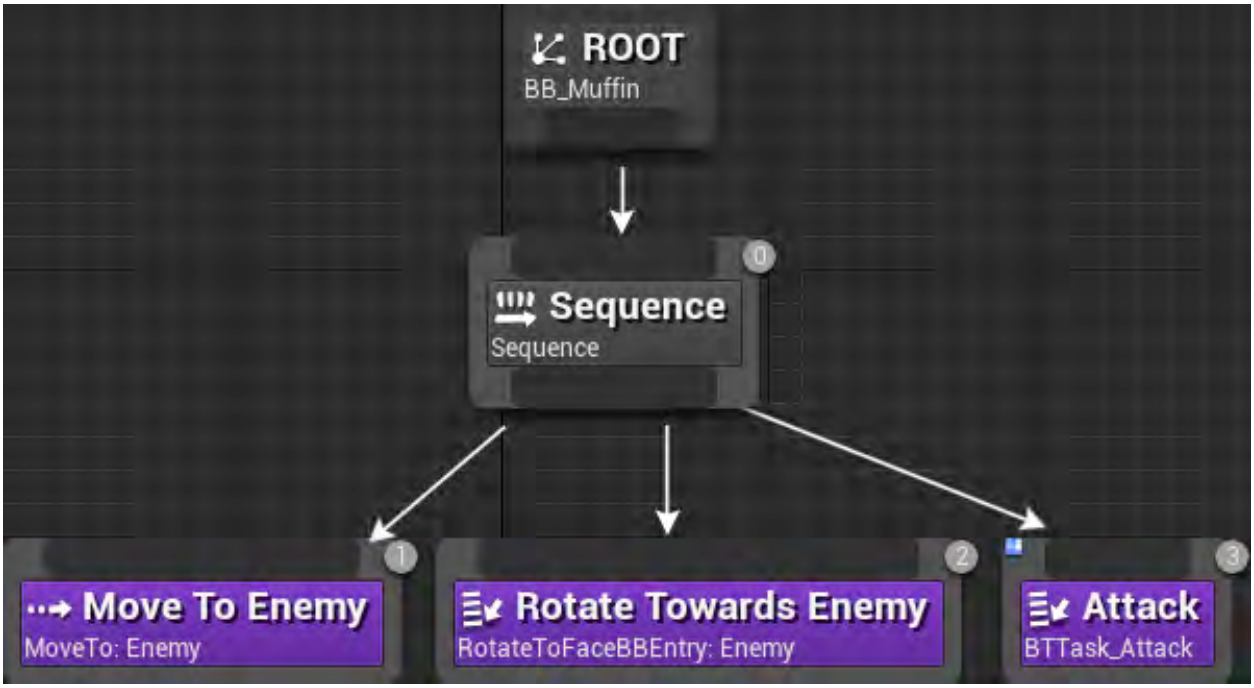
什么是Tasks和Composites

顾名思义，task节点通常需要完成某项“任务”。这种任务可能是复杂的复合操作，也可能是异常简单的等待。



为了执行任务，我们需要使用composites。行为树中包含了多个分支（behaviors）。每个分支的根部都是一个composite。不同类型的composites可以用不同的方式来执行子节点。
例如，我们可以使用类似下面的行为序列：

为了执行序列中的动作，我们需要用到Sequence composite。这是因为Sequence节点将从左向右执行其子节点，如下图所示：



注意：
从composite开始的都可以被称作一个subtree（字树）。通常来说也是行为。在我们这个例子中，Sequence,Move To Enemy，Rotate Towards Enemy和Attack可以被认为是攻击敌人的行为。

如果Sequence中的任何一个子节点执行失败，整个Sequence序列都会停止执行。
例如，Pawn角色无法移动到敌人附近，那么Move To Enemy就会执行失败。这就意味着序列中的Rotate Towards Enemy和Attack也无法顺利执行。不过如果Pawn角色可以成功的移动到敌人附近，那么其它操作也将顺利执行。

在随后的内容中，我们还将了解Selector composite。
目前来说，我们将使用Sequence让Pawn角色移动到某个随机位置，然后等待。
好了，本课的内容就到这里了。
我们下一课继续~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

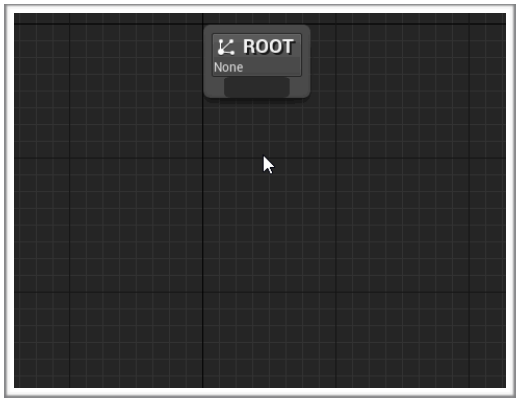
个人网站：
http://icode.ai/

欢迎继续我们的学习。

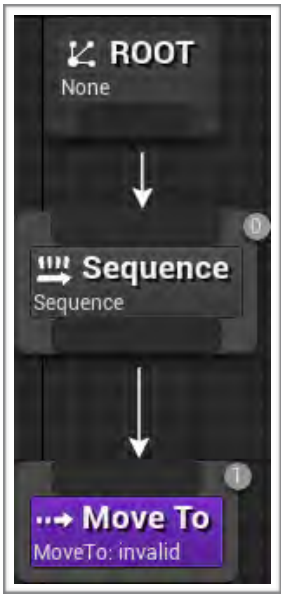
在这一课的内容中，我们首先需要让Pawn角色移动到某个随机的位置。

移动到某个随机位置

创建一个Sequence节点，然后将其连接到Root节点上。



接下来我们需要移动Pawn。创建一个MoveTo节点，然后将其连接到Sequence节点。该节点的作用是将Pawn移动到某个特定的位置或角色。



接下来创建一个Wait节点，然后将其连接到Sequence节点。注意这里要把Wait节点放在MoveTo节点的右边。在行为树中节点的顺序很重要，因为子节点将从向右执行。

注意：有一种简单的方法来确认节点的执行顺序，那就是查看每个节点右上角的数字编号。数字越小的节点越早被执行。

恭喜你，现在我们已经创建了自己的第一个行为树！它的作用是让Pawn角色移动到某个特定的位置，然后等待5秒。

为了移动Pawn角色，我们需要指定一个特定的位置。而MoveTo节点只支持从blackboards中接收数值。因此，我们需要创建一个blackboard。

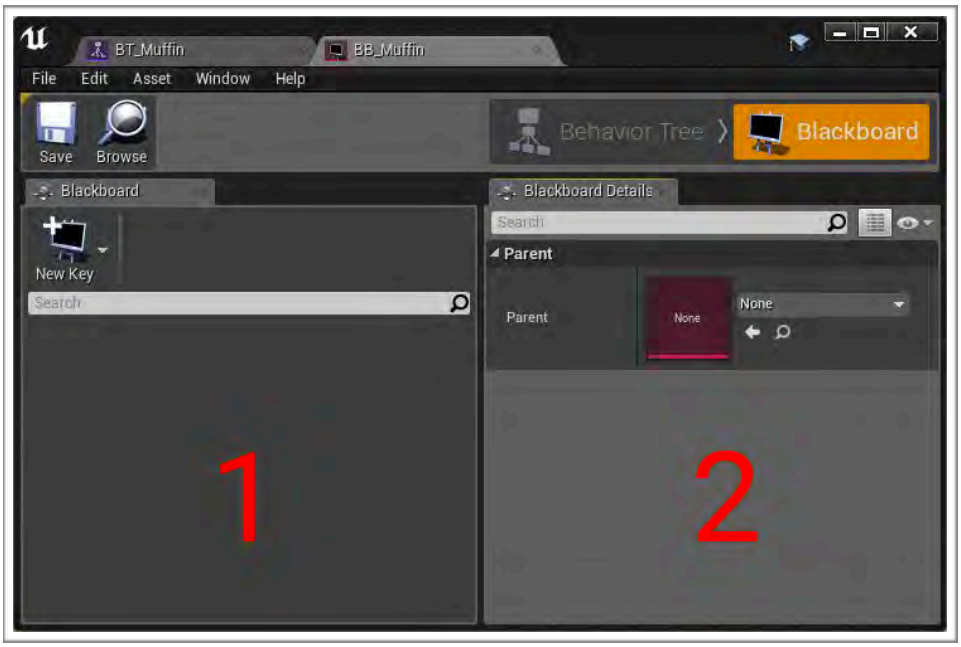
创建blackboard

blackboard是一种游戏资源，它的唯一作用就是保存变量（在虚幻4中用keys表示）。我们可以把blackboard看做AI的存储记忆。
尽管blackboard不是必需的，不过它提供了一种很便捷的保存和读取数据的方式。之所以说blackboard很方便，是因为行为树中的很多节点只接受blackboard中的key。

为了创建blackboard，我们需要返回Content Browser,然后选择Add New\Artificial Intelligence\Blackboard。将其命名为BB_Muffin，然后将其打开。

Blackboard编辑器

Blackboard编辑器包含了两个面板：

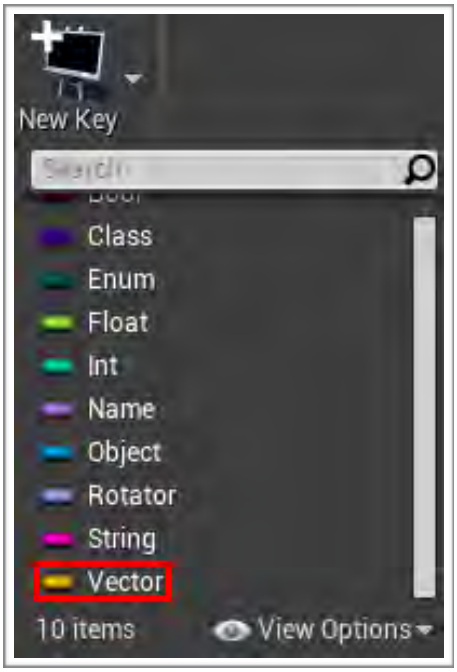


1.Blackboard:该面板将显示一系列的key
2.Blackboard Details:该面板将显示所选中的key的属性

接下来我们需要创建一个key，以保存目标位置。

创建目标位置的key

考虑到我们存储的位置信息处在3D空间，我们需要将其保存为一个vector变量。点击New Key，然后选择Vector，将其命名为TargetLocation。

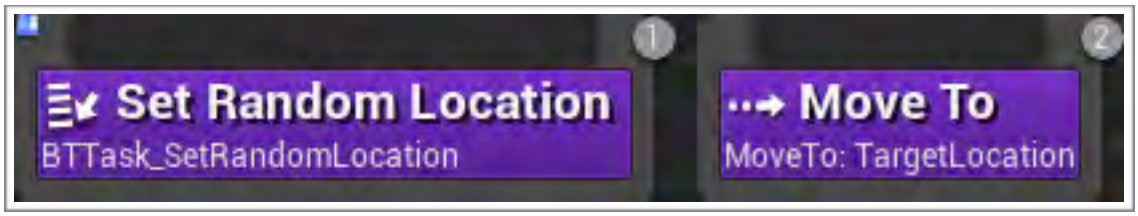


接下来，我们需要创建一个随机的位置，并将其保存在blackboard中。为此，我们将用到第三种行为树节点：service。

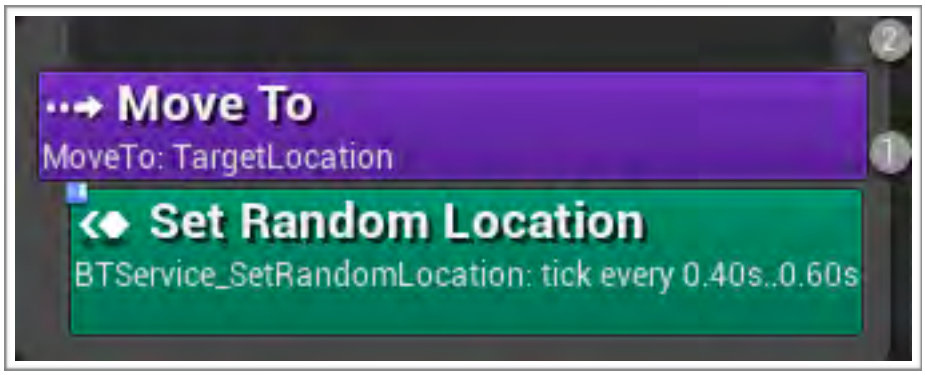
Services

和task类似，service用于完成某件任务。但是和task不同的是，task通常是让Pawn角色执行某个动作，而service则通常是用于执行检查或者更新blackboard。

services并非单独的节点，它们通常和tasks或composites关联在一起。通过使用services，可以让行为树的组织更加有条理。
当我们只使用task时，行为树会是这样的：



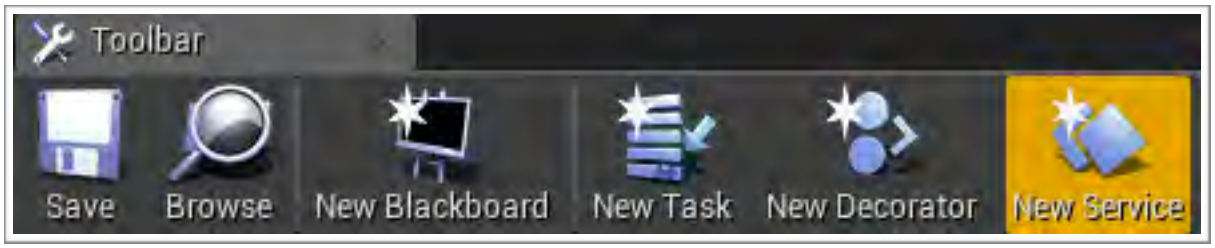
而当使用service时，行为树会是下面这样的：



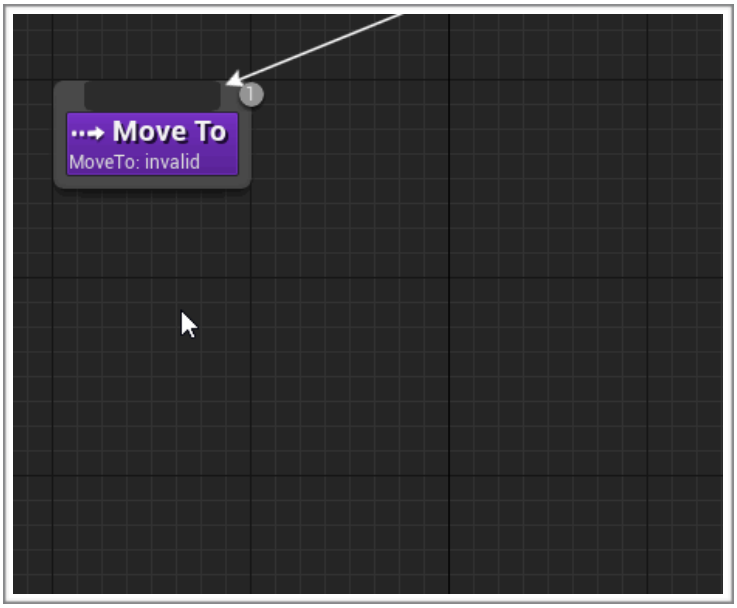
好了，接下来我们将创建一个service，用于生成一个随机的位置。

创建Service

返回BT_Muffin，点击New Service。



这样我们就创建了一个新的service,并且会自动将其打开。
将其命名为BTService_SetRandomLocation。（注意：需要返回Content Browser来重命名）。
该service只会在Pawn角色需要移动的时候执行。为此，我们需要将其关联到MoveTo节点上。
打开BT_Muffin，右键单击MoveTo，选择Add Services\BTService Set Random Location。

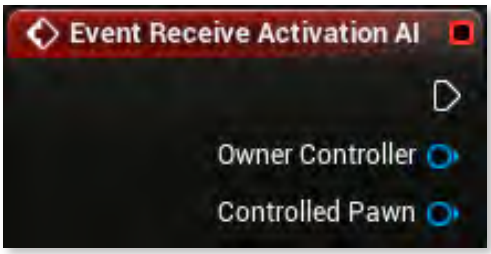


现在，当MoveTo节点激活的时候，BTService_SetRandomLocation也会激活。

接下来我们需要生成一个随机的目标位置。

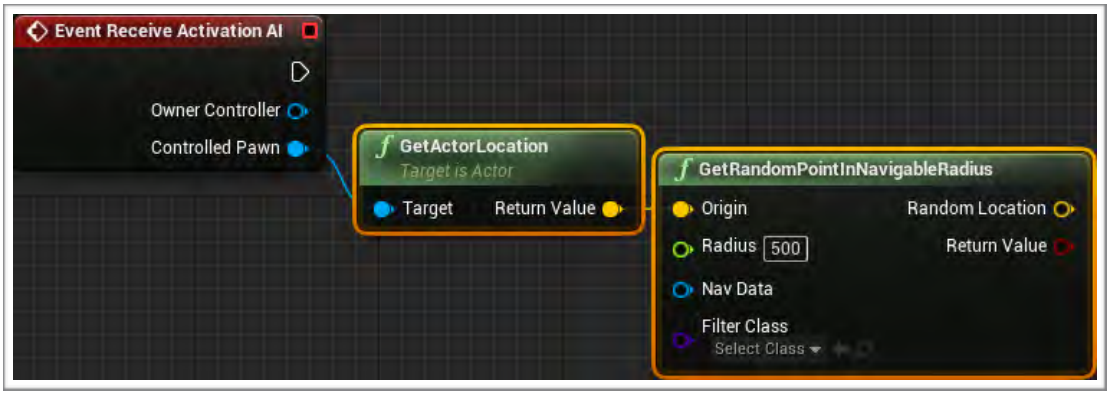
创建随机的位置

打开BTService_SetRandomLocation。
为了知道service何时激活，我们需要创建一个Event Receive Activation AI节点。这样，当service的父节点（所关联的节点）激活的时候，就会执行该节点。



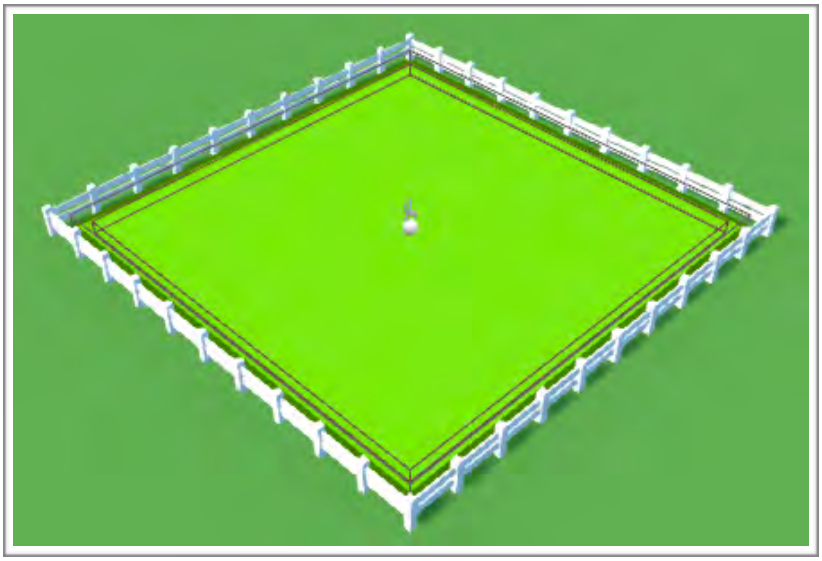
注意：除了Event Receive Activation AI节点，还有另外一个Event Receive Activation节点也可以完成同样的任务。两者的区别在于Event Receive Activation AI节点也提供了Controlled Pawn。

为了生成随机位置，还需要添加如下的高亮节点。注意需要将Radius设置为500。



这样，我们就获得了距离Pawn角色500个单位内的随机可寻路的位置。

注意：GetRandomPointInNavigableRadius使用寻路数据（称之为NaMesh）来判断某个点是否可到达。在本教程中已经创建好了NavMesh，大家在Viewport中选中Show\Navigation，就可以看到NavMesh。



如果我们希望创建自己的NavMesh，那么可以创建一个Nav Mesh Bounds Volume。使用缩放工具，从而让它涵盖我们希望到达的区域范围。

好了，本课的内容就先到这里了。
我们下一课继续~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

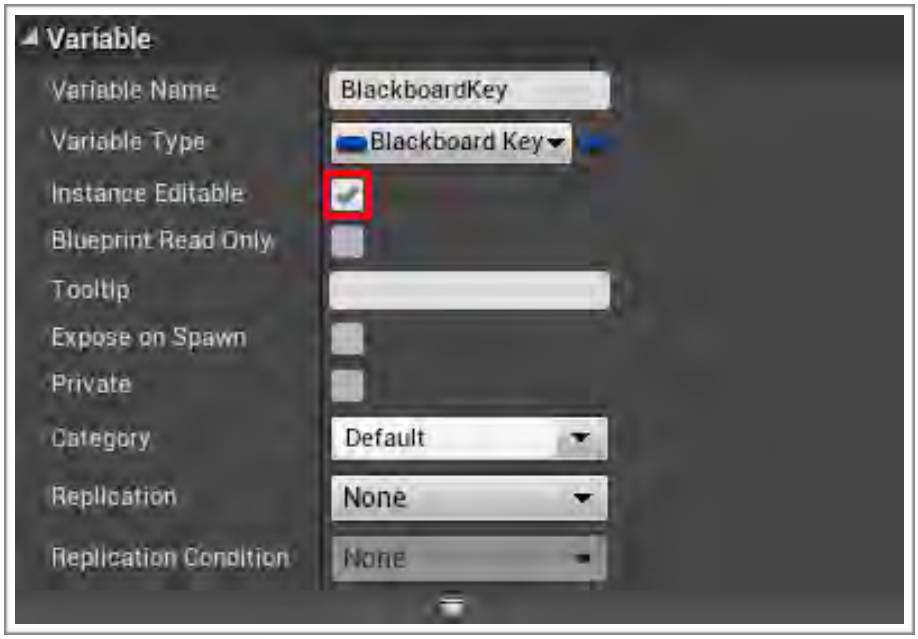
新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

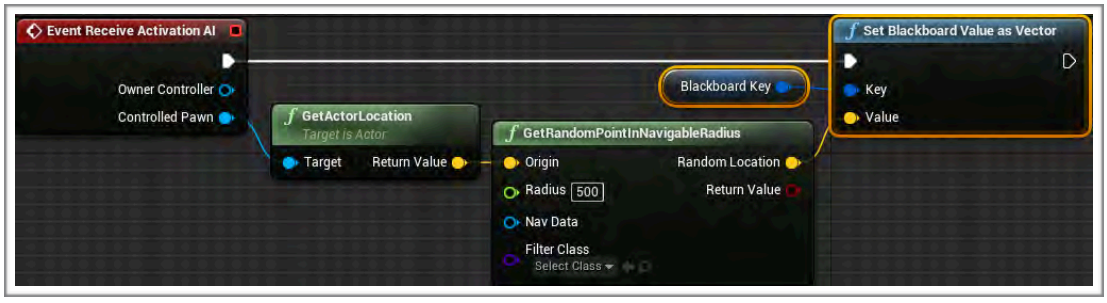
个人网站：
http://icode.ai/

欢迎继续我们的学习。
接下来我们将学习如何将Location信息保存到blackboard中。有两种方式可以用来指定要使用的key:

1.在Make Literal Name节点中使用其名称来指定key
2.将某个变量暴露给行为树，这样就可以从下拉列表中选择一个key。
这里我们将使用第二种方法。创建一个类型为Blackboard Key Selector的变量。将其命名为BlackboardKey，并启用Instance Editable。这样，当我们在行为树中选择service时，就会看到该变量。



接下来创建下图中的高亮节点：



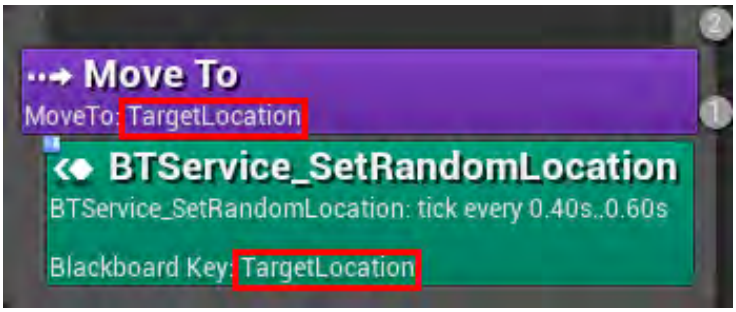
小结：
1.Event Receive Activation AI节点将在其父节点（这里是MoveTo)激活时执行

2.GetRandomPointInNavigableRadius将返回一个随机的可寻路位置，分布在受控制的松饼的500单位范围内。
3.Set Blackboard Value as Vector将把blackboard key（由BlackboardKey提供）的值设置为随机位置
点击工具栏上的Compile按钮，然后关闭BTService_SetRandomLocation。
接下来，我们需要通知行为树使用blackboard。

选择Blackboard

打开BT_Muffin，确保不选中任何东西。在Details面板中，在Behavior tree部分将Blackboard Asset设置为BB_Muffin。

接下来，MoveTo节点和BTService_SetRandomLocation节点将会自动使用第一个blackboard key，在这里就是TargetLocation。

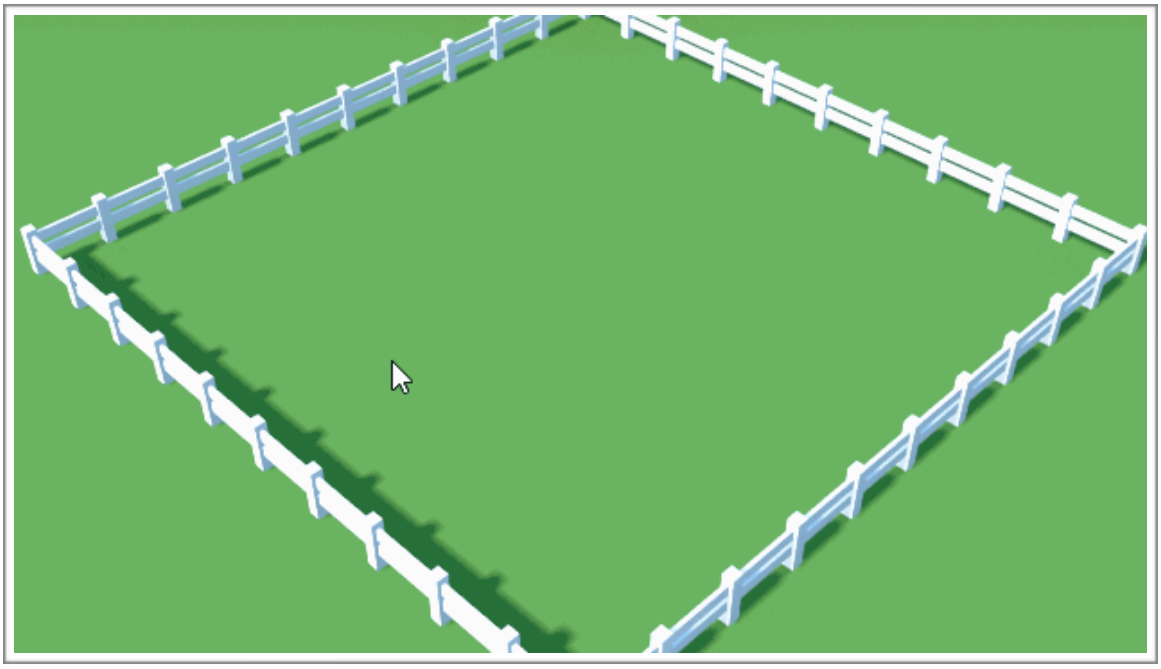


最后，我们需要通知AI Controller来使用行为树。

运行行为树

打开AIC_Muffin，将Run Behavior Tree节点连接到Event BeginPlay节点上。将BTAsset设置为BT_Muffin。

这样，当AIC_Controller生成时会运行BT_Muffin。
点击Compile 按钮，返回主编辑器。点击Play,生成一些松饼，然后看它们在场景中自由漫步~



好了，这一课的内容就先到这里了，我们下一课再见。

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

个人网站:
http://icode.ai/

欢迎继续我们的学习。
接下来我们需要设置AI controller，让它自动检测到视野范围内的敌人。为此，我们需要用到AI Perception。

设置AI Perception

当我们给角色添加了AI Perception之后，可以让AI具备senses感知（视力和听力）。
打开AIC_Muffin，然后添加AIPerception组件。

接下来，我们需要添加一种感知。因为我们希望检测另一个松饼何时进入视野，所以需要添加一个sight感知。
选中AIPerception，在Details面板中的AI Perception部分，给Senses Config添加一个新的元素。

将element 0设置为AI Sight config，然后将其展开。

对于sight感知有三个主要的设置选项：

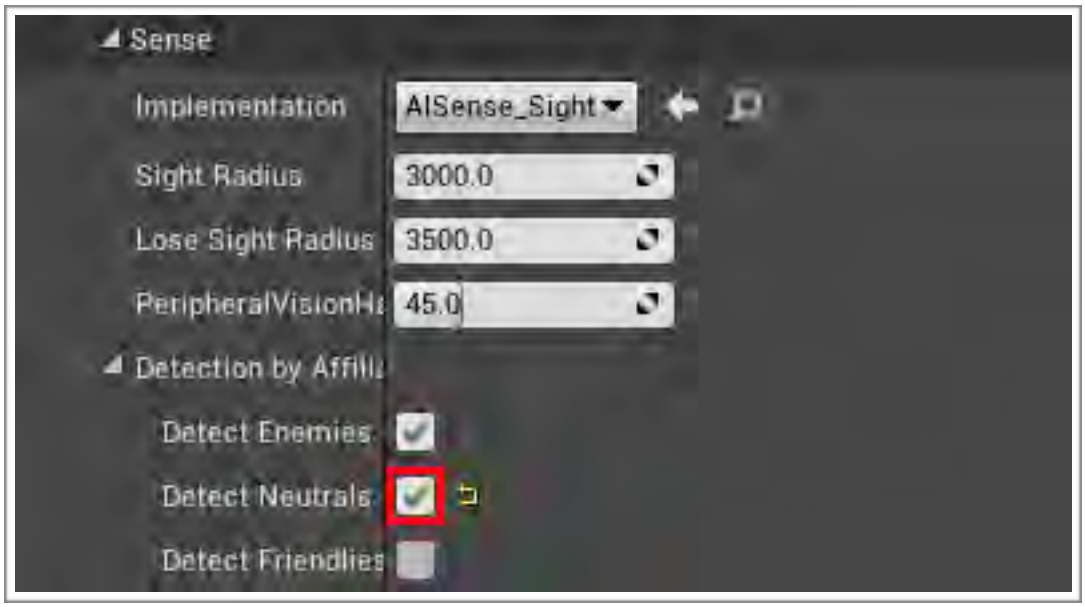
1.Sight Radius: 松饼可以“看到”的最远距离，这里设置为3000.

2.Lose Sight Radius: 如果松饼看到了一个敌人，那么这个值就代表着敌人需要移动多远才能从松饼的视野中消失，这里设置为3500

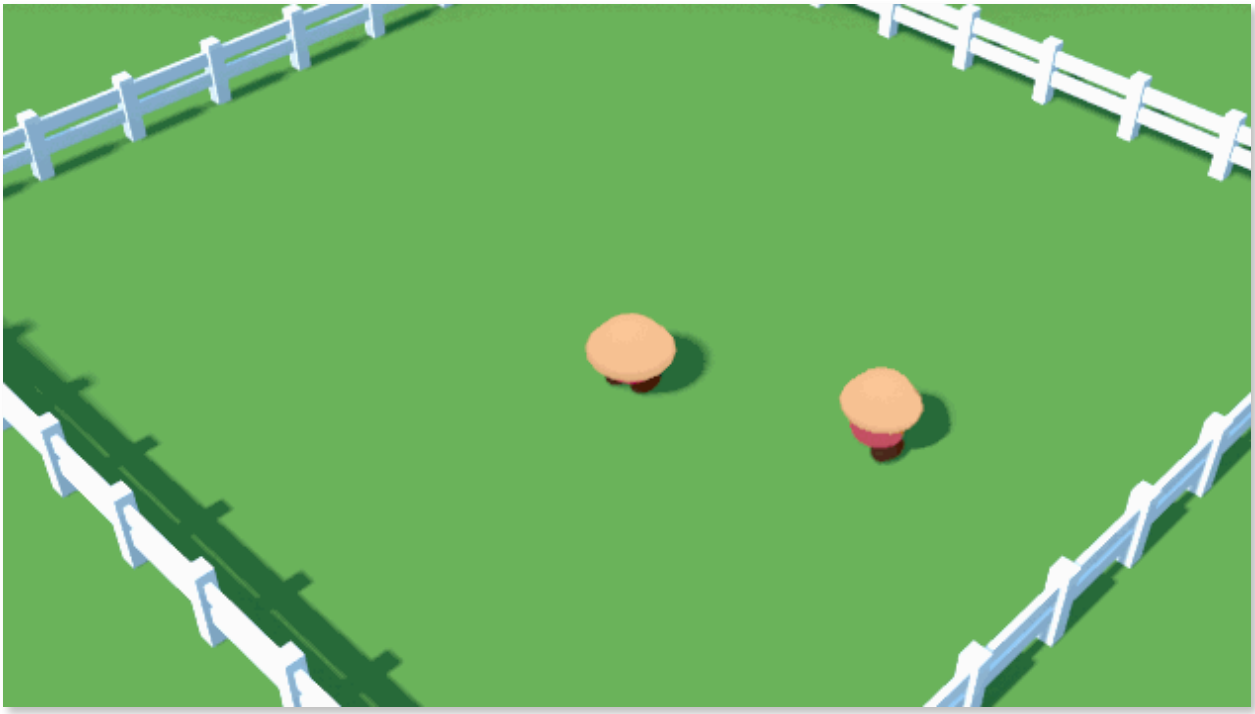
3.Peripheral Vision Half Angle Degrees:松饼的视角，这里设置为45，意味着松饼将拥有90度的视角。

默认情况下，AI Perception只会检测到敌人（因为角色被分配到另外一个team中~）。不过默认情况下角色并没有设置team。当某个角色没有team时，AI Perception会认为它是neutral(中立的）。

在虚幻4的当前版本中，还没有很好的办法通过蓝图来分配team。不过我们可以通知AI Perception来检测中立的角色。为此，需要展开Detection by Affiliation，然后启用Detect Neutrals。



点击工具栏上的Compile，然后返回主编辑器。点击Play以便生成一些松饼。按下键盘上的'键来显示AI debug界面。按下数字键盘上的4让AI perception可视化。当某个松饼移动进事业之中后，就会出现一个绿色的警戒范围~



好了，这一课的内容就到此结束，我们下一课再见~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

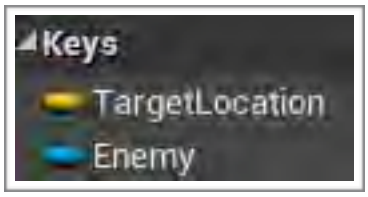
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

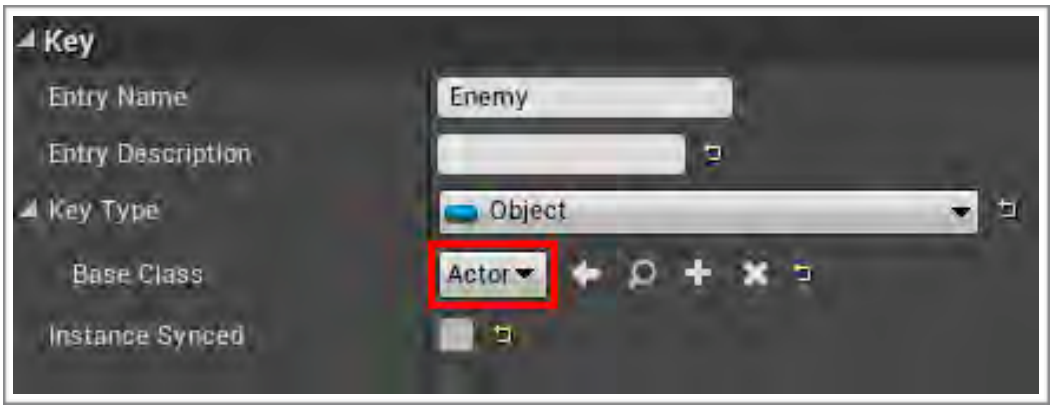
欢迎继续我们的学习。
接下来，我们将让松饼朝着敌人的方向进军。为此，行为树必须“知道”敌人。我们需要在blackboard中保存一个到enemy的引用。

创建一个Enemy Key

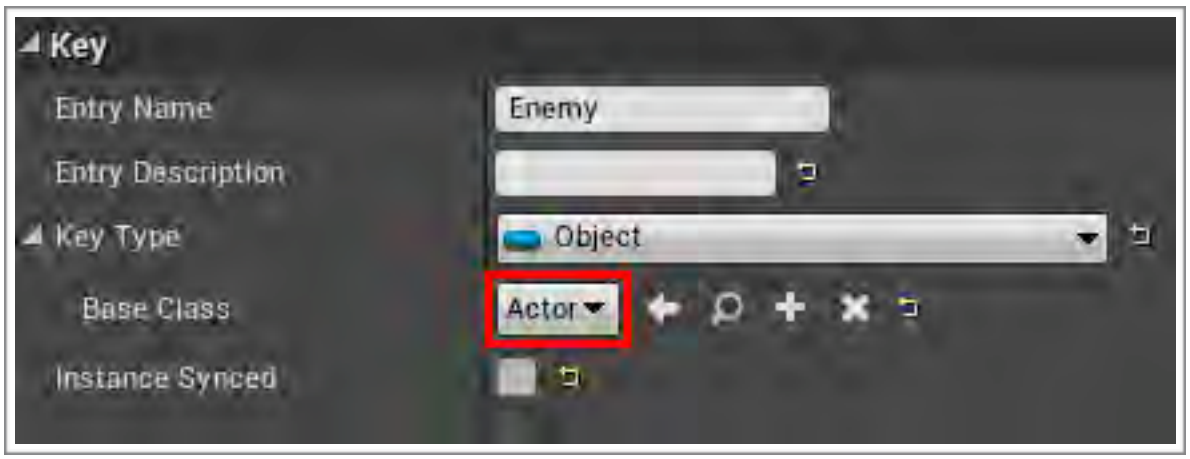
打开BB_Muffin，然后添加一个类型为Object的key，并将其更名为Enemy。



现在我们还没办法在MoveTo中使用Enemy。这是因为这个key属性Object类型，而MoveTo节点只接收Vector或Actor类型的key。



为此，选中Enemy，并展开Key Type。将Base Class设置为Actor，这样行为树就会把Enemy识别为Actor。

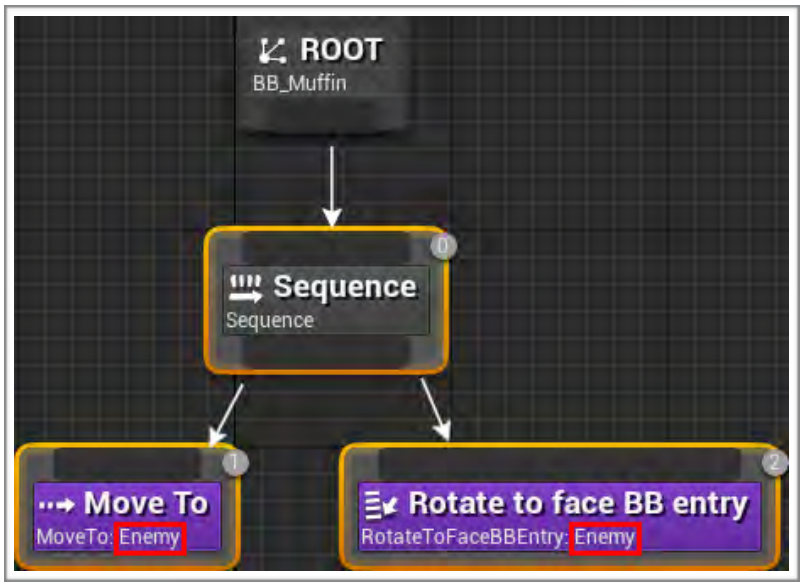


关闭BB_Muffin。现在，我们需要创建一个行为，以便让松饼向敌人靠近。

让松饼向敌人靠近

打开BT_Muffin，断开Sequence和Root节点时间的连接。断开连接很简单，只需要按住Alt键，然后点击连接两个节点之间的线就好了。暂时让漫游的子树放在一边。

接下来，创建如下的高亮节点，并将Blackboard Key设置为Enemy：

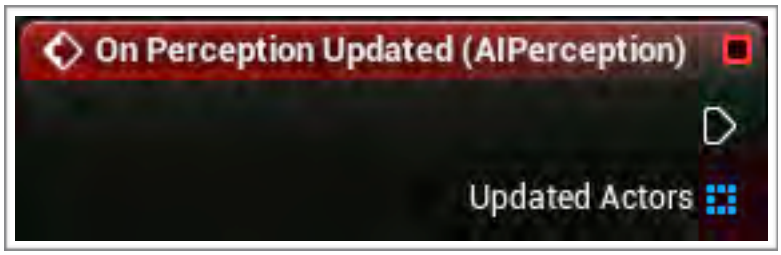


这样就会让Pawn角色向Enemy靠近。在部分情况下，Pawn角色并不会完全向目标靠近，所以我们还需要使用Rotate to face BB entry。

接下来，我们需要在AI Perception检测到另一个松饼的时候设置Enemy。

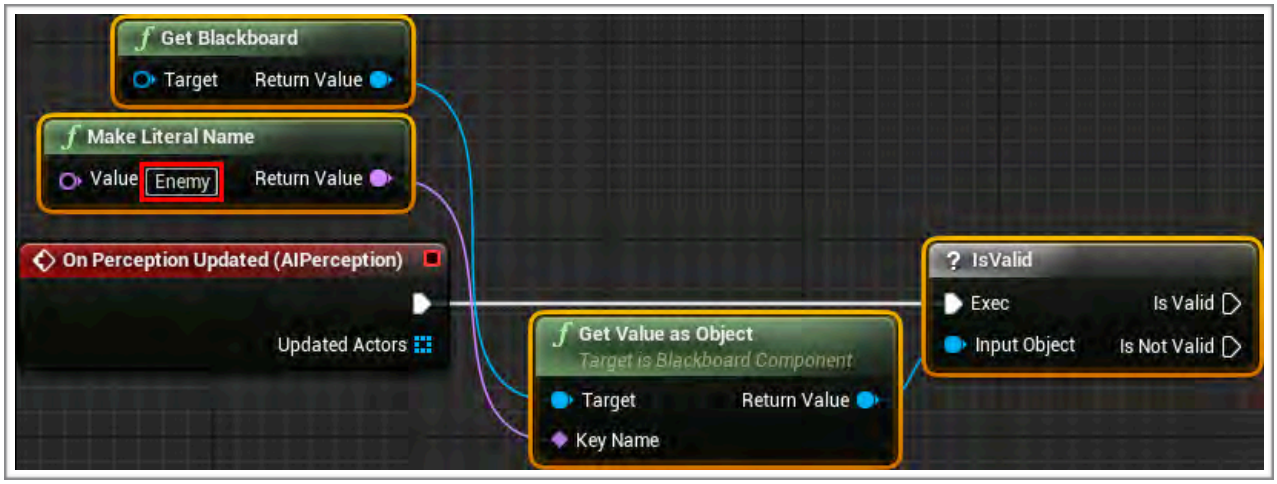
设置Enemy Key

打开AIC_Muffin，然后选择AIPerception组件。添加一个On Perception Updated event。



当sense更新的时候会执行该事件。在这里，指的就是当AI“看到”或者不再”看到“某个东西。该事件还将提供它当前所感知到的一系列角色。

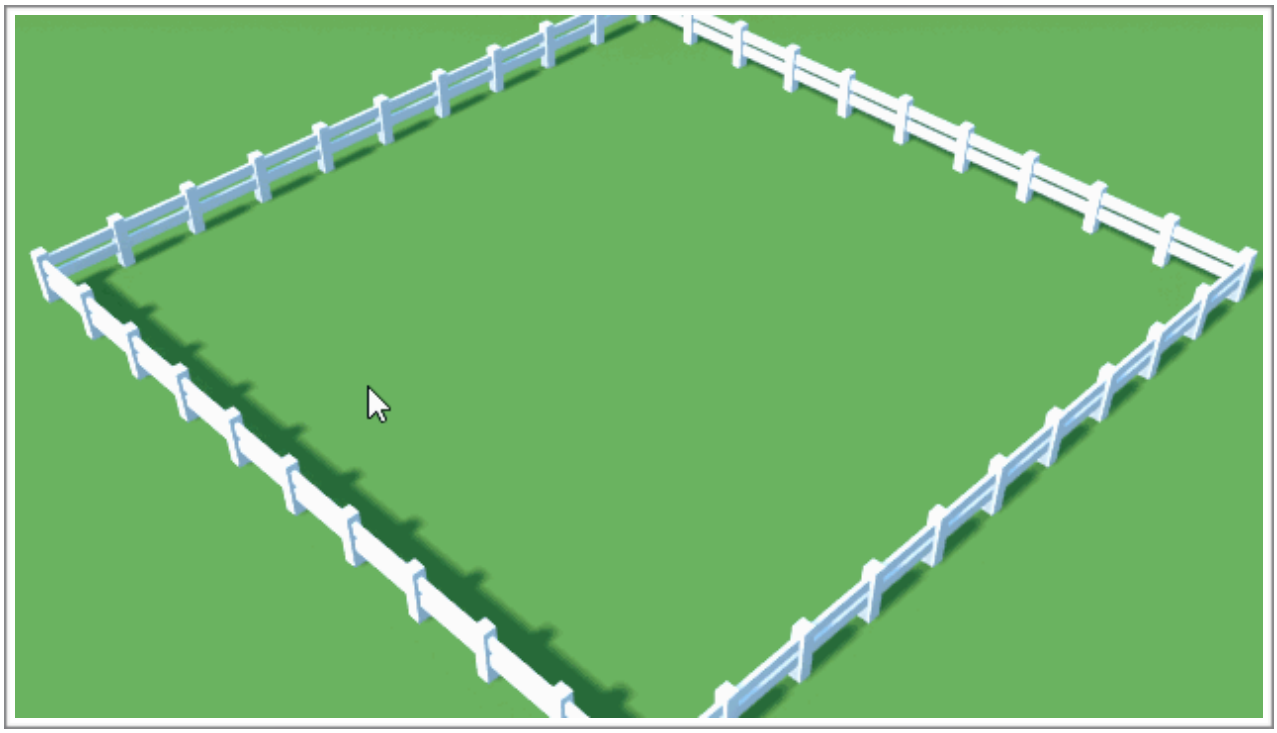
添加如下图中的高亮节点，确保将Make Literal Name设置为Enemy。



通过以上设置，可以检测AI是否已经有了敌人。如果还没有，我们需要指定一个。为此，添加以下的高亮节点：

小结：
1.IsValid为了检查Enemy key是否已设置。
2.如果还没有设置，那么将遍历当前所有已感知的角色
3.Cast To BP_Muffin将检查角色是否是松饼
4.如果是松饼，那么再检查它是否已死亡
5.如果IsDead返回false，那么将松饼设置为新的Enemy，然后中断循环。

点击Compile，并关闭AIC_Muffin。点击Play，然后生成两个松饼，其中一个在另一个的前面。这时，在后面的那个松饼将自动走向另一个松饼。



好了，本课的内容就到这里了，我们下一课再见~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

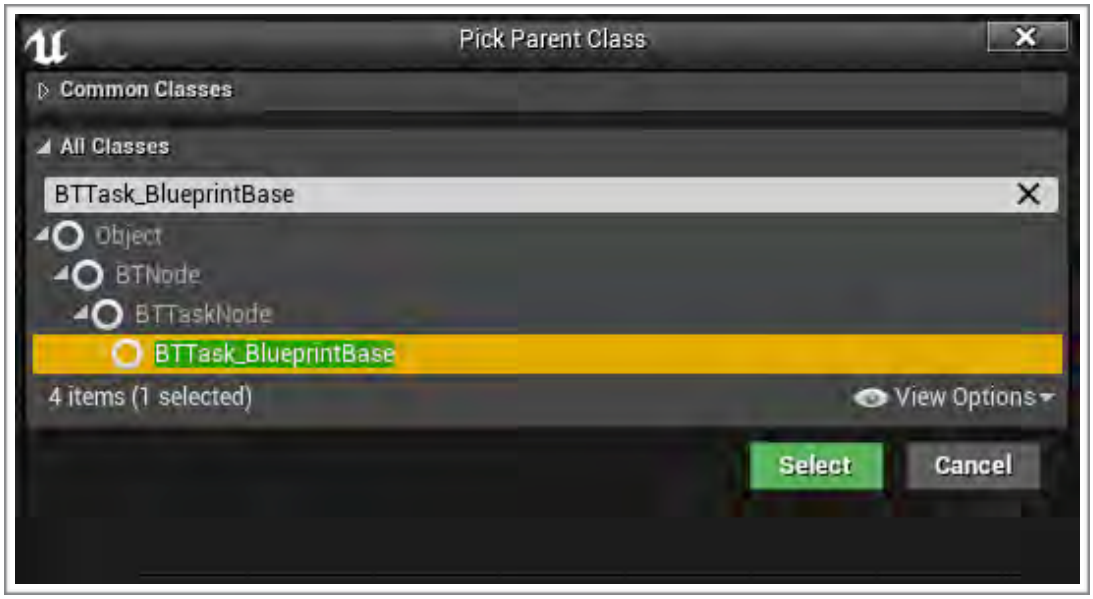
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

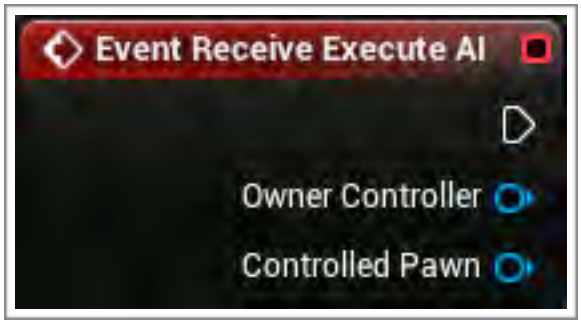
欢迎继续我们的学习。
接下来我们需要创建一个定制的task,可以让松饼执行攻击。

创建一个攻击任务（Attack Task)

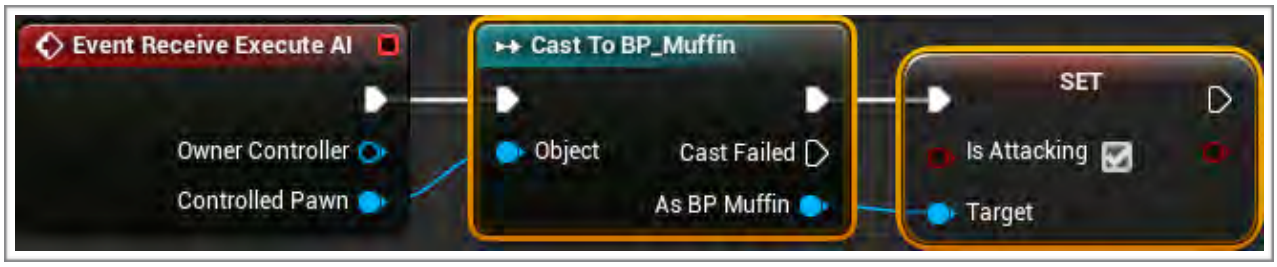
在Content Browser中创建一个task，注意不是在行为树编辑器中。创建一个新的Blueprint Class，选择BTTask_BlueprintBase作为父类。



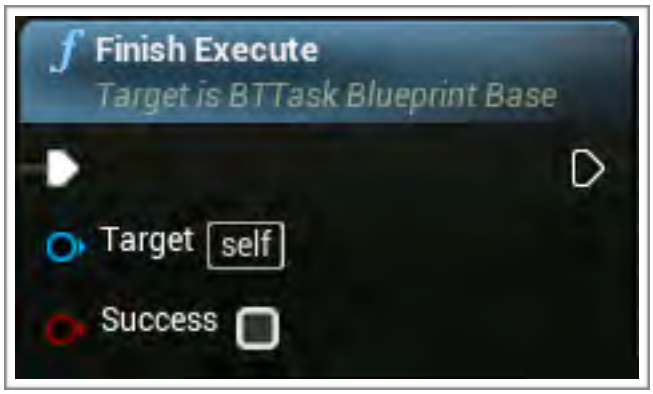
将其命名为BTTask_Attack，然后打开。添加一个Event Receive Execute AI节点。该节点会在行为树执行BTTask_Attack的时候执行。



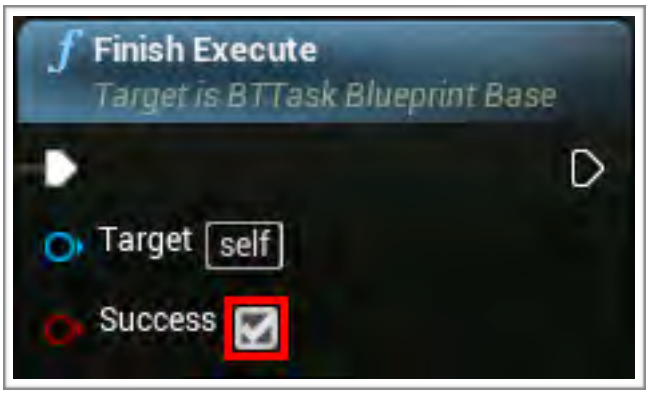
首先，我们需要让松饼展开攻击。BP_Muffin中包含了IsAttacking变量，一旦设置完成，就会执行攻击。为此，添加以下的高亮节点：



如果在当前状态下使用task,那么会执行不下去。这是因为行为树并不知道任务已经完成。为此，需要在节点链的最后添加一个Finish Execute节点。



接下来启用Success。因为这里使用了Sequence节点，因此节点将在BTTask_Attack之后执行。



下面是最终的图表连接：

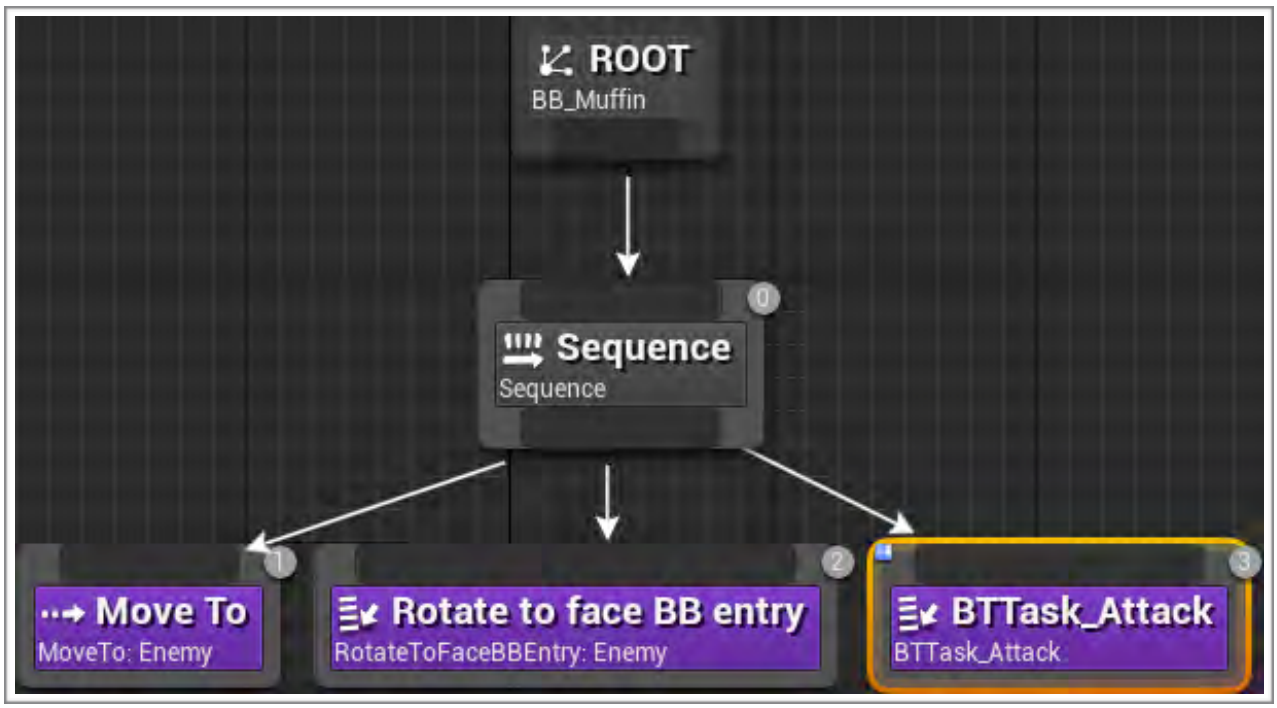
小结：
1.当行为树运行BTTask_Attack节点时，将执行Event Receive Execute AI节点。
2.Cast To BP_Muffin将检查Controlled Pawn的类型是否是BP_Muffin。
3.如果是，IsAttacking变量就会被设置。
4.Finish Execute节点将让行为树知道任务已顺利完成。

点击Compile，然后关闭BTTask_Attack。

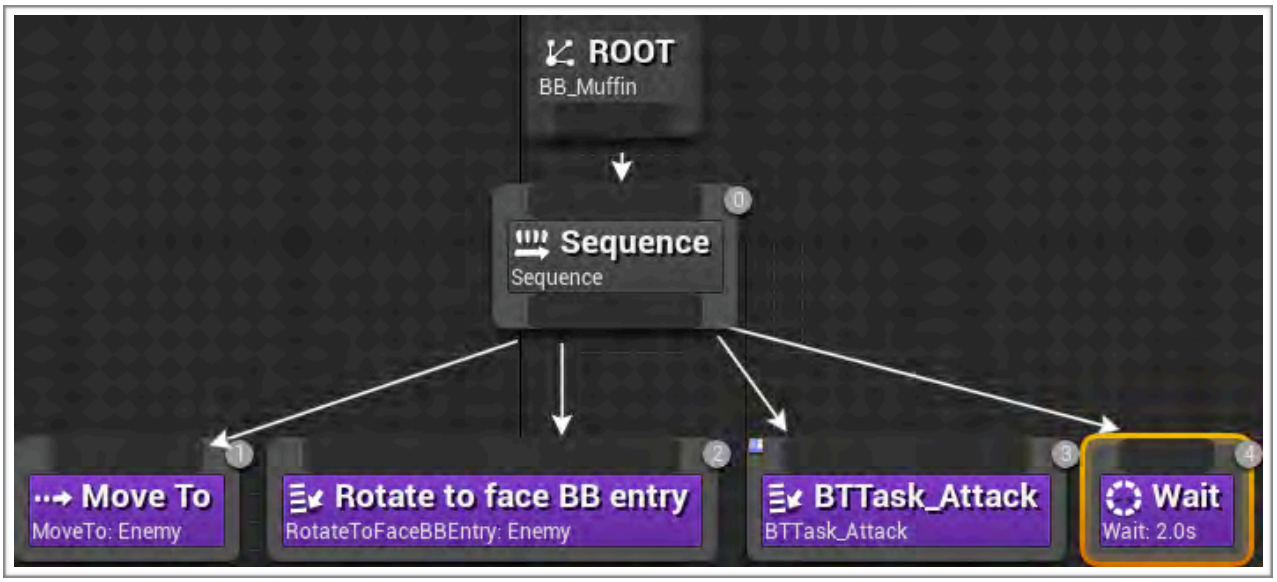
接下来，我们需要将BTTask_Attack添加到行为树。

将Attack添加到行为树

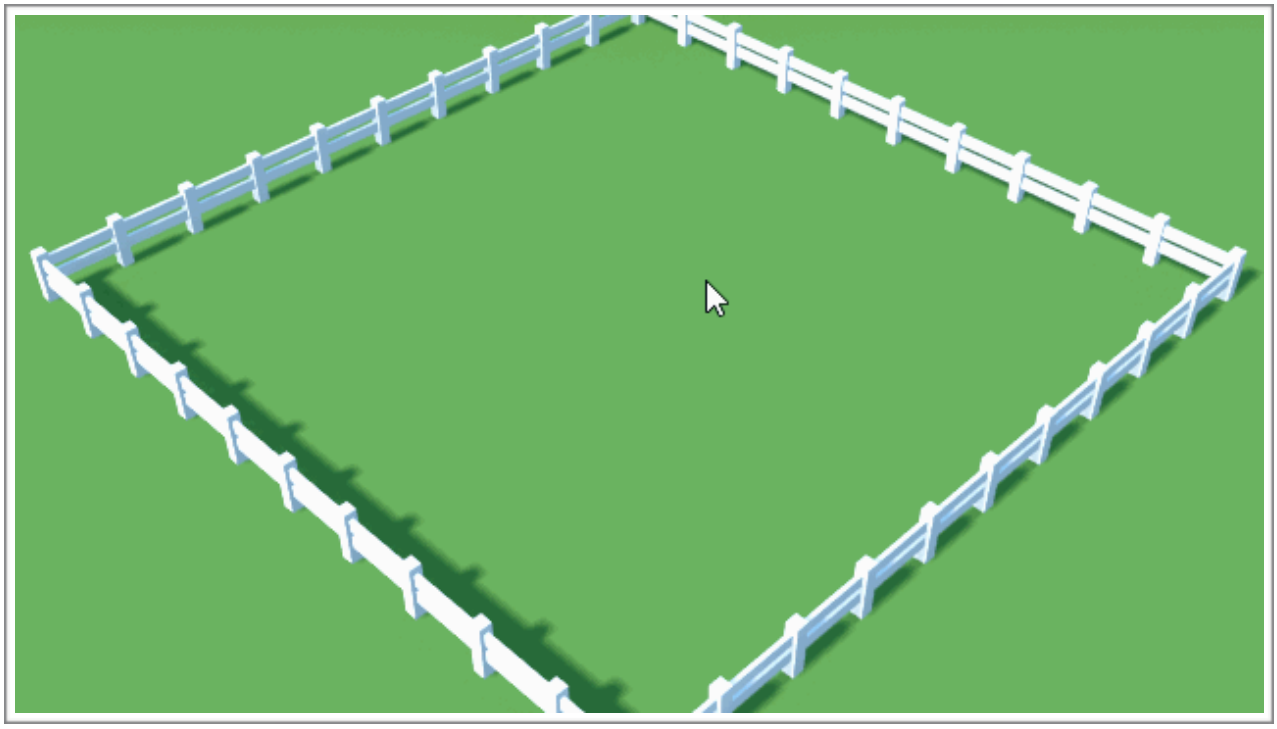
打开BT_Muffin。接着在Sequence节点的最后添加BTTask_Attack节点。



接下来添加一个Wait节点到Sequence的最后，将Wait Time设置为2.这样就不会让松饼持续不断的攻击。



返回主编辑器，点击Play预览游戏效果。和上次一样，生成两个松饼。松饼将向敌人的方向转过去并靠近。随后，它将攻击敌人，然后等待两秒。当看到另一个敌人后，整个Sequence将再次执行。



好了，本课的内容就到这里了，我们下一课再见~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

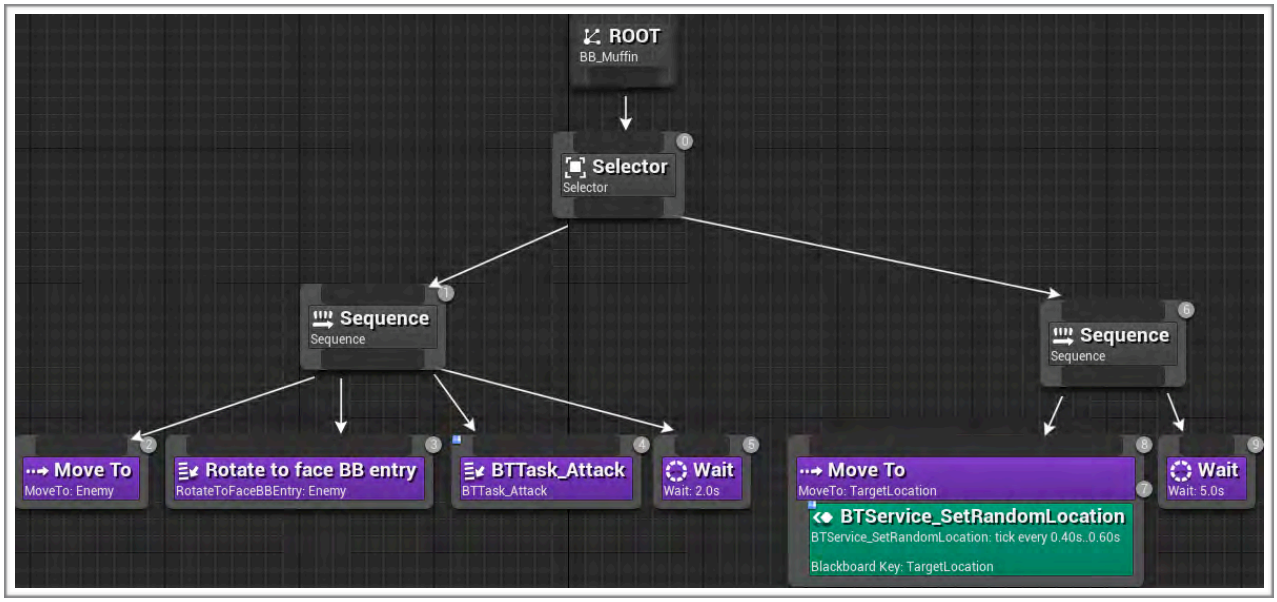
欢迎继续我们的学习。

本课也将是人工智能部分的最后一课。在本课的内容中，我们将把攻击和漫游的子行为树整合在一起。

将子行为树整合在一起

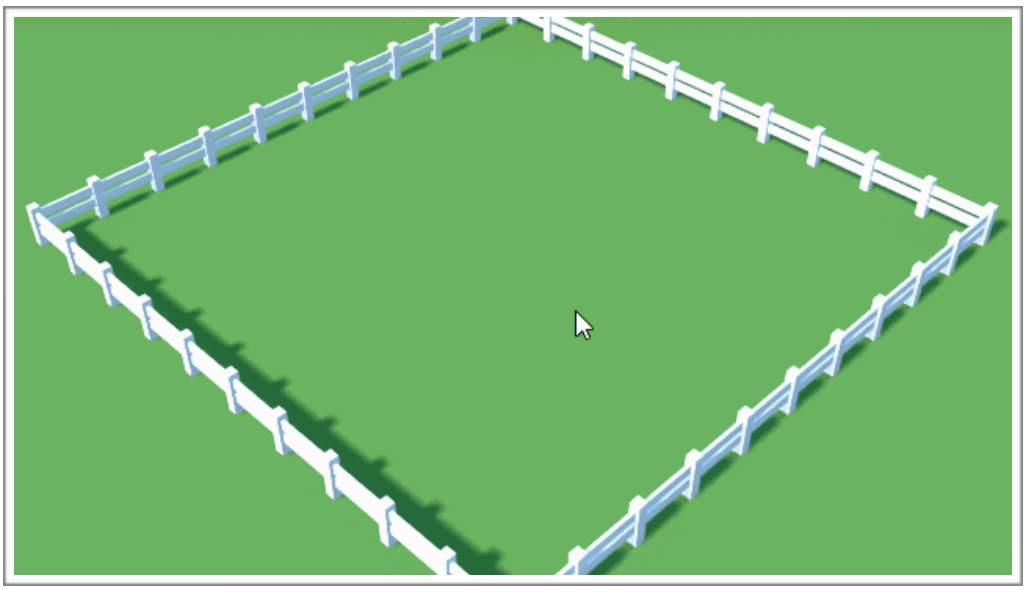
为了整合子行为树，我们需要用到Selector 这个composite。和Sequence节点类似，Selector中的子节点同样是从左向右执行的。不过不同的是，Selector将在子节点顺利完成之后停止，而不是在失败之后停止。通过使用Selector，我们可以确保行为树每次只会执行一种子行为树。

打开BT_Muffin，然后在Root节点之后创建一个Selector。随后使用以下的方式来连接子行为树：



通过以上的设置，一次只会执行一个子行为树。下面是每个子行为树的执行方式：
1.Attack: Selector将首先执行攻击子行为树。当完成所有任务后，Sequence也将顺利完成。Selector会检测到这一点，然后停止执行对应的Sequence。这样就不会紧接着执行漫游的子行为树。
2.Roam:Selector将优先执行攻击子行为树。在执行攻击的子行为树时，如果Enemy没有被设置，那么MoveTo节点就会执行失败。此时该部分的Sequence将执行失败，因此Selector将会执行另一个子行为树，也就是漫游。

返回主编辑器，点击Play。生成一些松饼，并测试一下效果。



不过问题来了，为什么松饼并不会立即攻击其它的松饼?
在传统的行为树中，将会从根节点每帧开始执行。这就意味着每次都会先尝试攻击子行为树，然后才是漫游子行为树。这就意味着如果Enemy的值发生变化，那么行为树的执行也会发生变化。

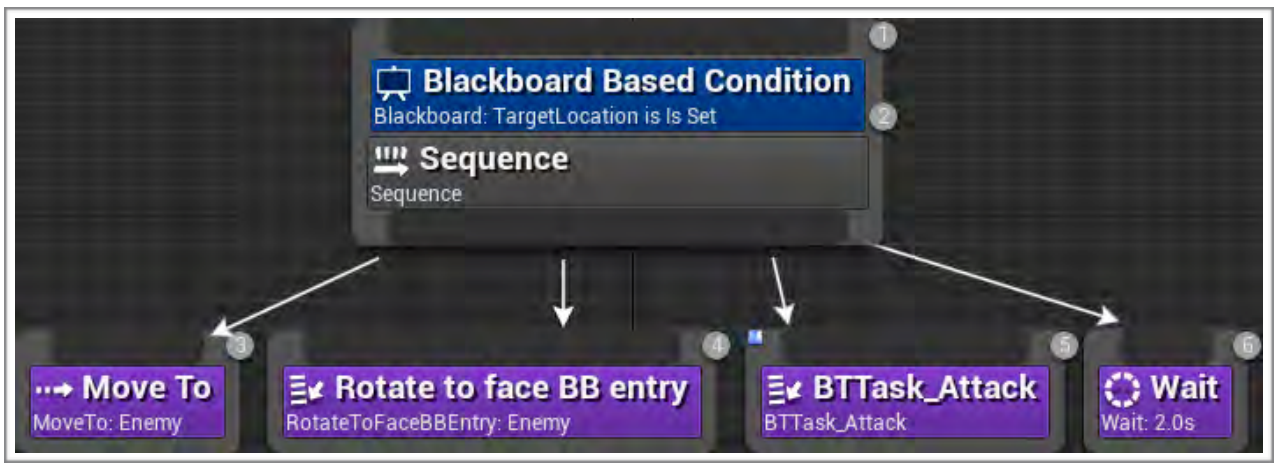
不过虚幻4中的行为树工作原理并非如此。在虚幻4中，总是从上一次执行的节点处开始。因为AI Perception不能立即感知其它的角色，所以漫游子行为树会开始执行。然后行为树在切换到攻击子行为树之前，需要等待漫游的子行为树结束。

为了修复这个问题，我们需要用到最后一种类型的节点：decorators。

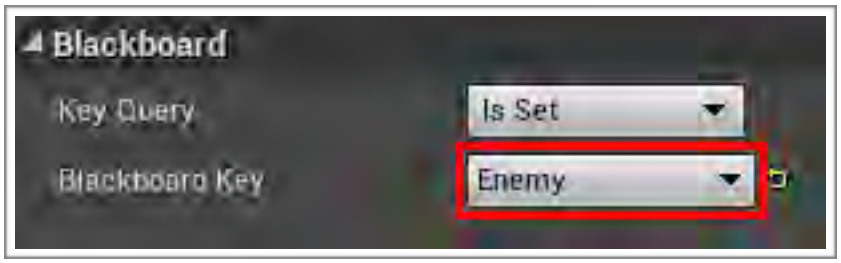
创建Decorator

和service类似，decorator将会关联到task或composite.通常来说，我们会使用decorator来执行检查。如果检查的结果是true，那么decorator将会返回true，反之则返回false。通过这种方式，我们可以控制decorator的父节点是否可以执行。

decorator还可以用来中止(abort)某个子行为树。这就意味着一旦Enemy被设置后，可以停止之前的漫游子行为树。这样就可以让松饼在检测到敌人之后立即开始攻击。
为了使用abort，我们可以使用Blackboard decorator。打开BT_Muffin，然后右键单击攻击子行为树的Sequence节点。选择Add Decorator\Blackboard。这样就可以在Sequence节点上关联一个Blackboard decorator。



接下来选中Blackboard decorator，然后在Details面板中将Blackboard Key设置为Enemy。



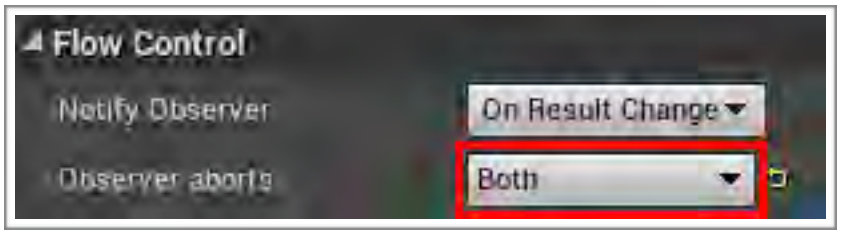
这里将会检查Enemy是否被设置。如果还没有被设置，那么decorator就会执行失败，并导致Sequence执行失败。这样漫游子行为树就会被执行。

为了中止漫游子行为树，我们需要用到Observer Aborts设置。

使用Observer Aborts

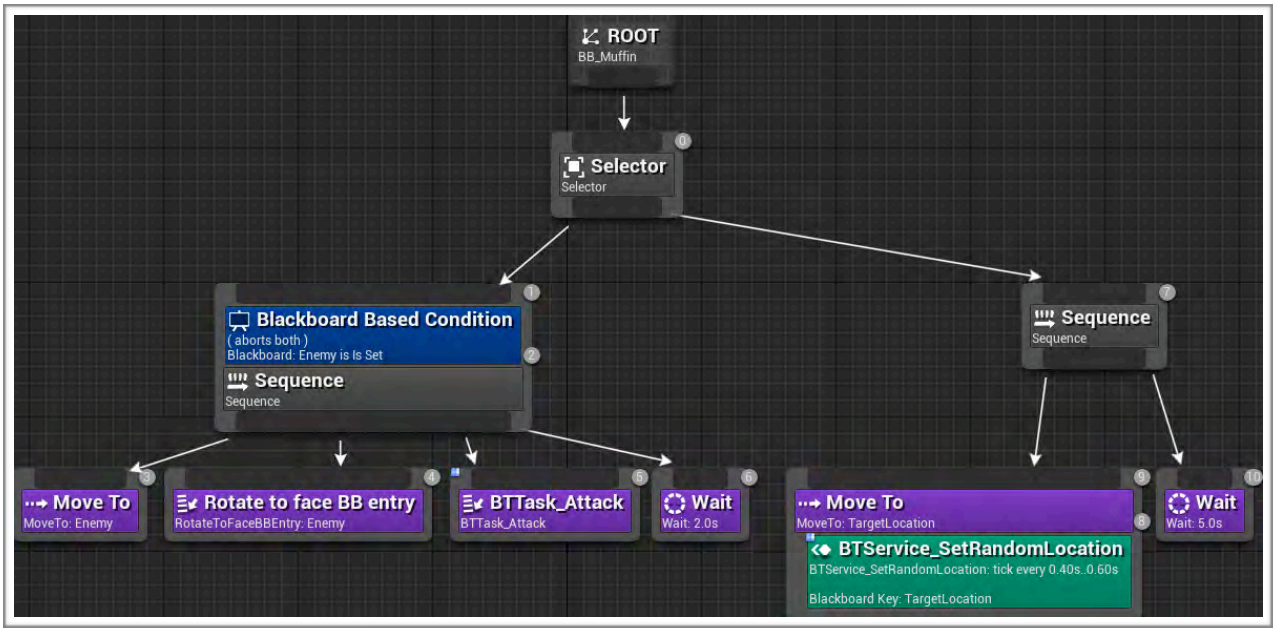
当所选的blackboard key被改变时，Observer aborts就会中止子行为树。有两种类型的aborts:
1.Self
这种设置将会在Enemy变得无效时让攻击子行为树中止自身。如果Enemy在攻击子行为树完成之前死了，就会发生这种情况。
2.Lower Priority
使用这种设置，当Enemy被设置时将中止低优先级的子树。因为漫游子树在攻击子树之后，因此其优先级更低。

这里将Observer Aborts设置为Both，这样两种类型的Abort都会被启用。



现在，当没有敌人的时候，攻击子树就会立即切换为漫游子树。同样的，当检测到敌人的时候，漫游子树也将立即切换为攻击子树。

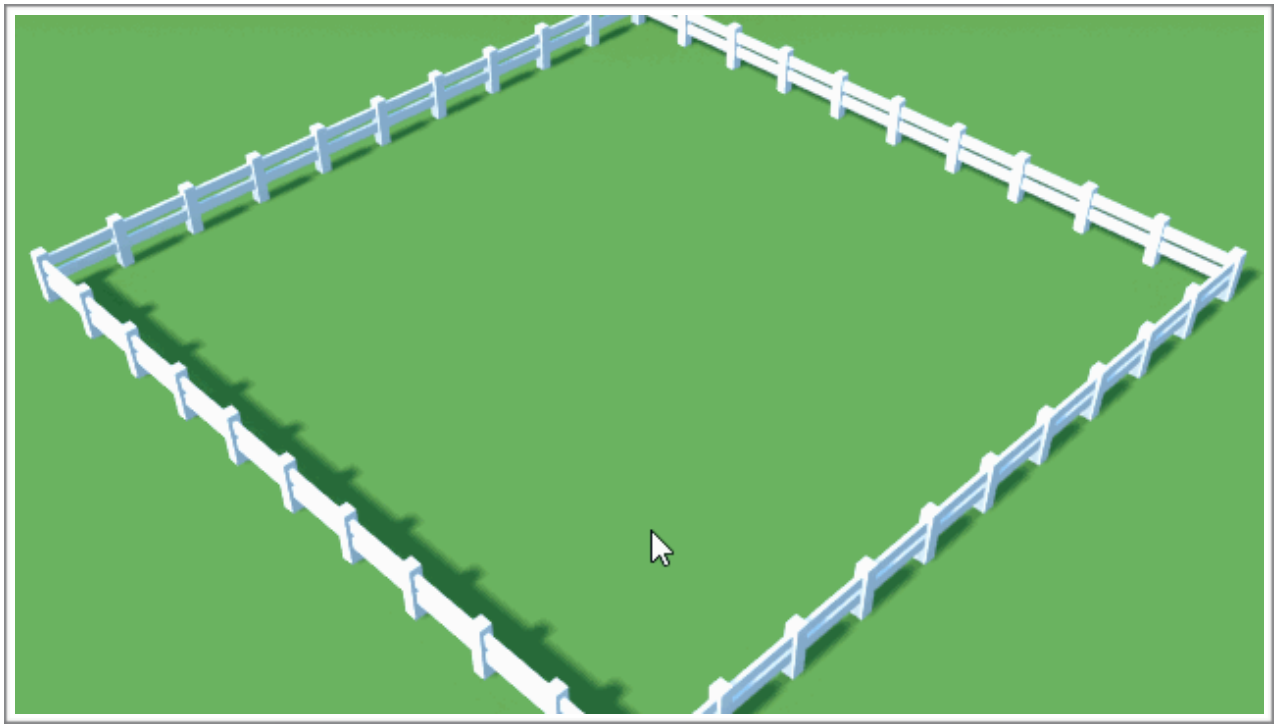
完整的行为树如下图所示：



这里大概解释下攻击子树的执行方式：
1.当Enemy被设置时，Selector将执行攻击子树
2.当Enemy被设置时，Pawn角色将会转向敌人，并向敌人靠近
3.随后角色将执行攻击
4.最后角色将等待两秒

接下来大概解释下漫游子树的执行方式：
1.当攻击子树执行失败的时候，Selector将会执行漫游子树。这里，执行失败的情况就是没有设置Enemy
2.BTService_SetRandomLocation将会生成一个随机的位置
3.Pawn角色将会向所生成的位置靠近
4.随后角色将等待5秒

关闭BT_Muffin，然后点击Play预览游戏。
在场景中生成一些松饼，然后看它们欢乐的战斗~



好了，关于游戏中的人工智能，就讲到这里。

在最后一部分的内容中，我们将学习如何创建一个简单的FPS射击游戏~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

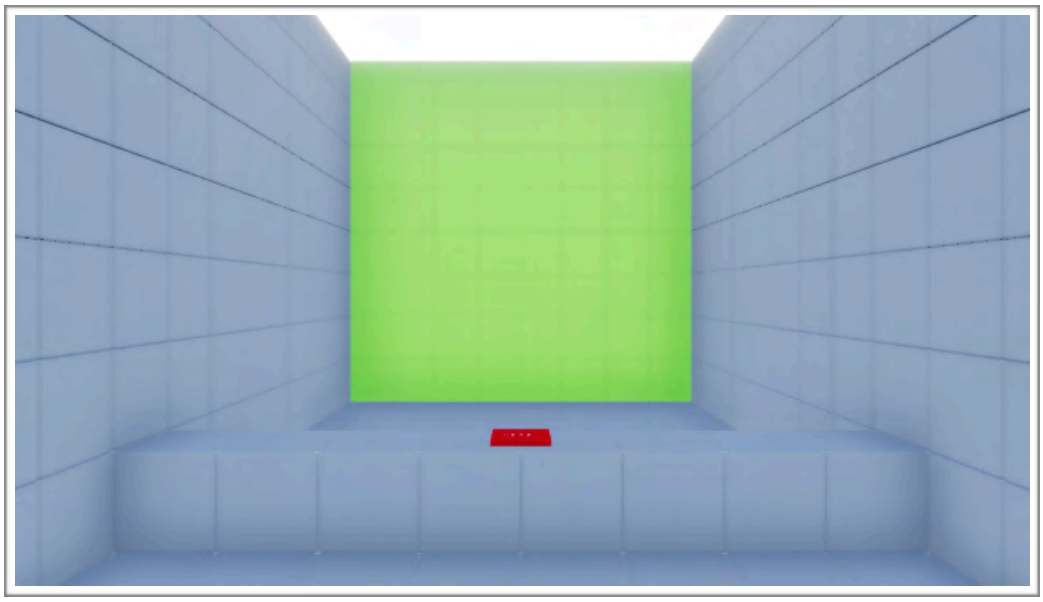
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续回到我们的虚幻4引擎学习之旅。
在之前的课程中，我们已经了解了关于虚幻4引擎的各个方面。而在这一课的内容中，我们将综合使用之前所学习的知识来创建一个简单的FPS游戏。
FPS(First-person Shooter，第一人称射击）游戏是游戏中的一种经典类型。玩家使用第一人称主视角来观察游戏中的场景，并使用各种武器和敌人对战。FPS游戏非常类型，经典的FPS游戏包括《使命召唤》，《CS》、《Overwatch》等。
实际上，虚幻4引擎诞生之初就是用来开发FPS游戏的，所以使用虚幻4引擎开发FPS游戏，开发者会感到非常得心应手。在本系列的课程中，我们将学到以下内容：
1.创建一个第一人称的Pawn角色，可以移动并四处观察
2.创建一把枪作为武器，并将其关联到Pawn角色上
3.使用line trace（光线跟踪，又称之为raycast)来发射子弹
4.将伤害应用到角色上

开始前的准备

首先下载起始项目并将其解压缩。链接:https://pan.baidu.com/s/1dRx_LSBCXRHbSnok1V5fJQ
密码:w7tm
打开项目文件夹，双击打开BlockBreaker.uproject，可以看到以下场景：



其中绿墙中包含了多个目标。但这些目标受到伤害时就会变红，而一旦目标的生命值变为0，就会从场景中消失。红色的按钮可以用来重置所有的目标。

好了，一切准备就绪。

接下来我们需要创建玩家的Pawn角色。

创建玩家的Pawn角色

在主编辑器中打开Blueprints文件夹，然后创建一个新的Blueprint Class，并选择Character作为父类，将其命名为BP_Player。

Character是Pawn角色的一种类型，但是相比Pawn有一些额外的功能，比如CharacterMovement组件。

该组件可以自动处理如行走和跳跃之类的运动。只需简单调用合适的函数，就可以让Pawn角色移动。此外，还可以在该组件中设置如行走速度和跳跃速度之类的变量。
在我们让Pawn角色移动之前，需要让它知道玩家何时按下了运动键。为此，我们需要将运动映射到W,A,S和D键。
注意：如果你已经忘了应该如何进行键盘映射，可以返回到之前的蓝图基础教程。

好了，这一课的内容就先到这里，我们下一课再见~

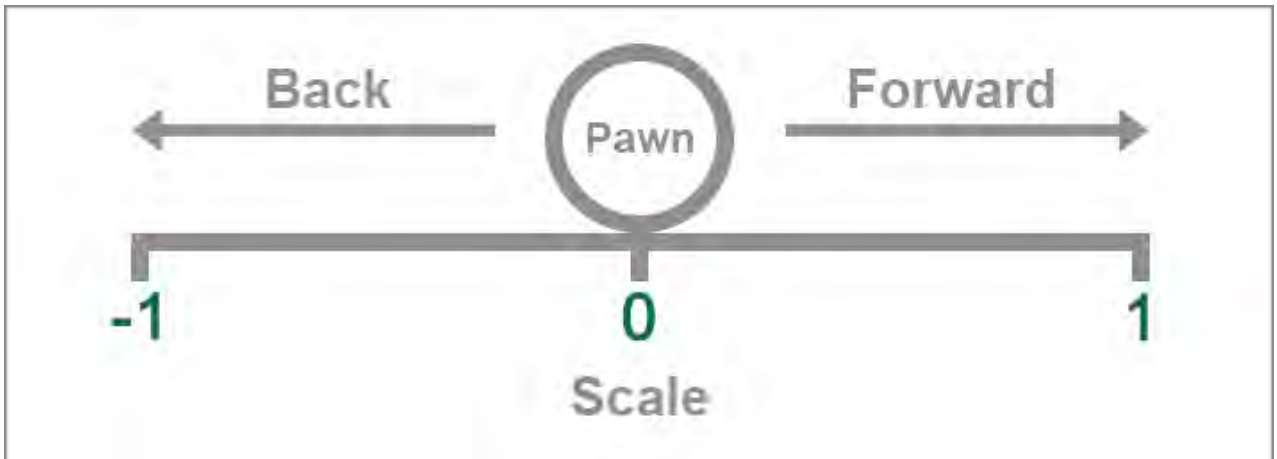
欢迎继续我们的学习。
接下来首先我们需要创建运动的映射

创建运动映射

在虚幻4编辑器的菜单中选择Edit\Project Settings，然后打开Input设置。
创建两个Axis Mappings，分别命名为MoveForward和MoveRight。其中MoveForward将用于处理前后方向的运动，而MoveRight则用来处理左右方向的运动。

对于MoveForward，将映射的键更改为键盘上的字母W，紧接着创建另外一个键，并将其设置为S。然后将S的比例设置为-1.0。

之后，我们将使用Pawn角色的前向向量乘以Scale比例。这样当Scale的数值为正数时，就会获得一个向前的向量。如果Scale的数值为负数，则会得到一个向后的向量。通过使用计算出的向量结果，就可以让Pawn角色前后移动。

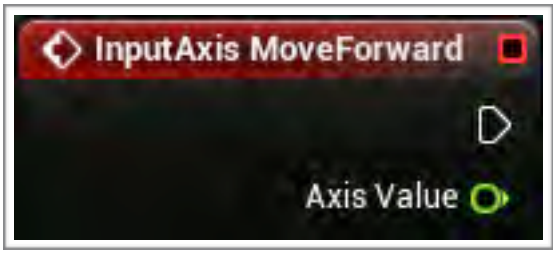


接下来，我们需要对左右运动执行类似的操作。将MoveRight的键设置为D。随后创建一个新的键，并将其设置为A。将A的Scale更改为-1.0。

现在键盘映射已经完成，接下来将使用它们来移动Pawn角色。

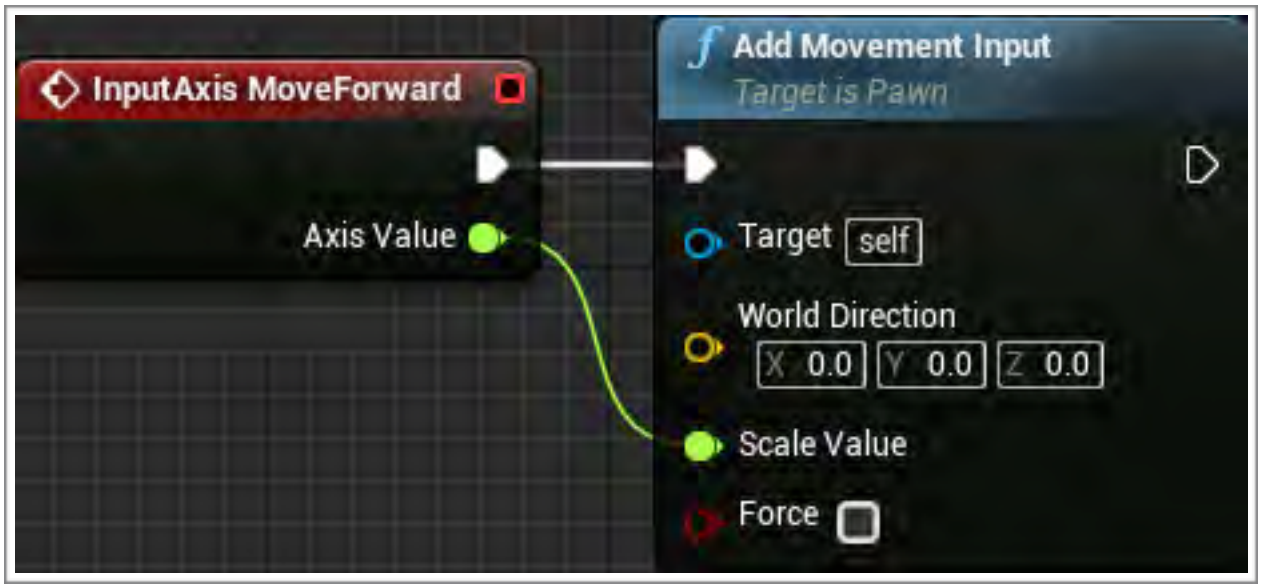
实现角色的运动

打开BP_Player，然后打开Event Graph视图。添加一个MoveForward事件（在Axis Events下面）。该事件将在游戏中的每一帧执行，即便我们什么也不按下。



该节点还将输出一个Axis Value，可以作为之前所设置的Scale数值。当我们按下W键时，该数值输出1，当按下S键时，输出-1。如果两个键都没有按，那么将输出0。

接下来，我们需要通知Pawn角色移动。为此，添加一个Add Movement Input节点，并使用以下方式来连接：



Add Movement Input节点将接收一个向量，并将其乘以Scale Value，从而将其转换为合适的方向。因为我们使用的是Character，所以CharacterMovement组件将向所生成的方向来移动Pawn角色。

接下来，我们需要指定角色移动的方向。因为我们希望向前运动，所以需要使用Get Actor Forward Vector。该节点将返回一个向前的向量。在视图中创建该节点，并使用下面的方式来连接：

小结一下：
1.MoveForward将在游戏的每一帧运行，并输出一个Axis Value值。当按下W键时将输出1，当按下S键将输出-1。如果两个键都没有按下，将输出0。
2.Add Movement Input节点将会把Pawn角色的forward vector乘以Scale Value。这样可以让向量根据所按下的键指向前方或者后方。如果不按下任何键，该向量将不会获得方向，也就意味着Pawn角色将不会移动。
3.CharacterMovement组件将从Add Movement Input节点中获取结果。然后它会让Pawn角色在该方向上运动

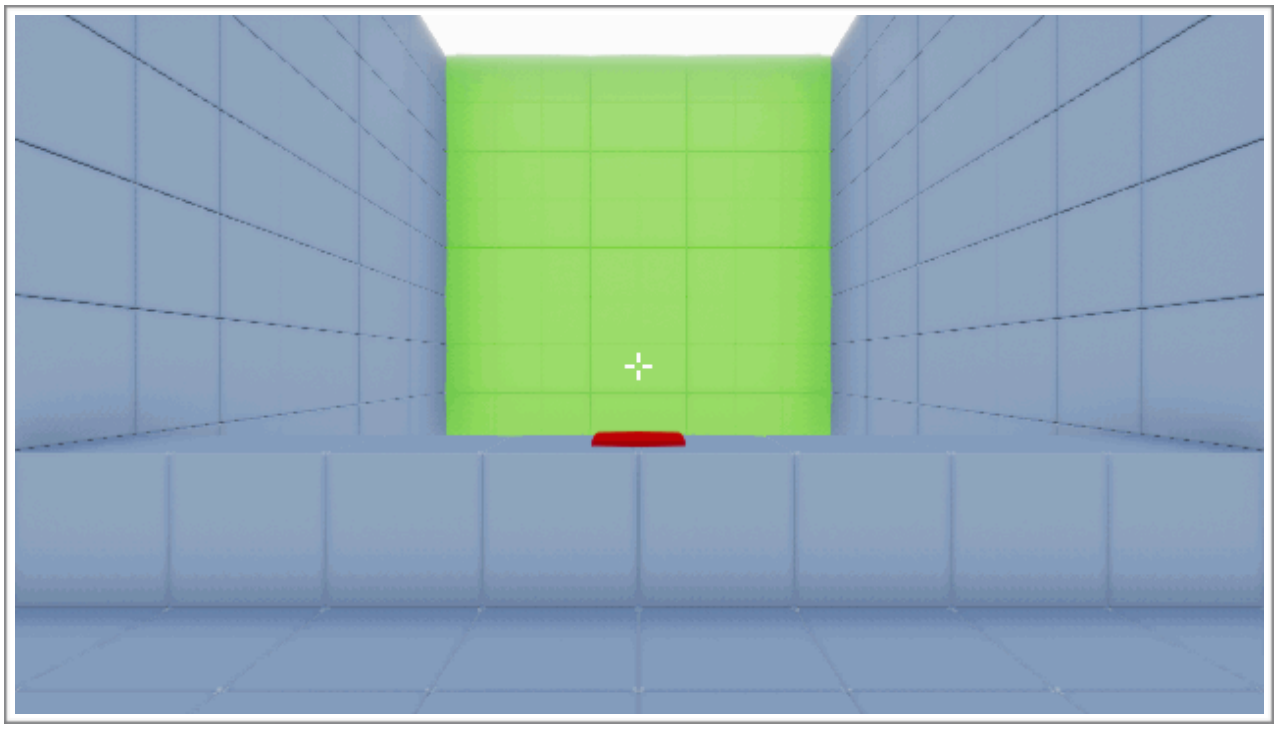
接下来对MoveRight节点重复类似的操作，只不过要把Get Actor Forward Vector换成Get Actor Right Vector。

在测试Pawn角色的运动之前，我们还需要在Game Mode中设置默认的Pawn角色。

设置默认的Pawn

点击蓝图编辑器工具栏上的Compile按钮，然后返回主编辑器。打开World Settings面板，找到Game Mode部分，将Default Pawn Class更改为BP_Player。

点击Play按钮，可以使用W,A,S,D键来左右前后移动了。



好了，本课的内容就先到这里了，我们下一课再见~讨论群-笨猫学编程QQ群：375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。
接下来我们需要创建映射，让玩家可以四处张望。

创建Look的映射

打开Project Settings，创建两个Axis Mappings，分别命名为LookHorizontal和LookVertical。

将LookHorizontal对应的键值更改为Mouse X。

这样，当我们向右方移动鼠标的时候，映射就会输出一个正值，反而则输出一个负值。
接下来将LookVertical对应的键值更改为Mouse Y。

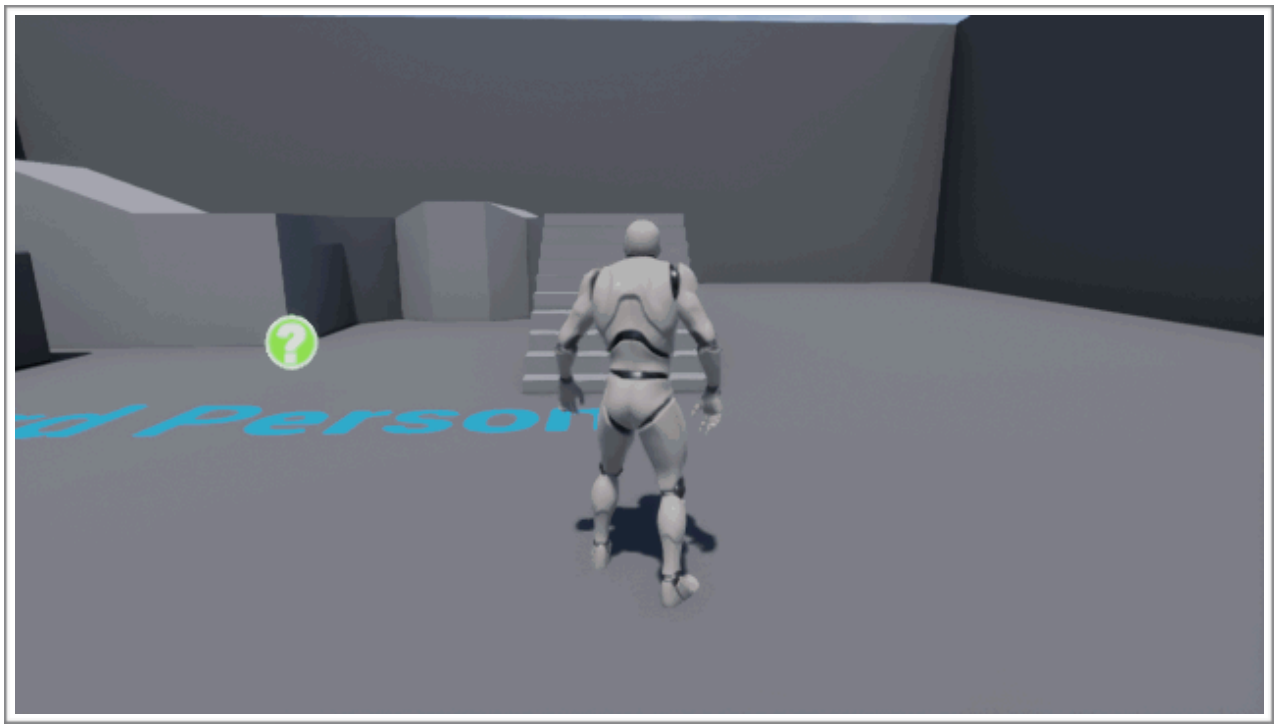
这样，当我们向上方移动鼠标的时候，映射就会输出一个正值，反而则输出一个负值。
接下来，我们需要创建让角色向四周观察的逻辑。

实现Look的逻辑

如果Pawn角色没有Camera组件，UE4会自动创建一个。默认情况下，该Camera会使用controller的rotation信息。

注意：如果想了解关于controller的更多信息，可以复习下AI系列的课程。

尽管controller是非物理的，它们仍然有自己的rotation。这就意味着我们可以让Pawn角色和摄像机面向不同的方向。例如，在第三人称游戏中，角色和摄像机通常不会面向同一个方向。

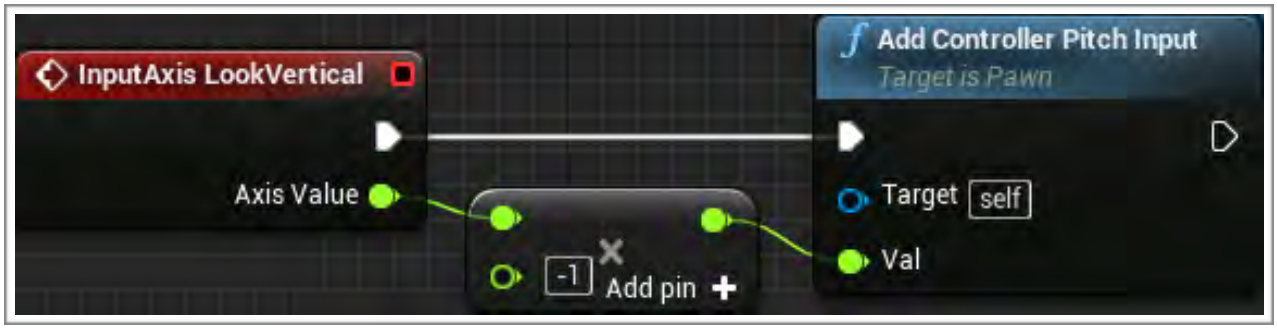


为了在第一人称游戏中旋转摄像机，我们只需要更改controller的rotation信息。
打开BP_Player，然后创建一个LookHorizontal事件。

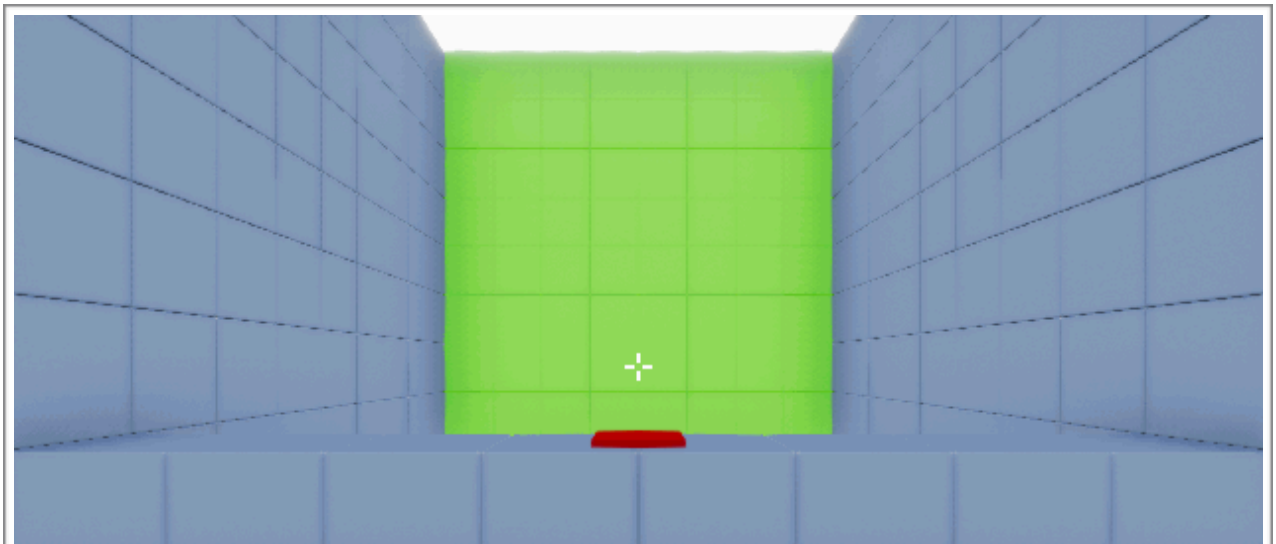
为了让摄像机可以向左或者右方旋转，我们需要设置controller的yaw。
在视图中创建一个Add Controller Yaw Input节点，然后使用下面的方式来连接节点：

现在，当我们在水平方向移动鼠标的时候，controller将会向左侧或者右侧旋转。而因为摄像机使用了controller的rotation信息，所以它也会向左右侧旋转。
接下来对LookVertical重复类似的过程，使用Add Controller Yaw Input节点替代Add Controller Pitch Input节点。

如果现在立即测试游戏，会发现垂直方向的查看是反向的。这就意味着当我们向上移动鼠标的时候，摄像机会向下看。
如果你是个强迫症患者，那么显然需要让Axis Value乘以-1。这样就会将Axis Value转换为所需的数据。



点击工具栏上的Compile按钮，然后点击Play按钮，移动鼠标来四处观察游戏场景。



好了，现在基本的运动和观察控制都已经设置完毕，接下来就可以来点好玩的了。
这一课的内容就到这里了，我们下一课再见~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

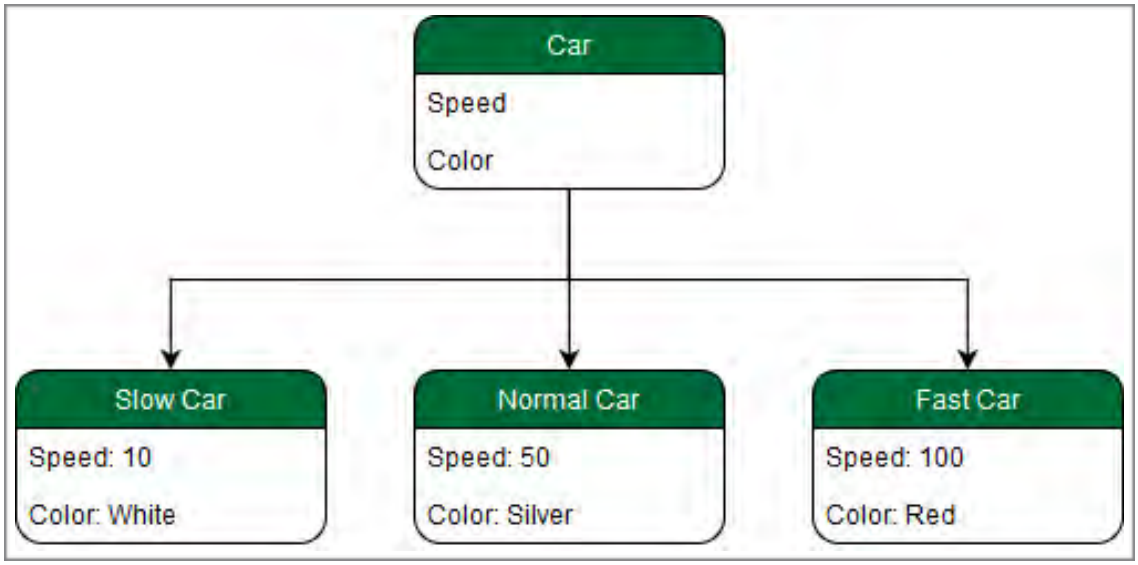
新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。
在这一课的内容中，我们将一起来创建射击用的武器Gun。
在之前的学习中，我们已经知道了当创建一个Blueprint Class的时候，可以选择一个父类。实际上，除了UE4默认提供的类，我们还可以选择自己的蓝图类作为父类。通过这种方式，就可以让不同类型的对象具有相同的功能或属性。

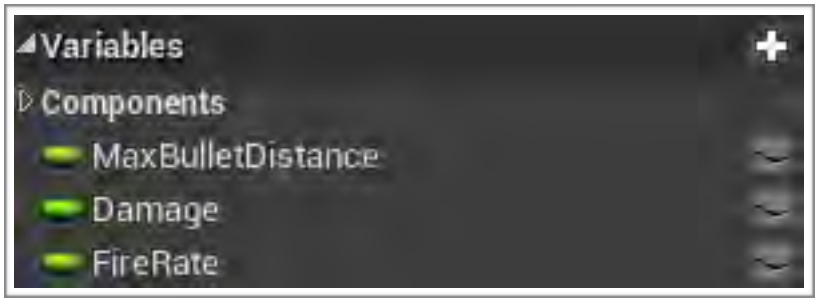
比如我们希望拥有多种类型的汽车，那么可以先创建一个基本的汽车类，其中包含一些常用的变量，比如速度和颜色。然后我们就可以使用这个基本的汽车类作为父类，来创建更多的具备特定功能的子类。每个子类都会包含父类的变量。通过这种方式，就可以轻松创建有不同速度和颜色的汽车。



我们也可以用类似的方式来创建枪支武器的基类。

创建基本的Gun类

返回主编辑器，创建一个Actor类型的蓝图类，将其命名为BP_BaseGun，然后将其打开。
接下来我们将创建几个变量来定义枪支的属性，创建以下float类型的变量：
. MaxBulletDistance: 代表每个子弹可以飞行的最大距离
. Damage: 代表当子弹击中角色时可以造成的伤害值
. FireRate: 代表枪支可以发射下一课子弹的时间间隔

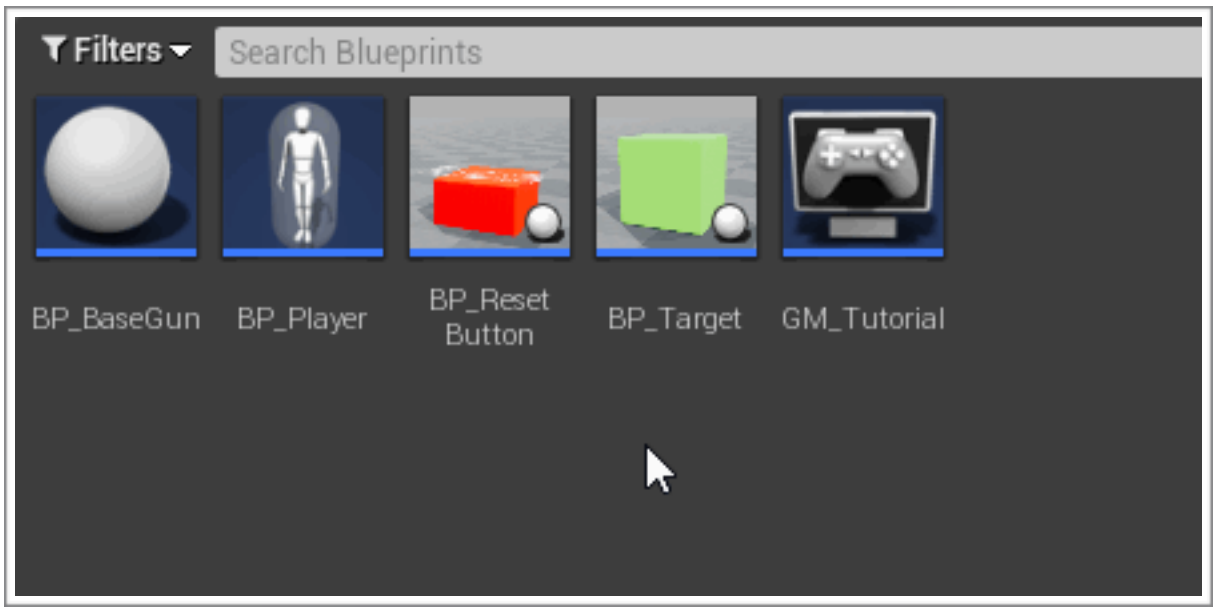


现在我们需要枪支的物理呈现形式。为此，需要添加一个Static Mesh组件，并将其命名为GunMesh。

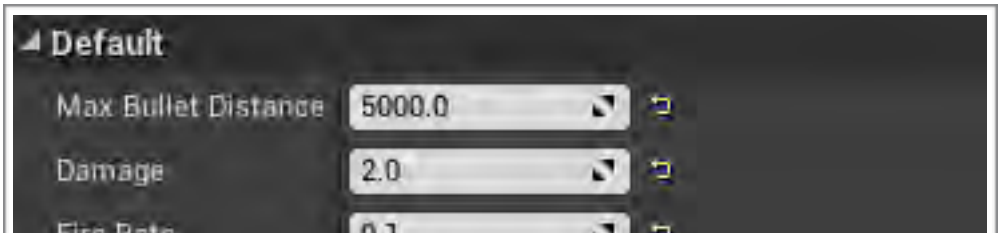
现在先别操心选择static mesh纹理贴图的事情，我们将在创建枪支子类的部分来完成这个工作。

创建枪支的子类

点击Compile并返回主编辑器。右键单击BP_BaseGun，然后选择Create Child Blueprint Class。

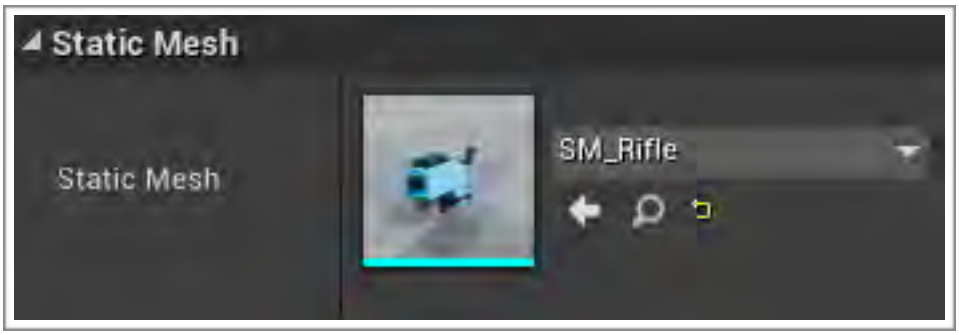


将其命名为BP_Rifle，然后将其打开。打开Class Defaults，然后把其中的变量设置成以下数值：
MaxBulletDistance: 5000
Damage: 2
FireRate: 0.1



以上数值的含义是：每颗子弹的最大飞行距离是5000。当它命中actor角色时，会造成2点的伤害值。当持续进行射击时，两次射击之间的间距是0.1秒。

接下来我们需要指定枪支的纹理。选择GunMesh组件，然后将Static Mesh设置成SM_Rifle。

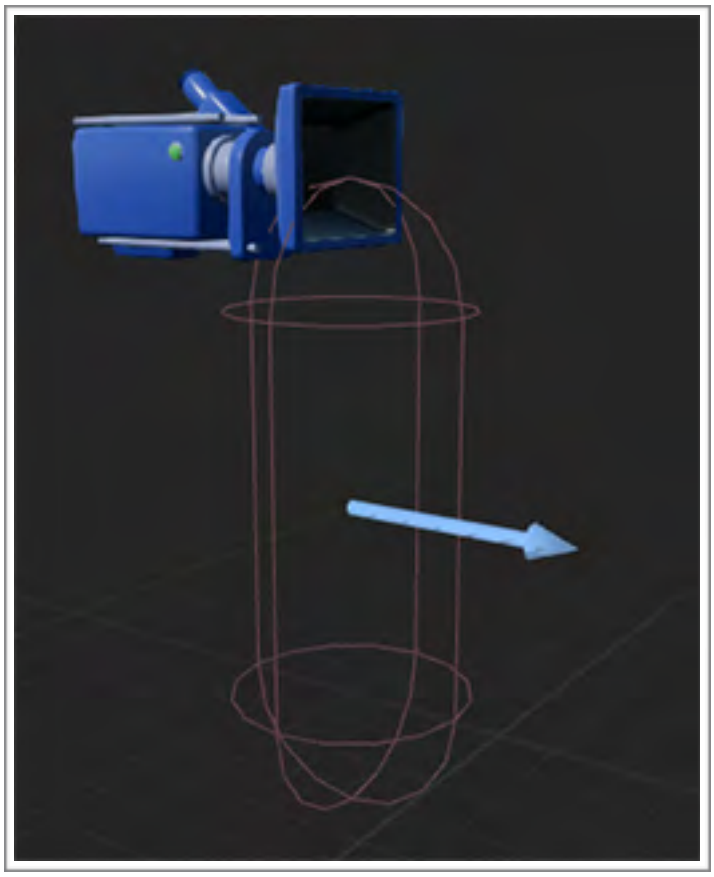


现在枪支的设置就完成了。点击工具栏上的Compile按钮，然后关闭BP_Rifle。
接下来我们需要创建自己的camera组件，这样可以让我们更好的控制摄像机。同时，我们还可以将枪支关联到摄像机上，从而让枪支始终显示在摄像机的前面。

创建摄像机

打开BP_Player，然后创建一个Camera组件，将其命名为FpsCamera。

摄像机的默认位置有点太低，可能会让玩家感觉很小。为此，我们需要将FpsCamera的location属性设置为（0，0，90）。



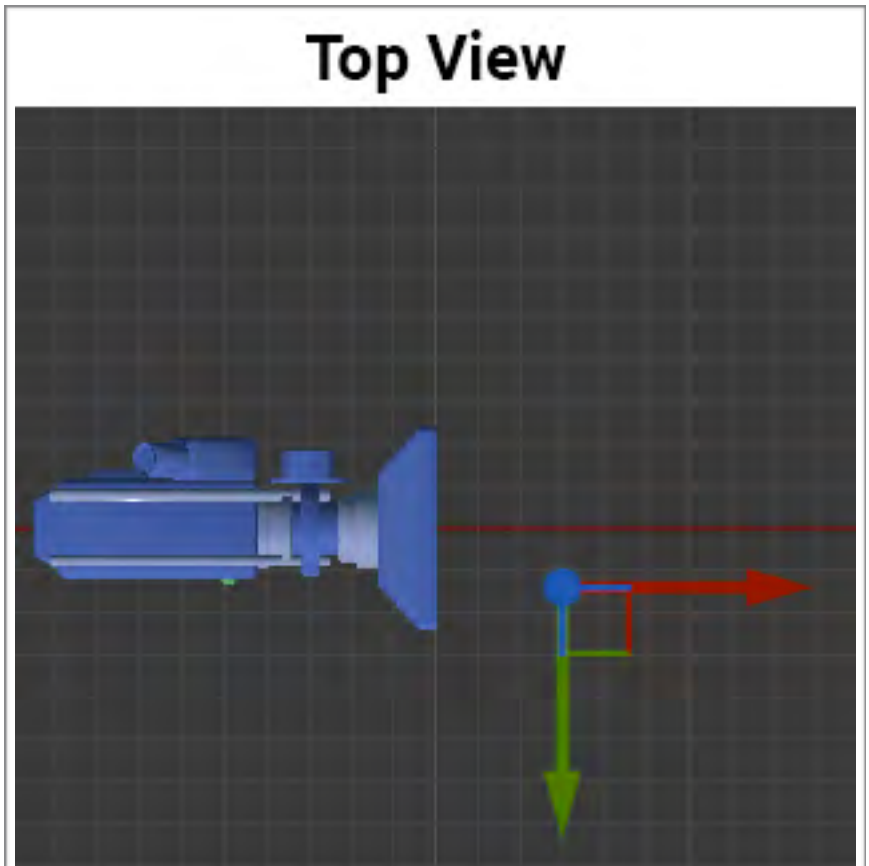
默认情况下，Camera组件并不使用controller的rotation信息。为此，我们需要在Details面板中启用Camera Options\Use Pawn Control Rotation。

接下来我们需要定义枪支的位置。

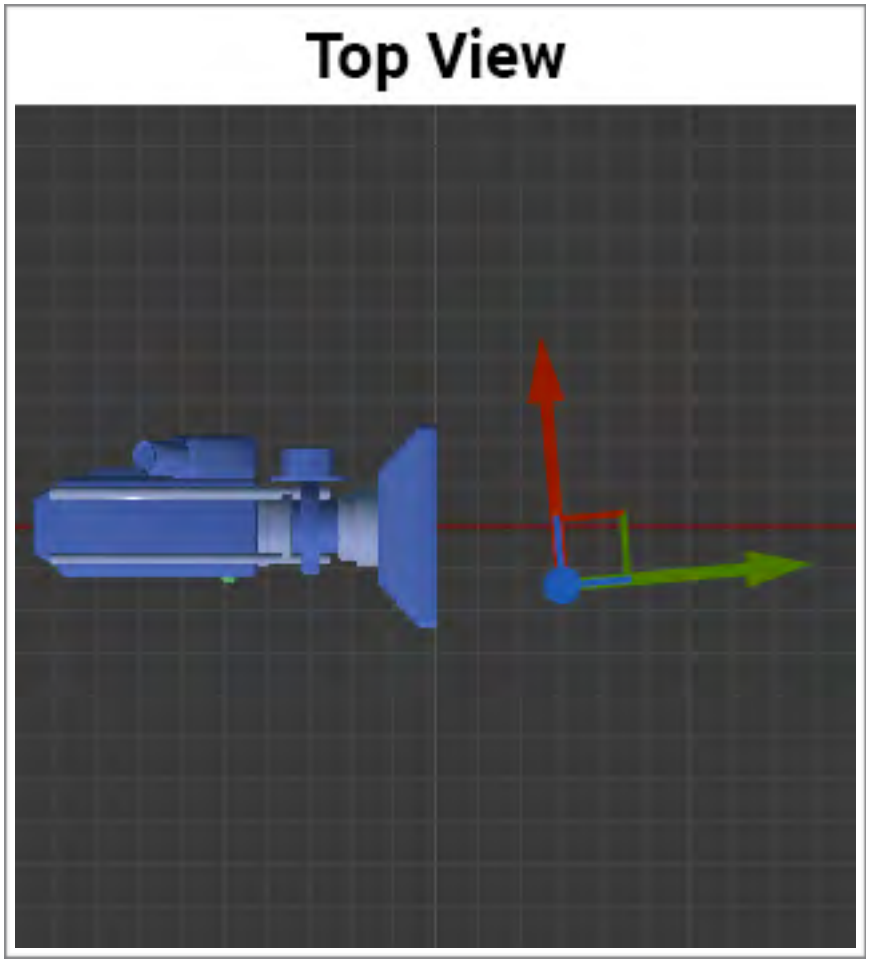
定义枪支的位置

为了创建枪支的位置，我们可以使用Scene组件，因为它只有一个Transform信息。确保选中FpsCamera，然后创建一个Scene组件。该组件将会关联到camera上，将其命名为GunLocation。

通过将GunLocation关联到FpsCamera上，枪支就会保持和摄像机的相对静止，这样我们就可以让枪支始终显示在摄像机的前方。
接下来将GunLocation的Location设置为（30，14，-12），这样就会让枪支显示在摄像机前方的一侧。



接下来将rotation设置为（0，0，-95），这样可以让它看起来瞄向屏幕的正中央。



好了，本课的内容就先到这里了。
在下一课的内容中，我们将生成枪支对象，并将其关联到GunLocation。
我们下一课再见~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

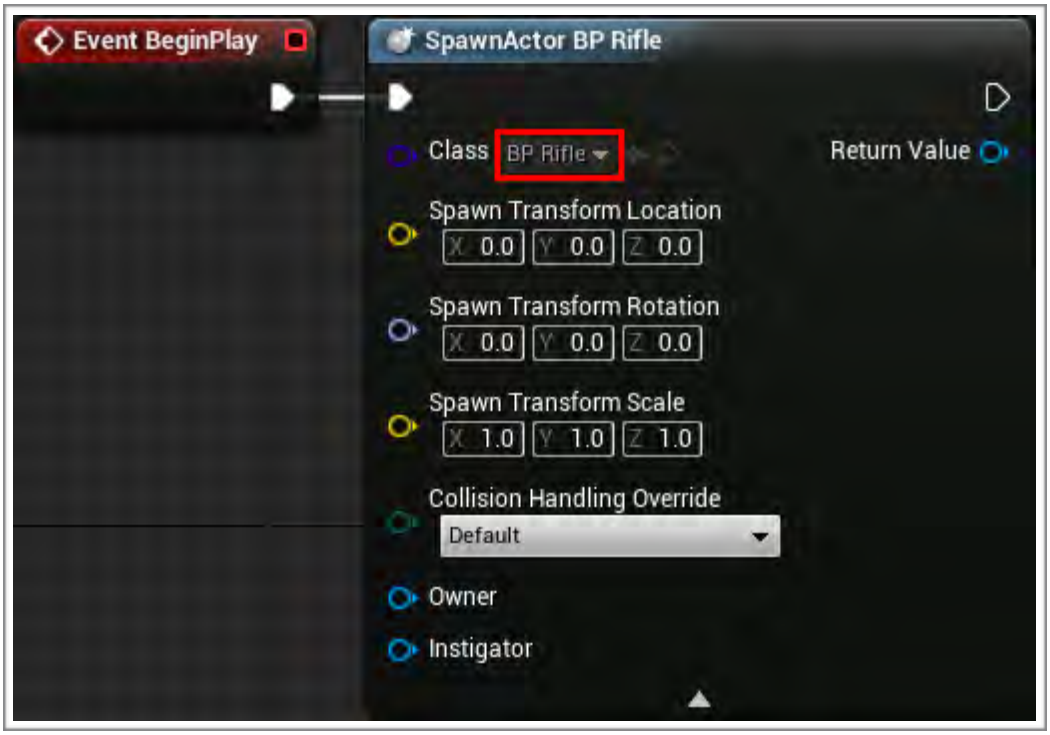
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

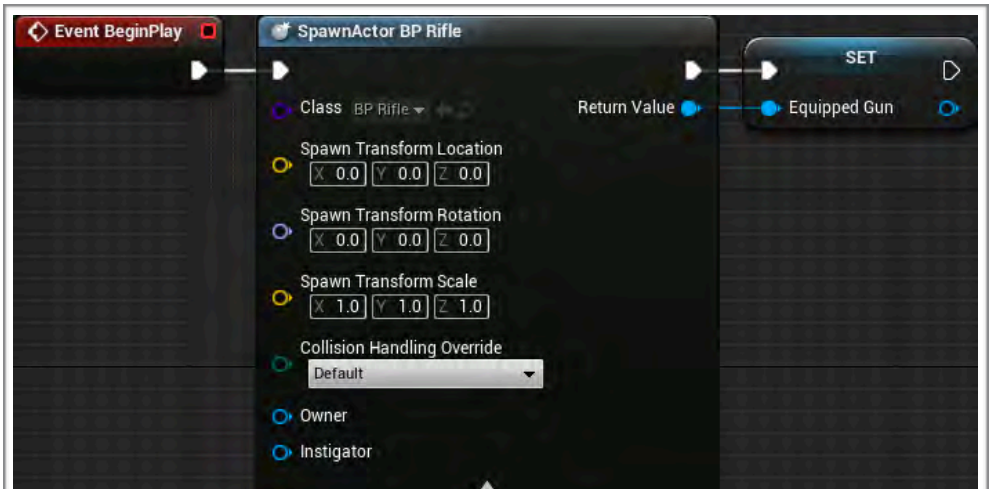
欢迎继续我们的学习。
在这一课的内容中，我们需要把枪支关联到角色上。

生成并关联枪支

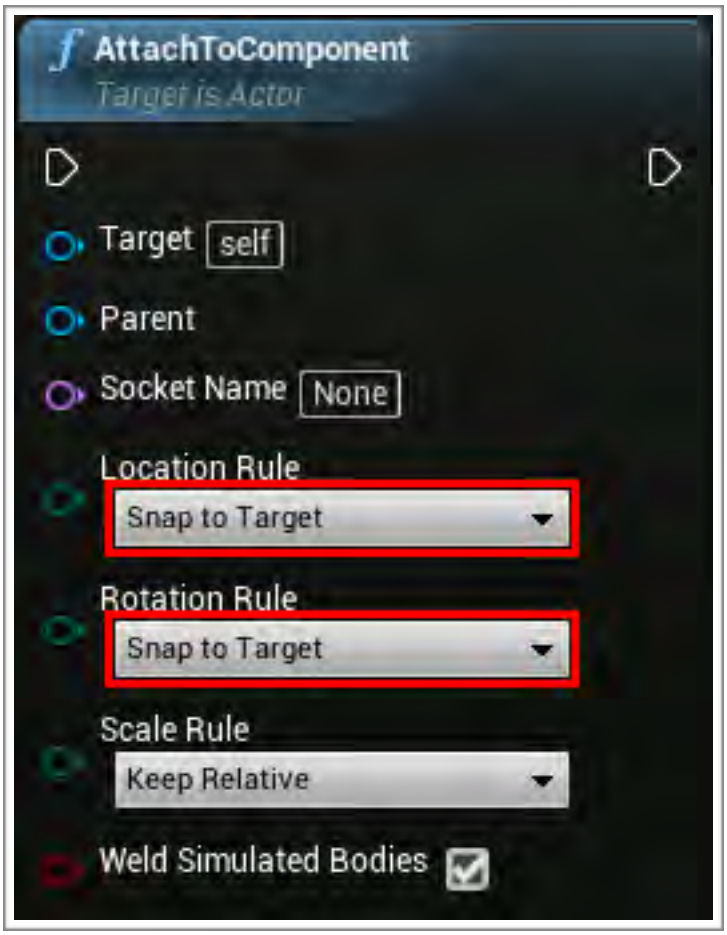
打开BP_Player，找到或创建Event BeginPlay节点，然后创建一个Spawn Actor From Class节点。将Class设置为BP_Rifle。



考虑到我们后面还需要用到枪支，因此需要将其保存在一个变量中。创建一个类型为BP_BaseGun的变量，并将其命名为EquippedGun。
需要注意的是，变量的类型不是BP_Rifle。这是因为玩家可能会使用不同类型的枪支，而来复枪只是其中的一种。当我们生成其它类型的枪支时，将无法保存到BP_Rifle类型的变量中。
接下来将Equippedgun设置为Spawn Actor From Class的Return Value。



为了关联枪支，我们可以使用AttchToComponent。创建一个该节点，然后将其中的Location Rule和Rotation Rule设置为Snap to Target。这样枪支就会和其关联的父类拥有相同的位置和旋转信息。



接下来创建一个到GunLocation的引用，并使用以下方式连线：

小结一下：

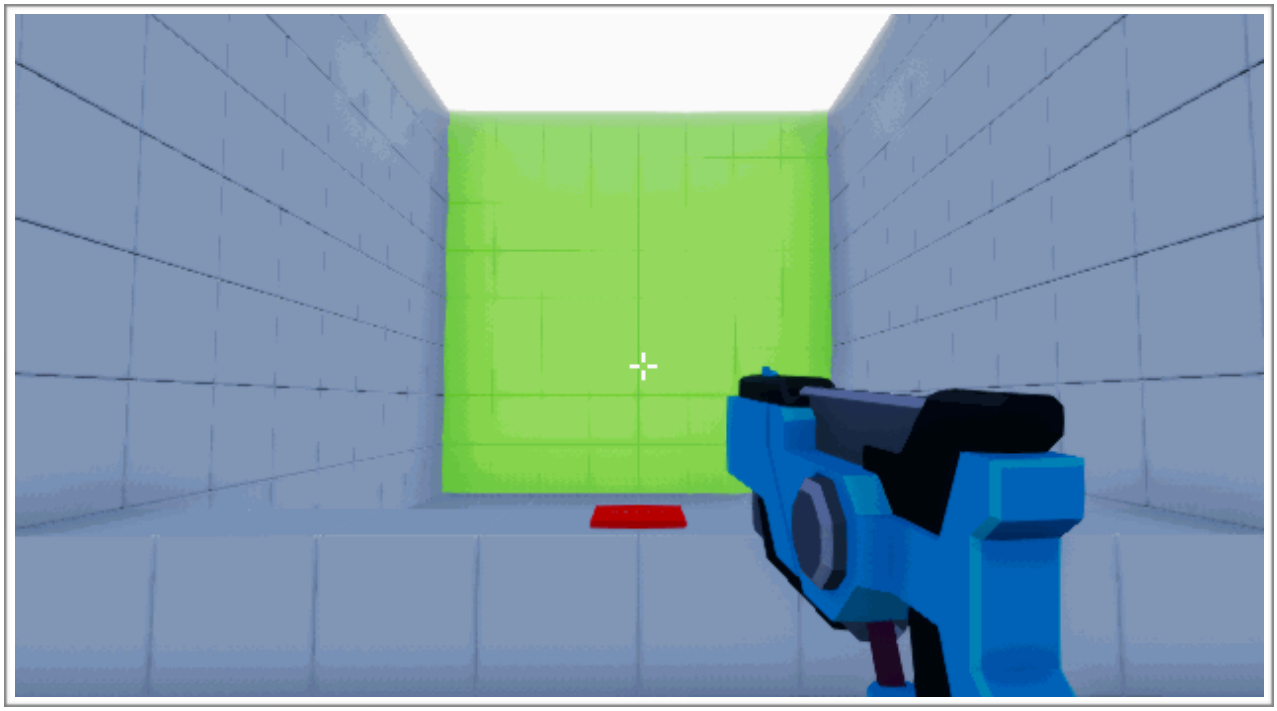
1.当生成BP_Player时，会生成一个BP_Rifle的实例对象。

2.EquippedGun将会保留一个到所生成的BP_Rifle的引用

3.AttachToComponent会把枪支关联到GunLocation上。

点击工具栏上的Compile按钮。返回主编辑器，并按下Play。

现在进入游戏后你有了一把枪，当四处查看的时候，枪始终会出现在摄像机的正前方。



好了，本课的内容就到这里了~

我们下一课再见。

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

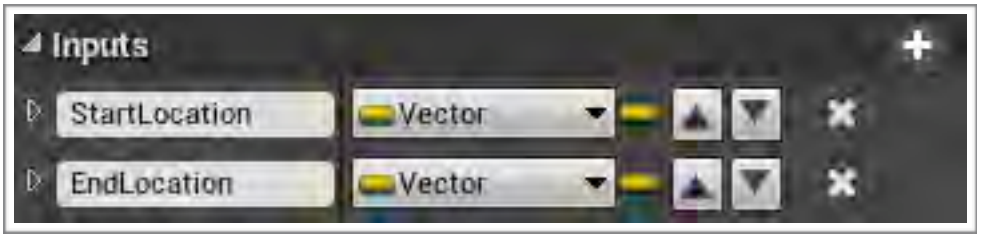
Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

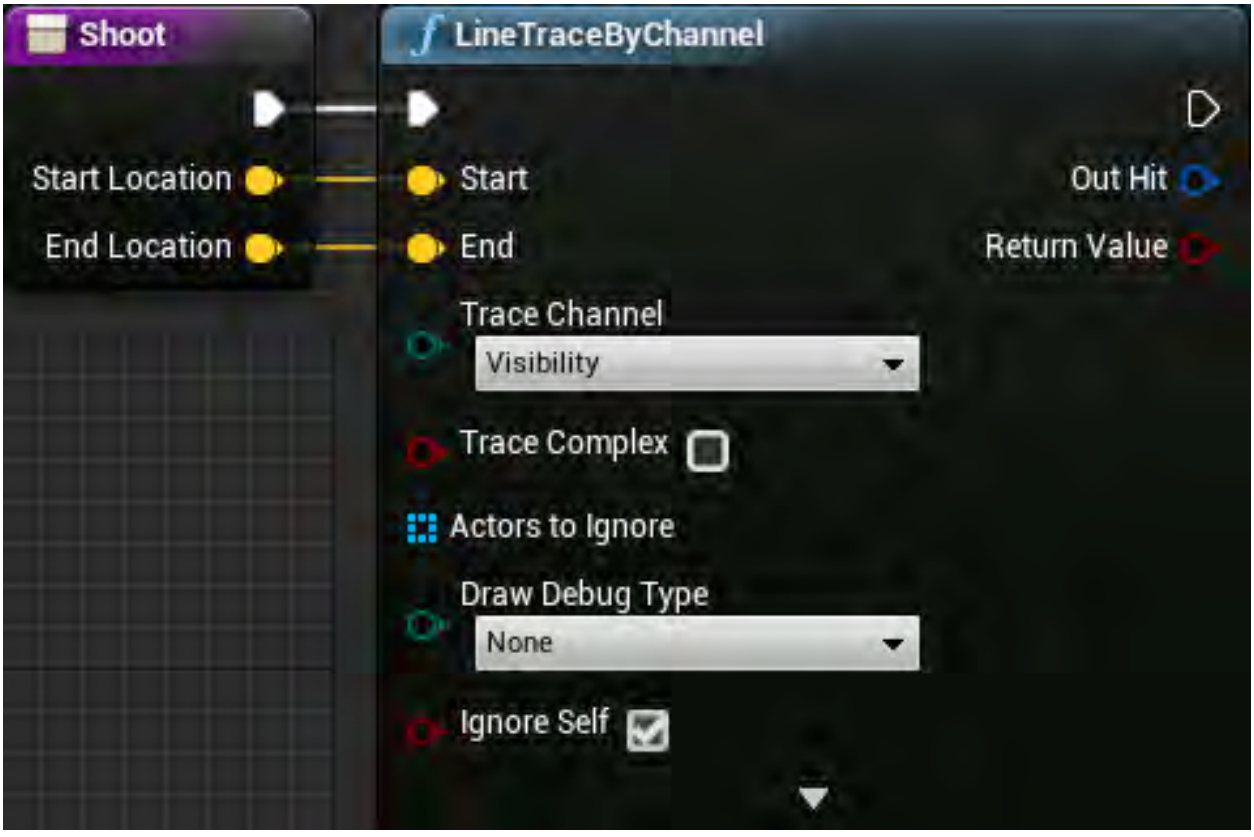
欢迎继续我们的学习。
在这一课的内容中，我们将学习如何发射子弹。而为了检测子弹是否命名目标，我们需要用到line trace。

发射子弹

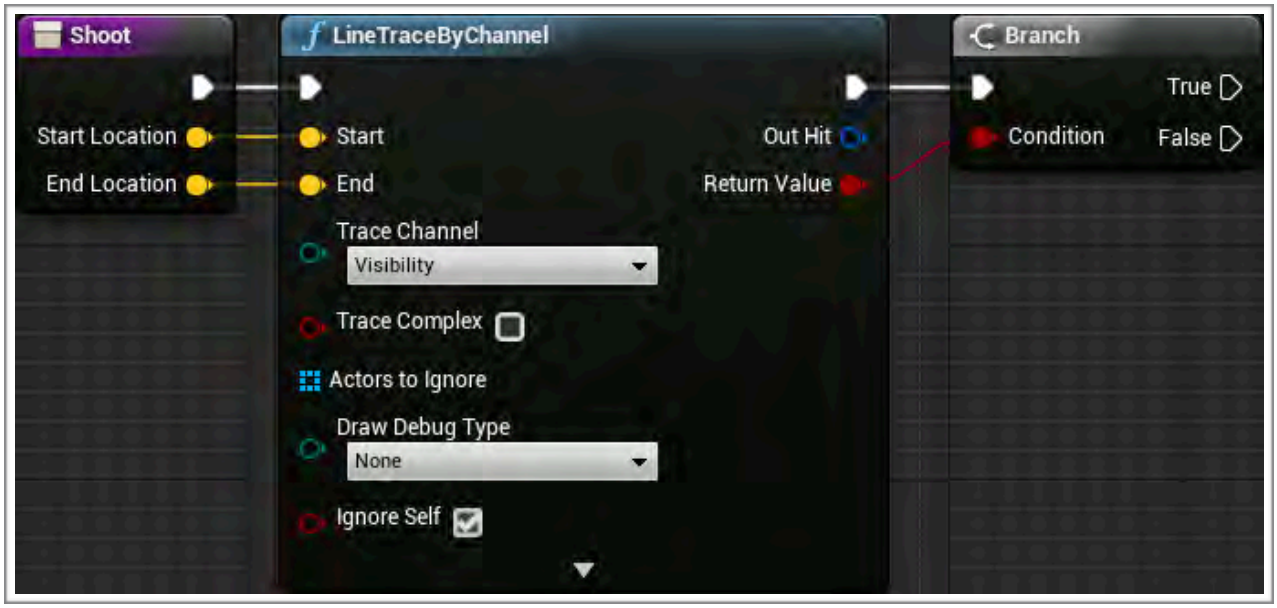
line trace（射线跟踪）将接收起点和终点（形成一条直线）的位置作为参数，然后沿着直线检查每个点，直到其命中某个东西。在游戏中，这是检测子弹是否命名目标的最简单方式。
因为射击时枪支的功能，因此需要在枪支class中实现，而非玩家类中实现。打开BP_BaseGun，然后创建一个名为Shoot的函数。
接着创建两个Vector类型的输入参数，分别命名为StartLocation和EndLocation。这两个参数将是射线跟踪函数的起点和终点（将从BP_Player中传入）。



接下来我们可以使用LineTraceByChannel来执行射线跟踪。该节点将使用Visibility或Camera的碰撞通道检查是否命名。创建该节点，并使用以下的连线方式：



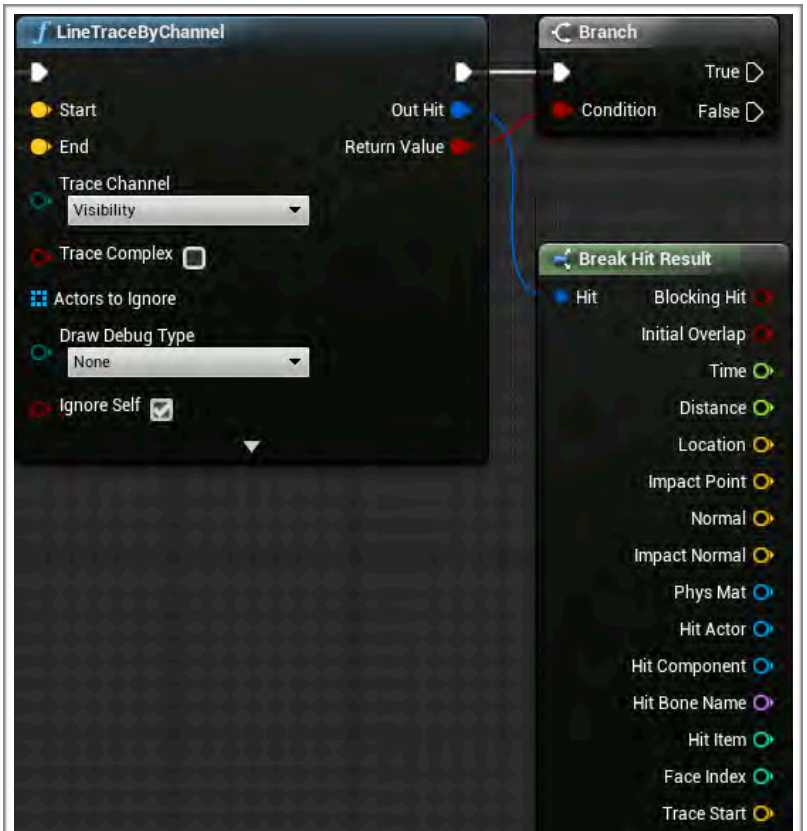
接下来，我们需要检查射线追踪是否命名了东西。创建一个Brach节点，并使用以下连线：



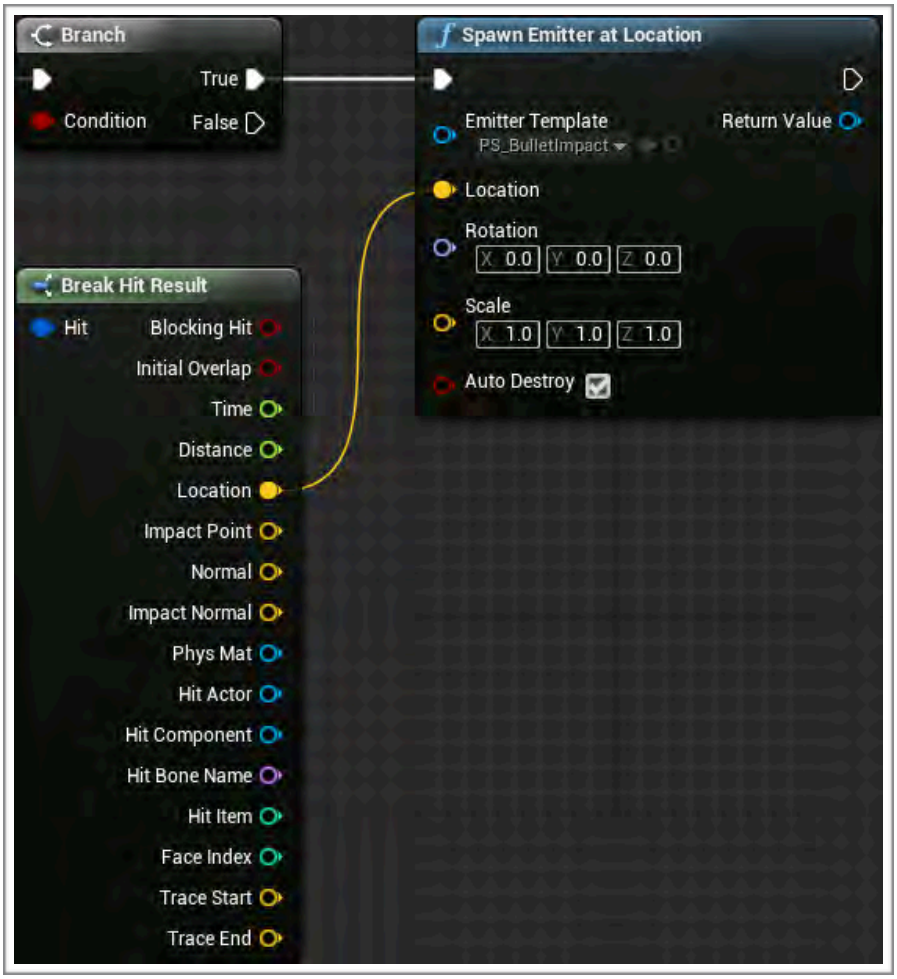
当检测到碰撞时，Return Value将输出true，反之则输出false。
为了让玩家获得子弹命中的视觉反馈，我们还需要用到粒子特效。

生成子弹碰撞的粒子特效

首先，我们需要获取射线追踪的位置。从Out Hit拉出一条线，然后在视图中选择Break Hit Result。



创建一个Spawn Emitter at Lcoation节点，然后将Emitter Template设置为PS_BulletImpact。随后将其Location连接到Break Hit Result的Location端口。



现在整个视图中的节点连接如下：

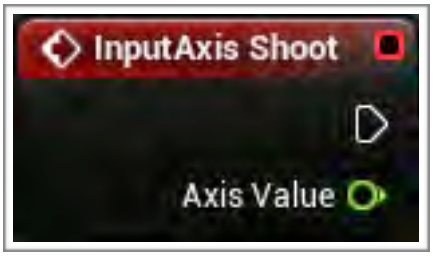
小结一下：
1.当Shoot执行的时候，会使用所提供的起点和终点生成一个射线追踪。
2.如果检测到碰撞，Spawn Emitter at Lcoation节点将在碰撞位置生成PS_BulletImpace粒子特效。

现在射击机制已经完成，接下来我们将实际使用。

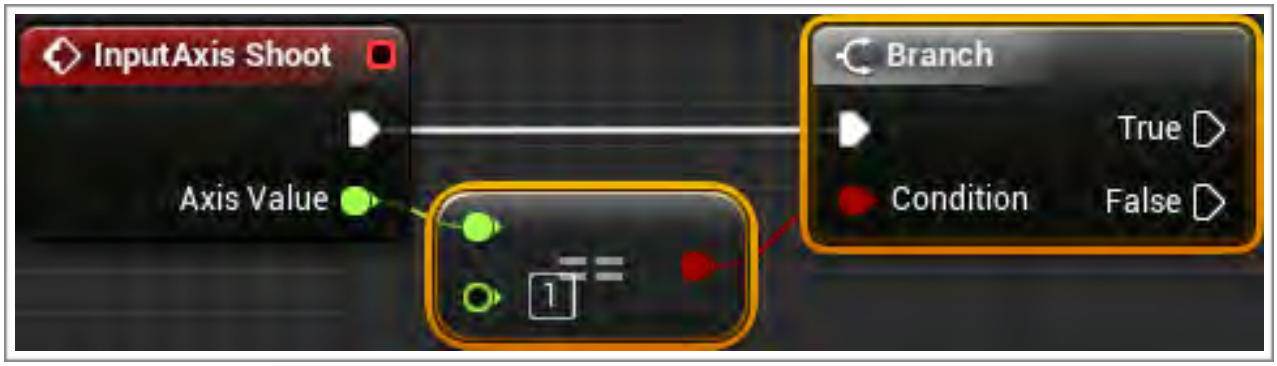
调用Shoot函数

首先我们需要创建设计的键盘映射。点击工具栏上的Compile按钮，然后打开Project Settings。创建一个新的Axis Mapping，并将其命名为Shoot。将所映射的键更改为Left Mouse Button，然后关闭Project Settings。

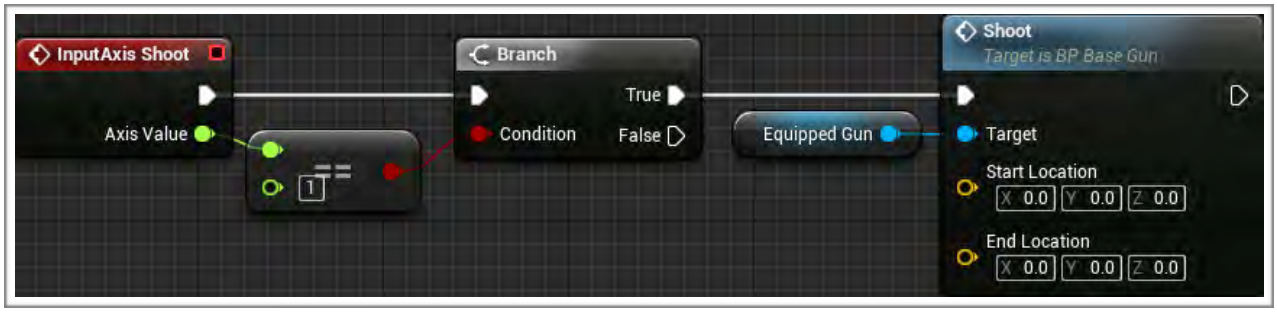
接下来打开BP_Player,然后创建一个Shoot 事件。



为了检查玩家是否按下Shoot键，我们需要检查Axis Value是否等于1，为此创建如下的节点：



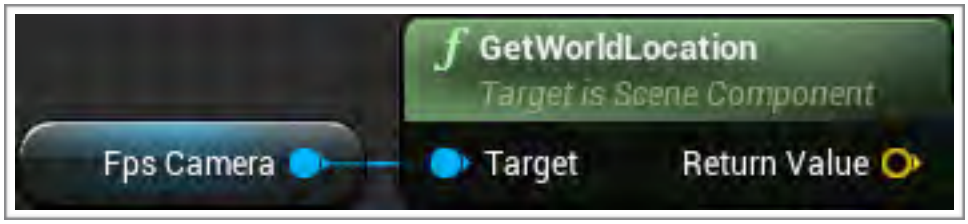
接下来，创建一个到EquippedGun的引用，然后调用Shoot函数。



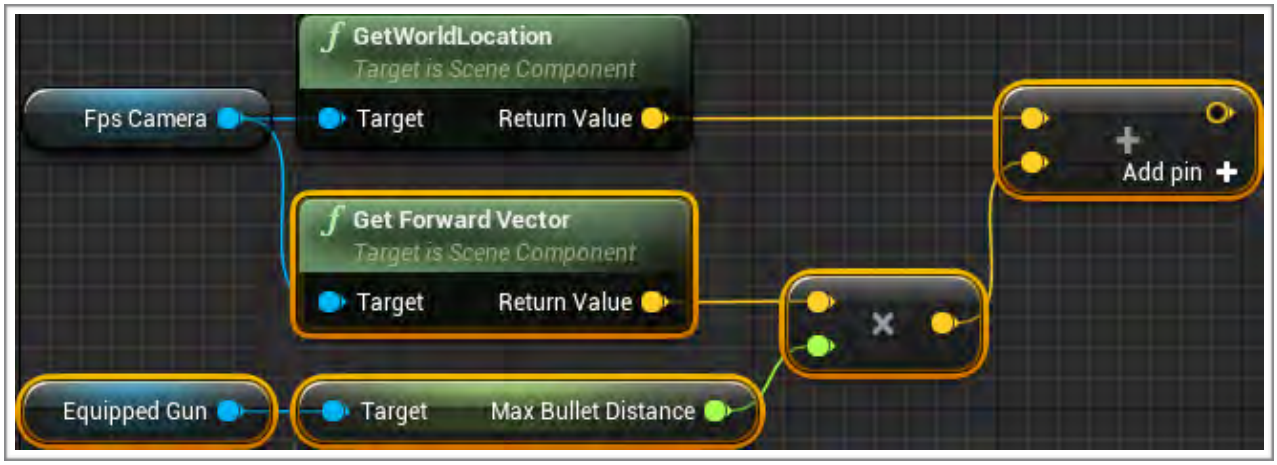
接下来我们需要计算射线追踪的起点和终点位置。

计算Line trace的位置

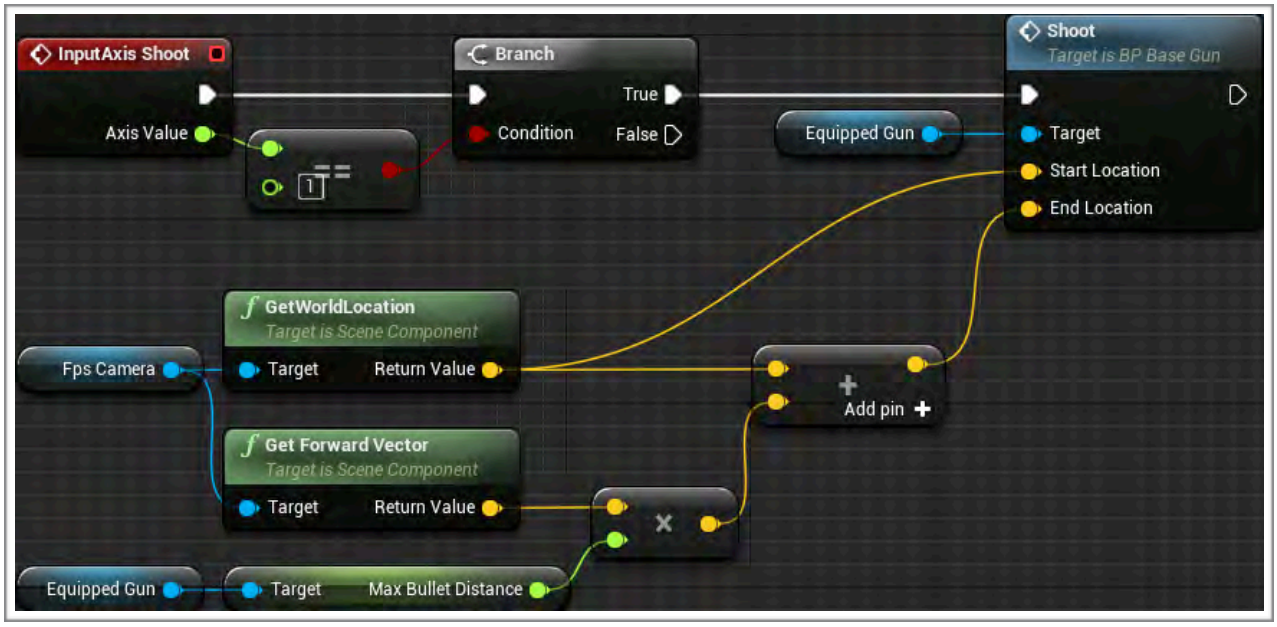
在很多FPS游戏中，子弹从摄像机的位置发射，而不是从枪支的位置。这是因为摄像机通常和十字准星的位置完美对齐。因此当子弹从摄像机的位置发射时，就可以保证子弹飞往准星所瞄准的位置。
创建一个到FpsCamera的引用，然后将其连接到GetWorldLocation。



接下来我们需要确认终点的位置。记住枪支有MaxBulletDistance这个变量，这就意味着重点应该是摄像机前的MaxBulletDistance。为此，创建以下节点：



接下来按照以下方式连接各个节点：



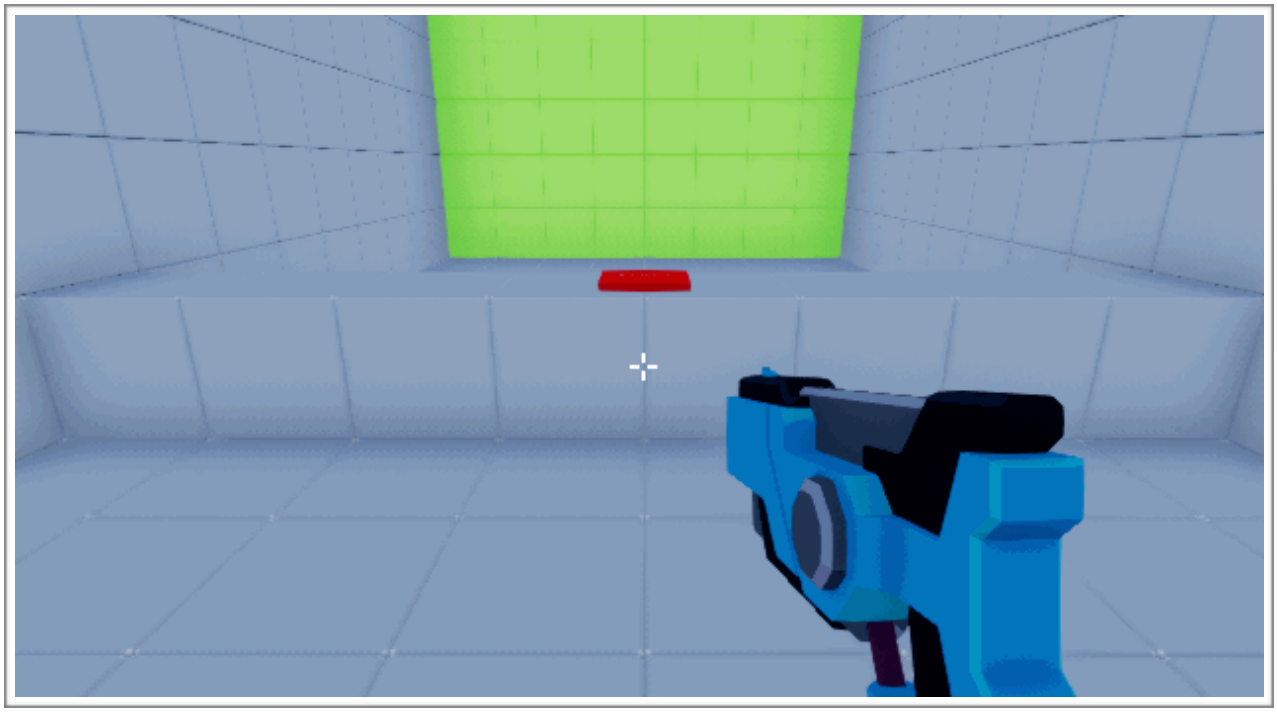
小结：

1.当玩家按下鼠标左键时，枪支会从摄像机的位置发射子弹。

2.子弹将飞行MaxBulletDistance的距离

点击工具栏上的Compile按钮。

然后在主编辑器中点击Play按钮测试游戏，按下鼠标左键开始射击~



现在枪支每一秒都在发射，这个似乎有点太快了，所以接下来我们需要降低开火的速率。
好了，本课的内容就先到这里，我们下一课再见~

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/

欢迎继续我们的学习。

在上一课的内容中，我们已经可以让枪支发射子弹了，但是发射速度似乎有点太快。因此，接下来我们需要降低枪支发射子弹的速率。

减慢开火速度

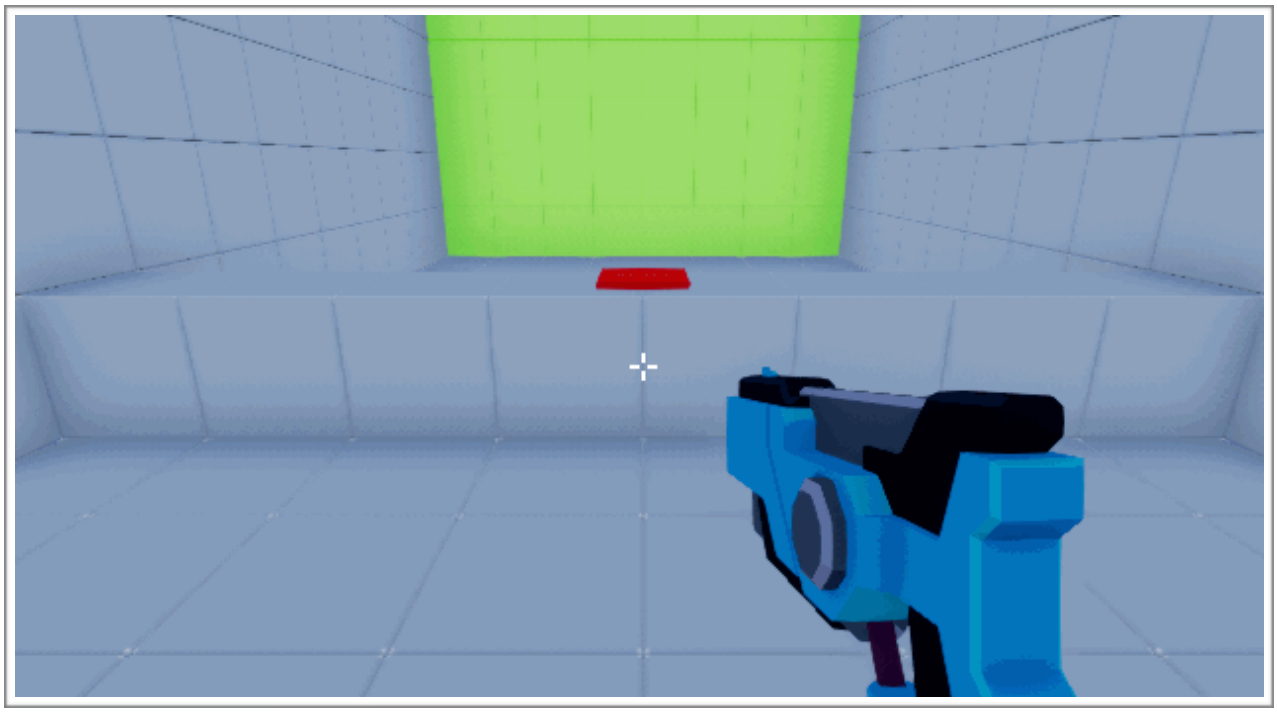
首先，我们需要需要一个变量来判断玩家是否可以射击。打开BP_Player，创建一个boolean类型的变量，并将其命名为CanShoot。将其默认值设置为true。如果CanShoot的值是true,那么玩家角色可以射击，反之则不行。
更改Branch部分的连线如下：

好了，现在玩家只有当Shoot键被按下，而且CanShoot的值为true时才能射击。

接下来按照以下方式添加高亮的节点：

简单说明一下这里的变化：
1.玩家只有在按下鼠标左键，且CanShoot的值为true时才能射击。
2.一旦玩家发射出一枚子弹，CanShoot将被设置为false。这样就会防止玩家再次射击。

3.在FireRate生成的等待期结束后，CanShoot将被设置回true。
点击Compile按钮，然后关闭BP_Player。点击Play按钮预览游戏效果。

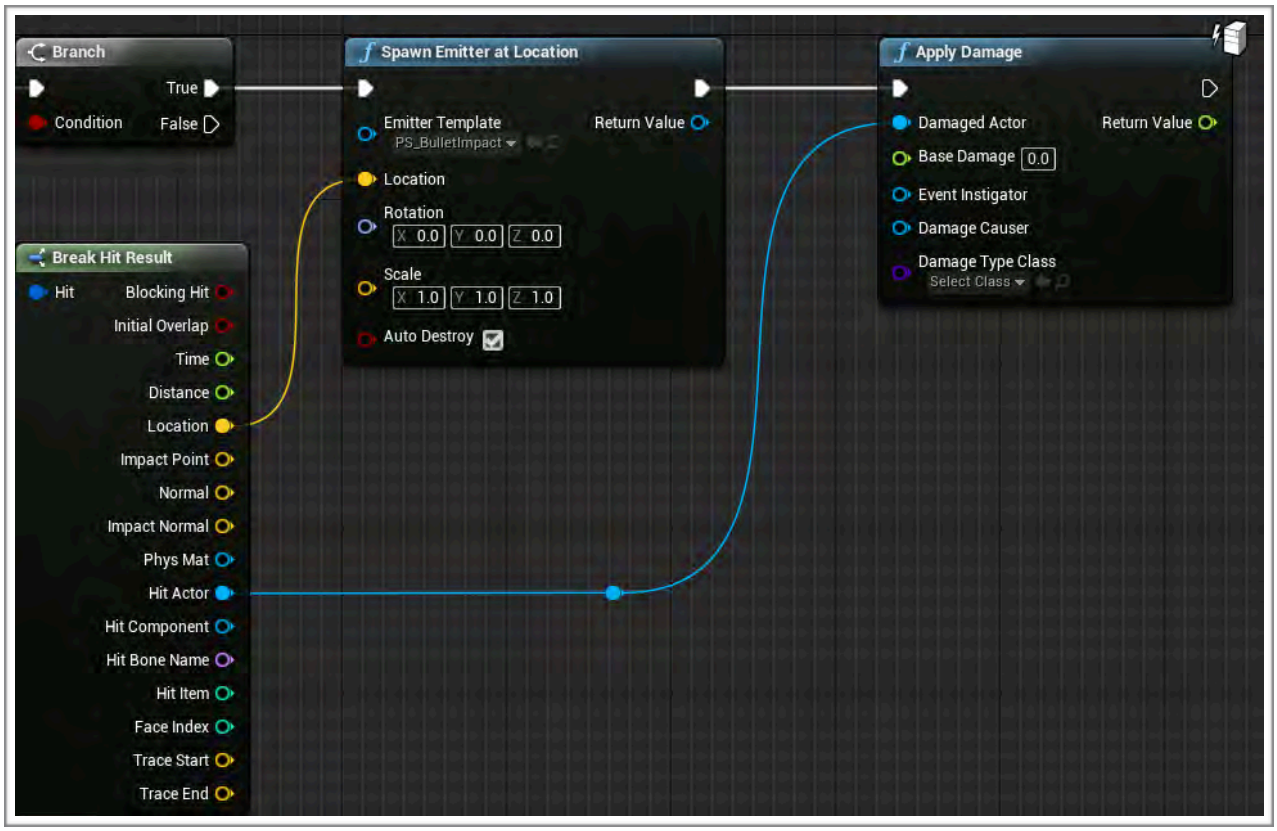


接下来我们还需要让受到攻击的目标对子弹产生反应，不然不是太无趣了吗~

在虚幻4中，每个角色都可以受到伤害，但是开发者可以决定角色如何对伤害作出响应。
例如，当受到伤害时，一个战斗中的游戏角色生命值会降低。但如果是气球，可能就直接爆了。
因此，当🎈受到伤害时，我们需要通过程序让它直接爆掉。

不过在我们处理角色所受到的伤害前，首先要做的就是应用伤害。打开BP_BaseGun，然后在Shoot函数的最后添加一个Apply Damage节点。

接下来，我们需要指定想要伤害的角色对象。在这里，具体来说就是射线追踪所碰到的角色。将Damage Actor连接到Break Hit Result的Hit Actor上。



最后，我们需要指定要应用的伤害值。获取一个到Damage的引用，然后将其连接到Base Damage上。

现在，当我们调用Shoot的时候，它就会对射线追踪所命中的角色造成伤害。点击Compile，然后关闭BP_BaseGun。

接下来我们需要对角色造成的伤害进行处理。

处理角色所受到的伤害

首先，我们需要处理目标对象如何受伤。打开BP_Target，然后创建一个Event AnyDamage节点。每当角色受到的伤害不是0的时候，都会执行该事件。

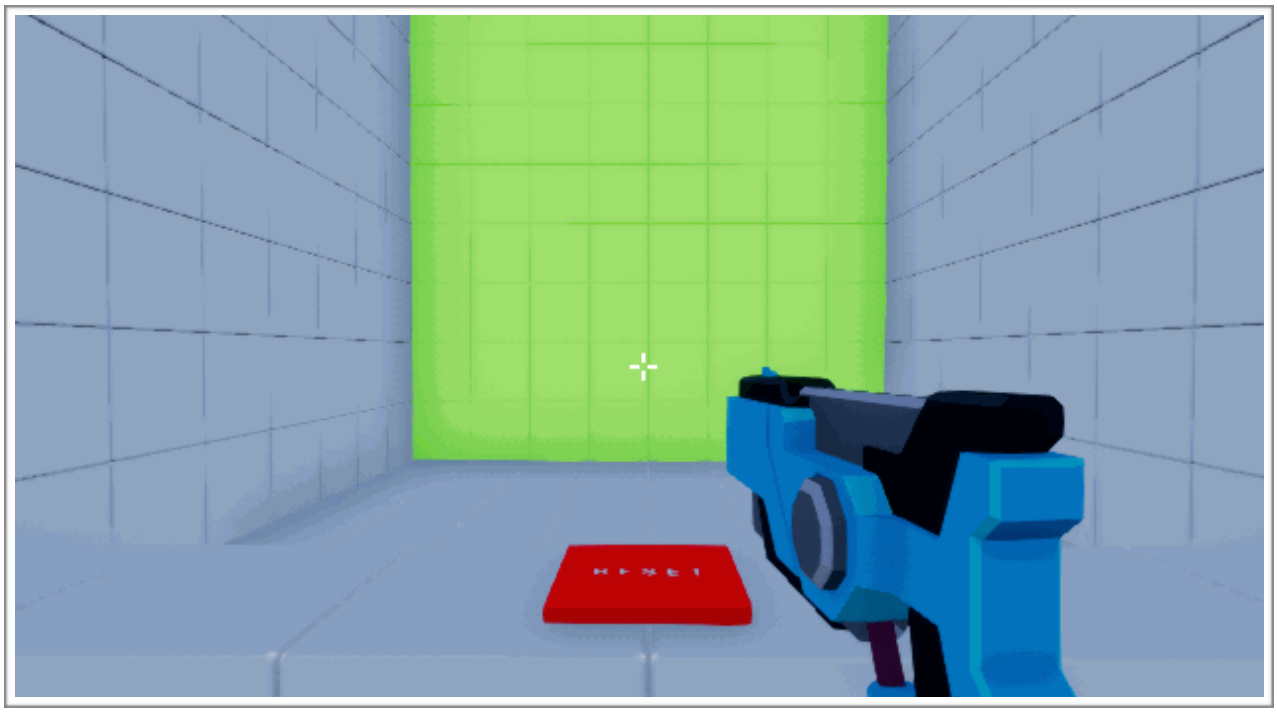
接下来调用TakeDamage函数，并连接Damage端口。这样就可以将角色的生命值从Health变量中抽取出来，并更新目标角色的颜色。

现在，当目标受到伤害的时候，生命值会降低。点击Compile按钮，然后关闭BP_Target。

接下来我们需要处理按钮如何接收伤害。打开BP_ResetButton，然后创建一个Event AnyDamage。然后调用ResetTargets函数。

这样，当按钮受到伤害的时候，就会重置所有目标。点击Compile，然后关闭BP_ResetButton。

回到主编辑器，点击Play按钮，然后开始射击目标。如果想要重置目标，只需要射击按钮即可。



好了，本系列的课程内容就到此结束了。
完整的项目请参考这里：
链接:https://pan.baidu.com/s/1qrglUur11QaKsz4T2ACOKg 密码:wkft

讨论群-笨猫学编程QQ群：
375143733

答疑论坛：
http://www.vr910.com/forum.php?mod=forumdisplay&fid=52

知乎专栏：

https://zhuanlan.zhihu.com/kidscoding

新浪博客:
http://blog.sina.com.cn/eseedo

Github:
https://github.com/eseedo

个人网站：
http://icode.ai/