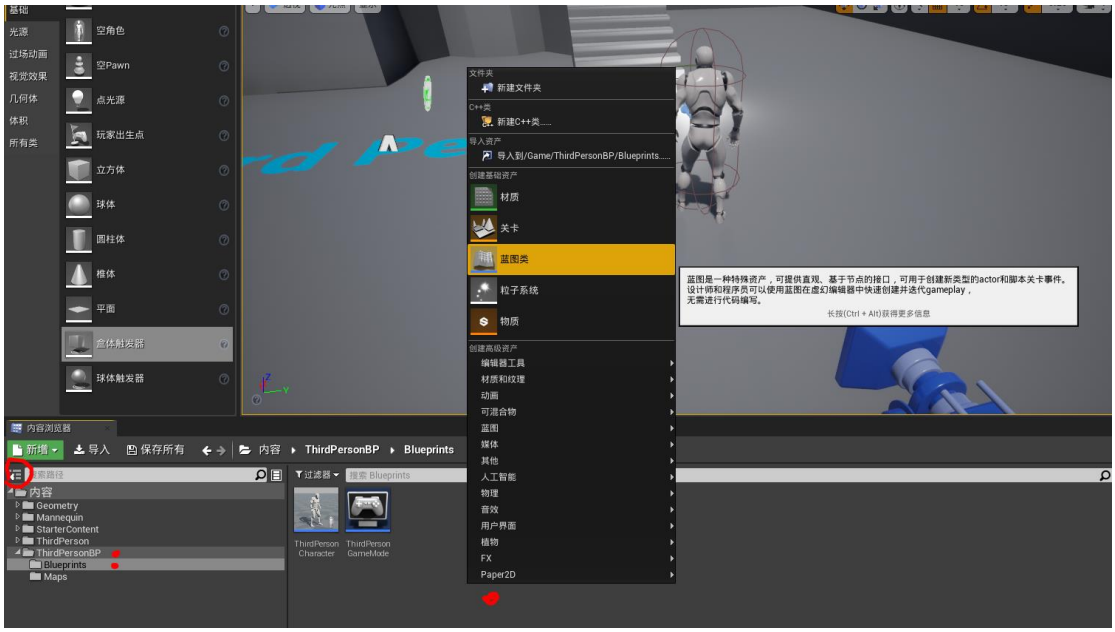
第 3 课 蓝图节点

2、事件（Event）类型节点

（3）OnComponentBeginOverlap 节点

与 OnActorBeginOverlap 节点不同，OnComponentBeginOverlap 节点用于检测组件（Component）的碰撞重叠事件。本节中，仍然通过实现接触灯光而改变灯光状态，来理解 OnComponentBeginOverlap 事件节点的作用。

沿用之前的工程项目，先创建新的蓝图（Blueprint）。打开 Unreal Engine 编辑器，在内容浏览器 ThirdPersonBP 文件夹下的 Blueprints 子文件夹中的空白处鼠标右键，选择“创建基础资产”“蓝图类”。



在弹出的“选择父类”窗口中，为新创建的蓝图选择父类。所有类都可以作为一个蓝图的父类，也包含自定义的类。有几个常用的父类：

a. Actor 类：用于定义可放置或生成在关卡中的对象。

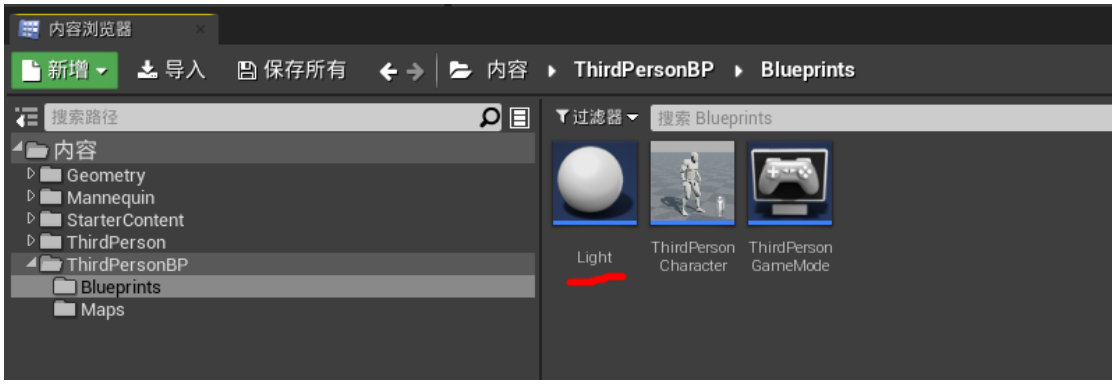
b. Pawn 类：Actor 类的子类，用于定义一个玩家或者电脑人工智能的游戏对象，是游戏中可控的 Actor 类。

c. 角色（Character）类：Pawn 类的子类，包含行走、跑步、跳跃以及更多动作。

d. 玩家控制器（Player Controller）类：Actor 类的子类，控制玩家使用的 Pawn。

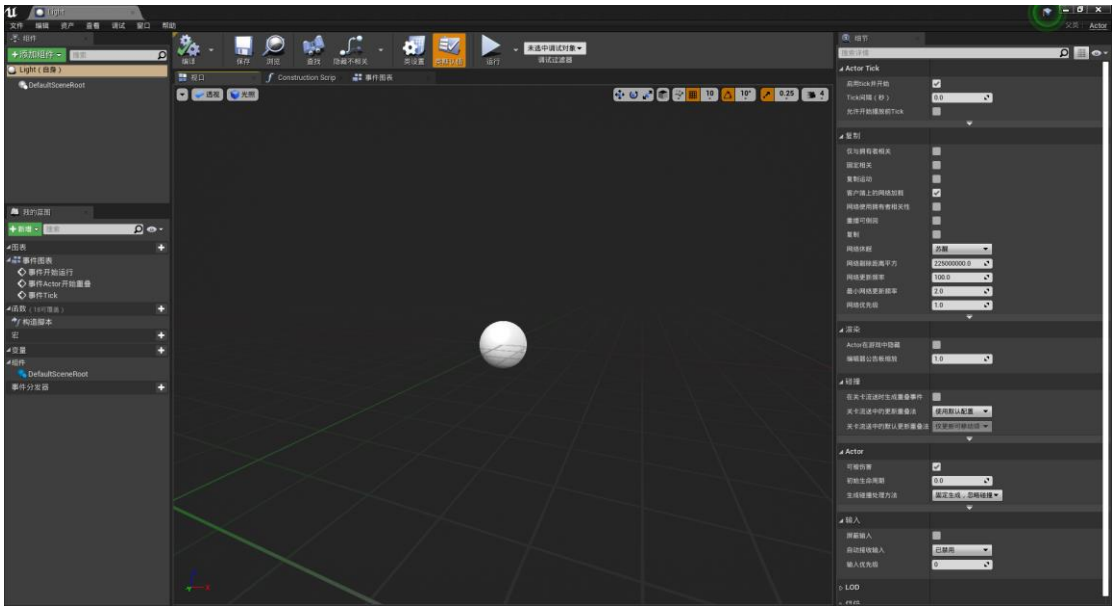
e. 游戏模式基础（Game Mode）类：定义游戏规则。

选择 Actor 类作为创建蓝图类的父类，并把新建的蓝图命名为 Light



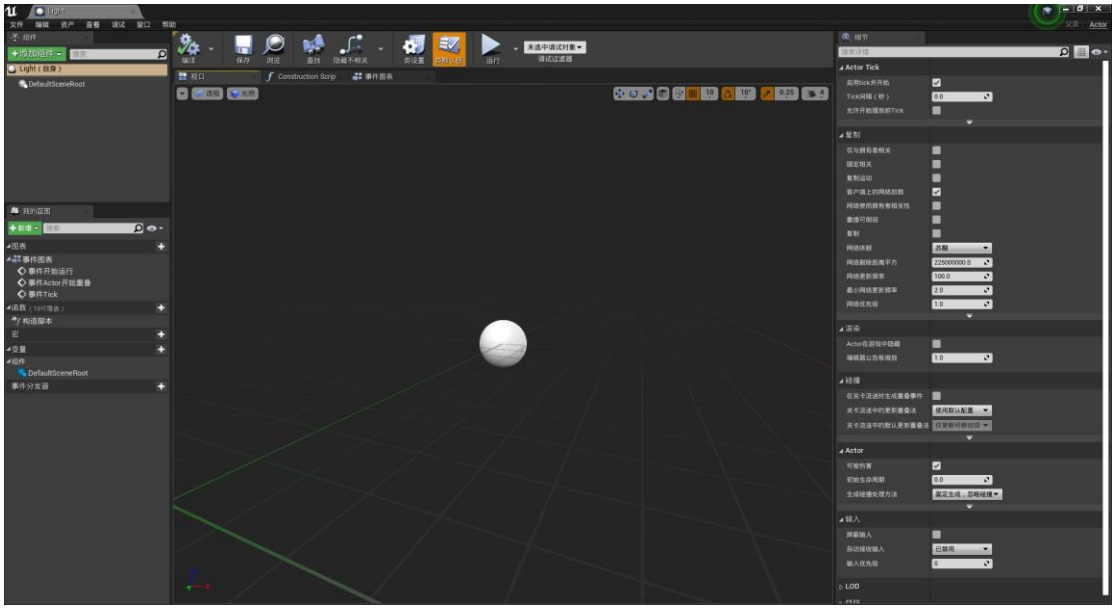
这里我们把蓝图建在“ThirdPersonBP>>Blueprints”是为了规范项目开发过程中文件存放的位置，所有蓝图类文件建议放置在 Blueprints 文件夹中。事实上，文件可以在任意文件夹中创建，都不影响游戏正常执行，但并不建议这么做。

双击创建的 Light 蓝图，以 Actor 为父类的蓝图编辑器界面如图。



编辑器面板有 3 个选项卡，分别是视口（Viewport）、构造脚本（Construction Script）、事件图表（Event Graph）。

在视口面板中可以查看、摆放和编辑创建的组件（Component）。



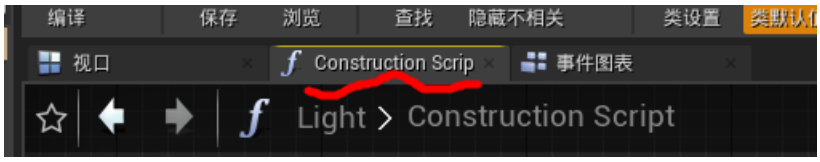
组件是一种特殊类型的对象，用作为 Actor 类中的一个子对象。组件一般用于需要简单更换部件的地方，以改变该组件所在 Actor 类的特定行为或功能。比

如，汽车、飞机、船的控制和运动有很大差别，然而这些交通工具也存在共性，这时可以把这些交通工具拆分成由多个组件共同组成，并通过保留共性组件，更换差异组件，迅捷地把一种交通工具设计改变为另一种交通工具，节省开发成本。

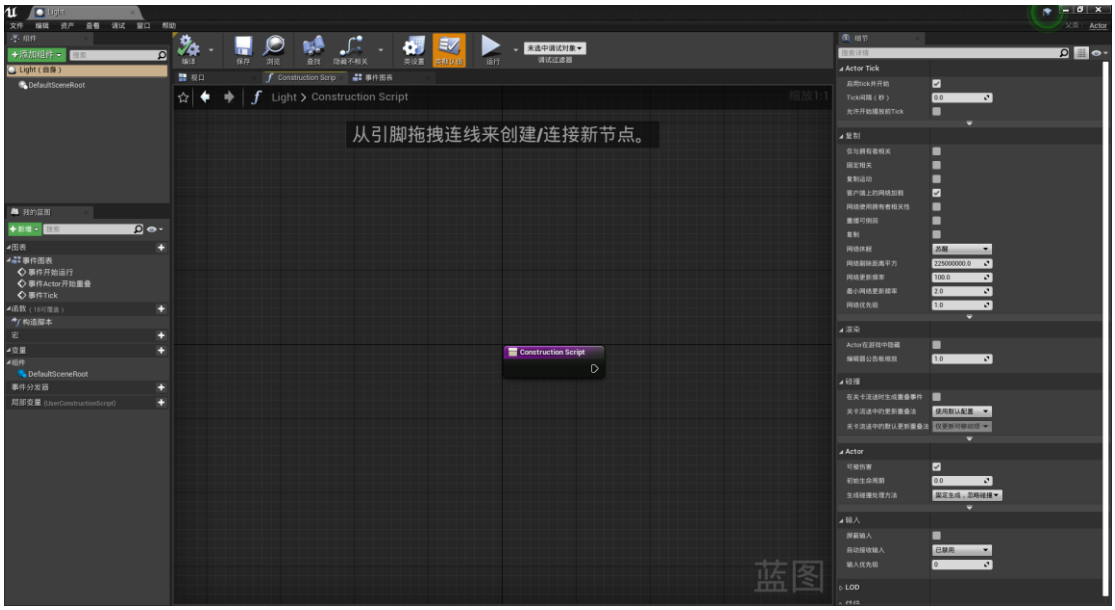
对于汽车而言，车轮、方向盘、车身、车灯和发动机都可以看作组件，汽车本身是 Actor。在 Actor 中添加组件，就好比是拼凑 Actor 中的各个构成部分。此时，即使不提供任何 Blueprint 脚本或者 C++代码，也可以将拼好的 Actor 放置于关卡中。当然，要实现一定的功能，还需进行设计。例如“油门踏板”是汽车 Actor 的一个组件，通过脚本或代码，可表示已踩下了踏板，从而为汽车加速提供了逻辑。可以用此种方式让每个组件与整体 Actor 产生互动。

组件实例化：与普通子对象的默认行为相反，创建为 Actor 内子对象的组件都进行了实例化，即每个 Actor 实例都获得了组件的独特实例。例如汽车 Actor 类中有一个组件是车轮。在创建一个汽车实例（对象）时，将为这辆汽车的 4 个车轮组件专门创建新实例，与其他汽车的车轮组件不共用内存。否则，当一辆车在游戏世界中移动时，所有汽车的车轮都是转动的，这明显不是我们想要的行为。默认的组件实例化简化了为 Actor 快速添加独特子对象的过程。

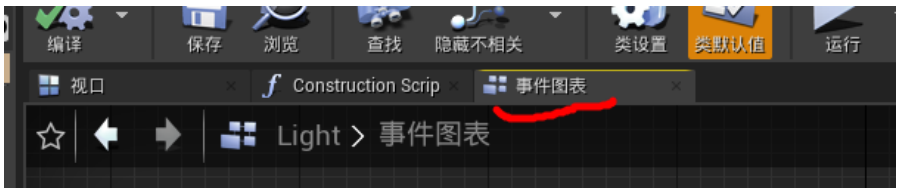
编辑器面板的第二个选项卡是构造脚本（Construction Script）。



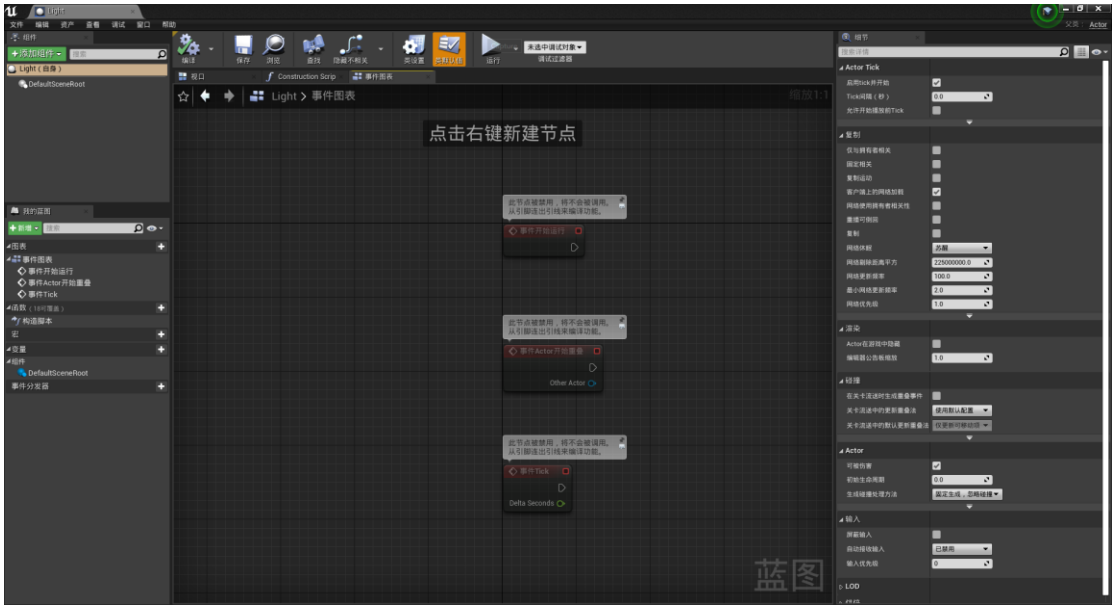
当在 Unreal Engine 关卡编辑器中放置或者在游戏中创建 Actor 时，会自动执行 Construction Script 编辑器面板中的蓝图节点，但在游戏正常过程中不会被执行。



编辑器面板的第三个选项卡是事件图表（Event Graph）。

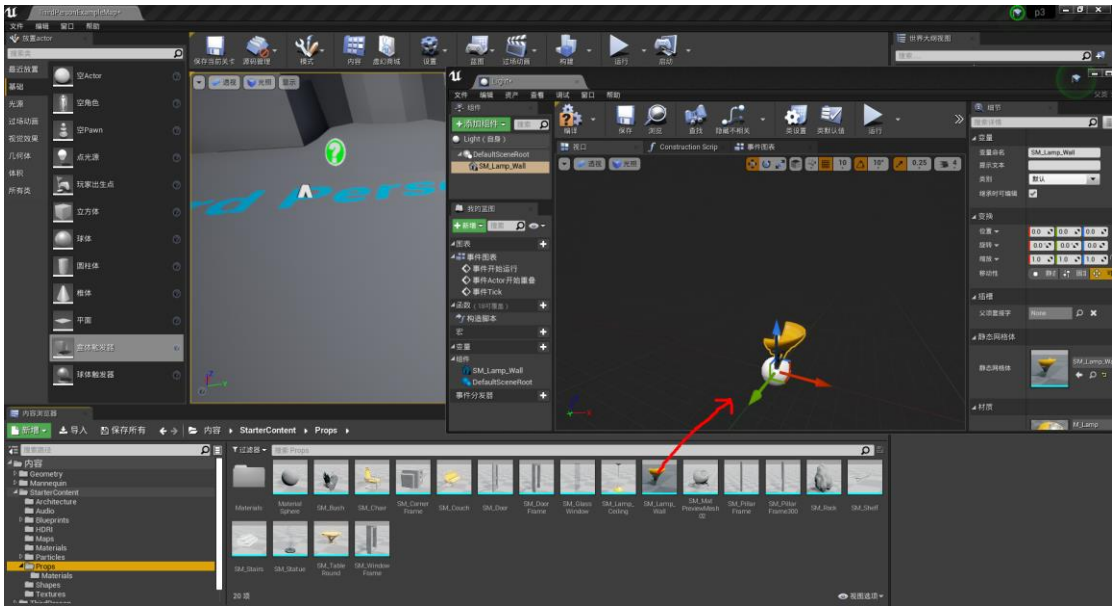


事件图表编辑器用于编写游戏脚本，当游戏运行时会被执行。（注意比较与 Construction Script 在运行时机上的区别）。事件图表使用事件和函数调用来执行动作，对与该蓝图相关的游戏事件做出反应。事件图表通常用于给蓝图的所有实例添加功能，以及设置交互性和动态反应。



介绍完编辑器面板后，开始为 Light 蓝图添加组件。有两种方法可以在蓝图中添加组件。以下将分别使用两种不同方法各添加一个组件到 Light 蓝图类中。

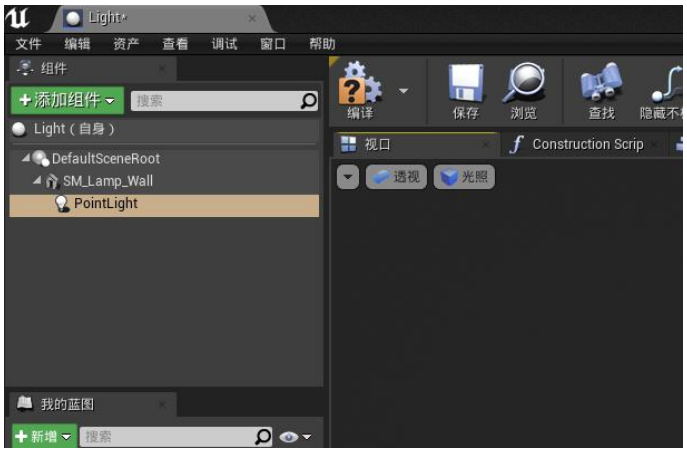
第一种方法：从主界面内容浏览器中，找到 StarterContent>>Prop>>SM_LAMP_WALL，鼠标左键拖入 Light 蓝图视口面板中，从而在蓝图中添加了一个称为 Static Mesh 的组件，在 Light 蓝图组件面板中默认名字为 SM_Lamp_Wall。

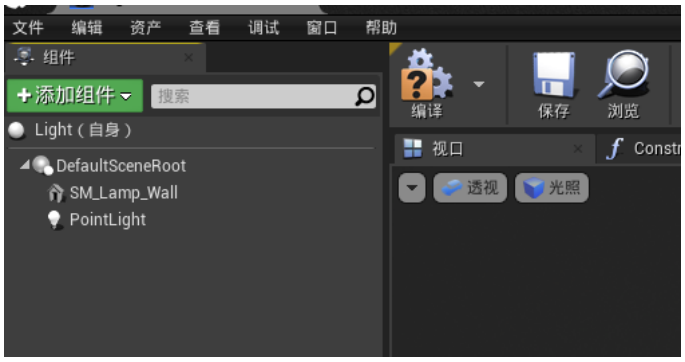


第二种方法：在 Light 蓝图编辑器中直接添加。具体操作是在组件面板中，单击“增加组件”按钮，在列表菜单搜索栏中输入“点光源”，选中找到的点光源组件。

目前 SM_Lamp_Wall 和 PointLight 组件都已经被添加进 Light 类了，如果我

们希望这两个组件的层次关系是平级的，而不是隶属关系。用鼠标左键按住 PointLight，并移动到 SM_Lamp_Wall 上方，松开鼠标左键。

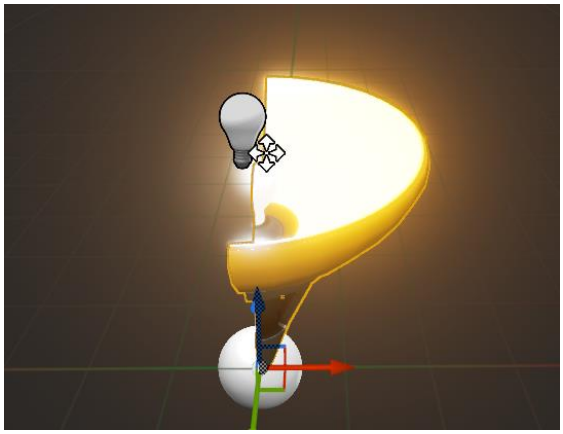




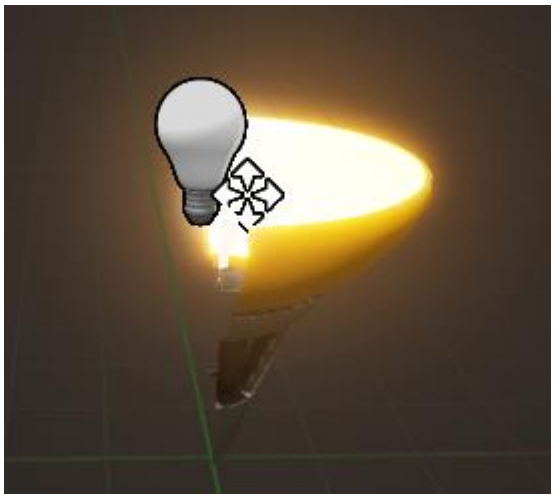
如果相同的操作重复一次，则 PointLight 再次隶属于 SM_Lamp_Wall。两个组件处于隶属状态时，移动 SM_Lamp_Wall，则 PointLight 与 SM_Lamp_Wall 的相对位置保持不变；移动 PointLight，则 SM_Lamp_Wall 保持不动。

适当调整 PointLight 的位置，让灯发射出的光线更合理。

目前 Light 类的视口的显示效果如下所示，下方白色的球是组件面板中 DefaultSceneRoot 的显示结果。



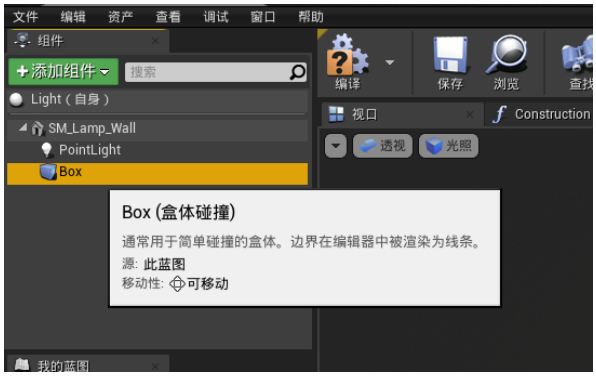
如果要消除白色的球，需要以其中的一个组件取代默认的根节点，例如 SM_Lamp_Wall。鼠标左键按住 SM_Lamp_Wall，移动到 DefaultSceneRoot 上方，松开鼠标左键。（PointLight 组件作为 SM_Lamp_Wall 的子组件会比较合理）

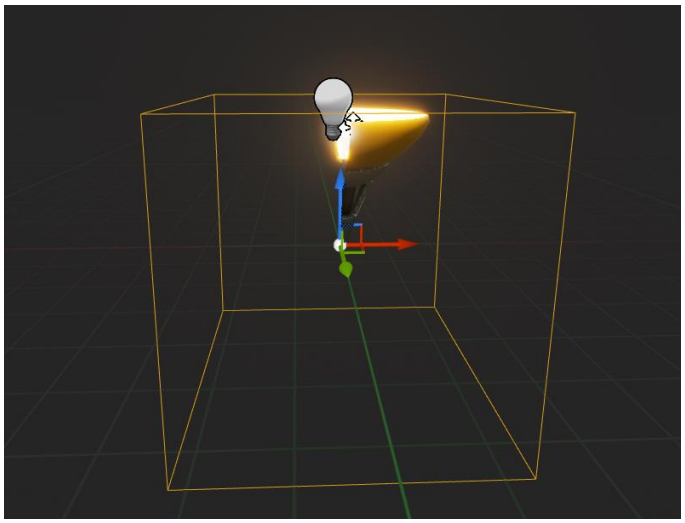


随着根节点的替换，Light 类的外形就更像一盏灯了。图中的灯泡在游戏场景中不会显现，只表示该处有一点光源。

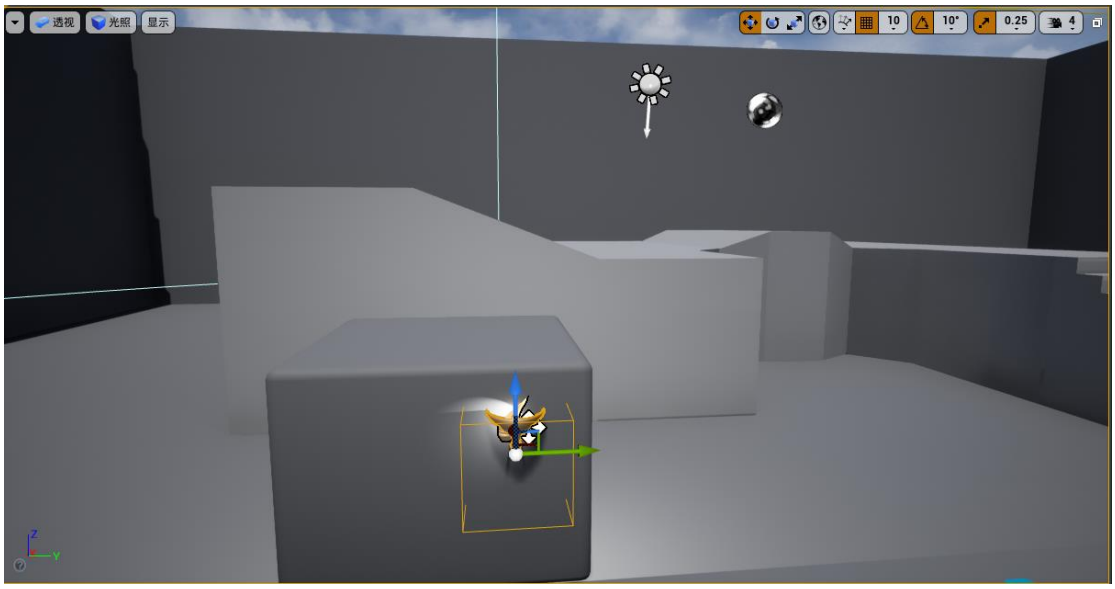
继续在“添加组件”处添加“Box Collision（盒体碰撞）”，系统会自动把该组件命名为 Box，把 Box 组件也设置为 SM_Lamp_Wall 的子组件，在视口中让它包围

住壁灯。

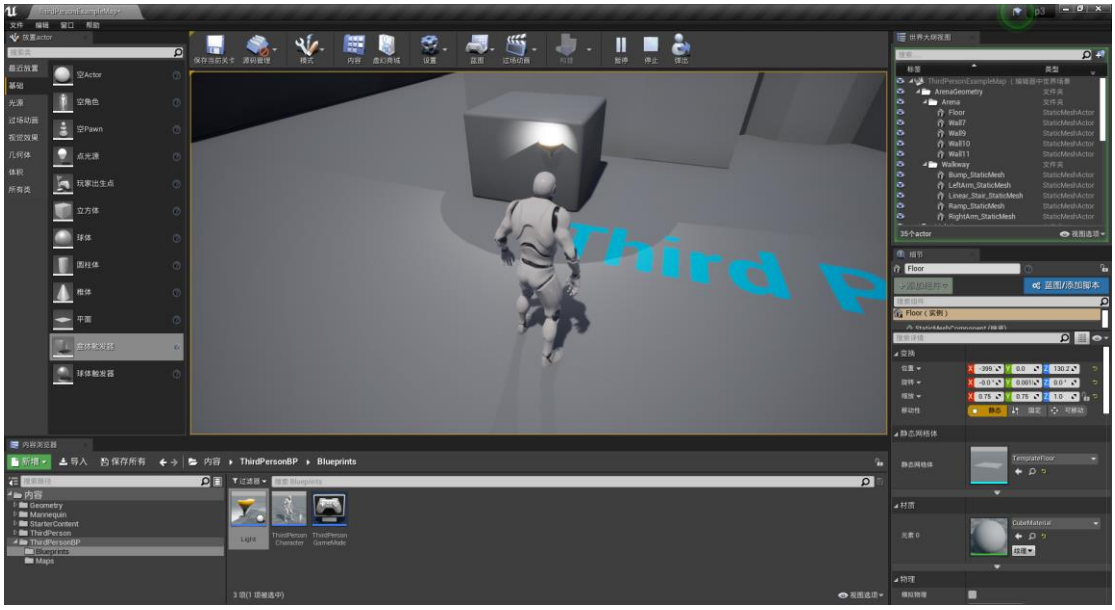




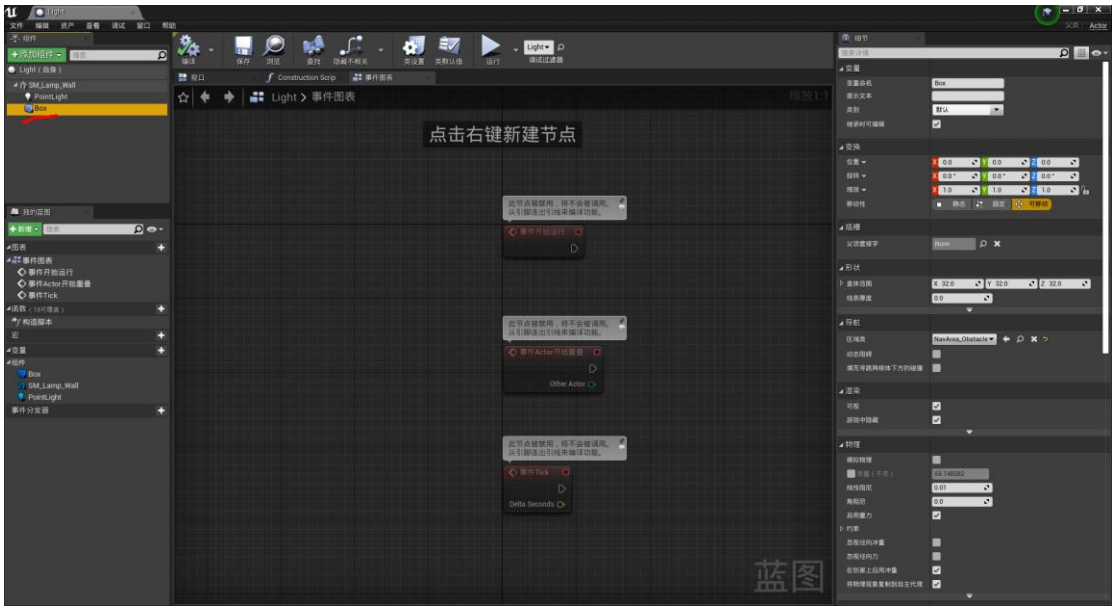
目前为止，我们完成了 Light 类组件方面的设计（还缺蓝图逻辑）。编译 Light 蓝图，然后回到 Unreal Engine 关卡编辑器（主界面），在内容浏览器中找到 Light 类（如果之前按课件设计，在 ThirdPersonBP>>Blueprints 文件夹中）。按住鼠标左键将其拖动放置到视口中，从而创建了一个 Light 蓝图类的实例对象，默认命名为 Light。在视口中通过平移、旋转、缩放，把灯放置在一个合理的位置，确保是角色能接触到的位置。注意灯的朝向和美观。



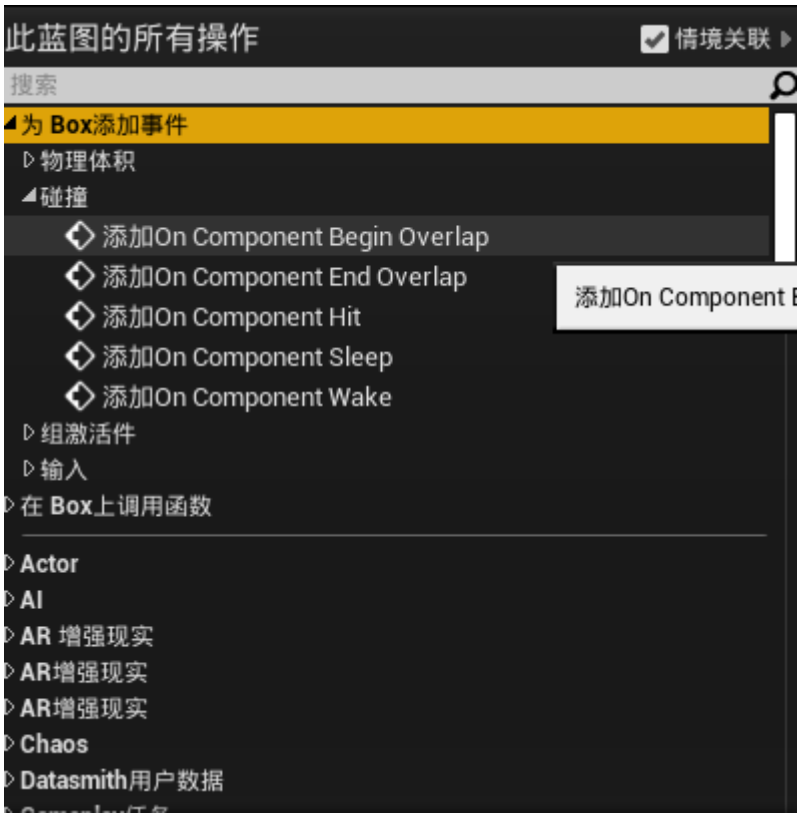
运行游戏，如果发现灯光效果不好，可以修改 Light 类中 PointLight 组件的位置。



回到 Light 蓝图，打开事件图表编辑器。可以把默认的 3 个节点删掉（以后如果需要还可以添加，也可以不删）。



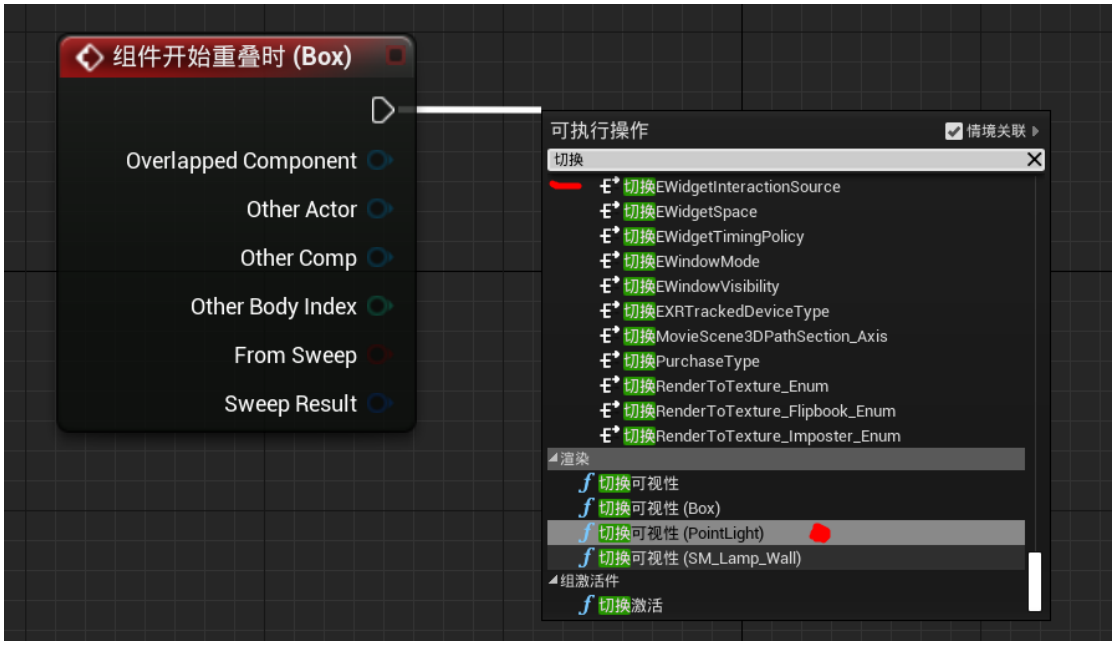
在左侧组件处选中 Box 组件，然后在事件图表编辑器空白处鼠标右键，选择“为 Box 添加事件”>>“碰撞”>>“添加 On Component Begin Overlap”，



在事件图表编辑器界面中会出现一个红色的“组件开始重叠时（Box）”碰撞事件节点，括号中为指定的组件名字，即为 Box 这个碰撞体添加了一个碰撞检测事件。当它与其他 Actor 之间发生重叠碰撞后，“组件开始重叠时（Box）”事件节点

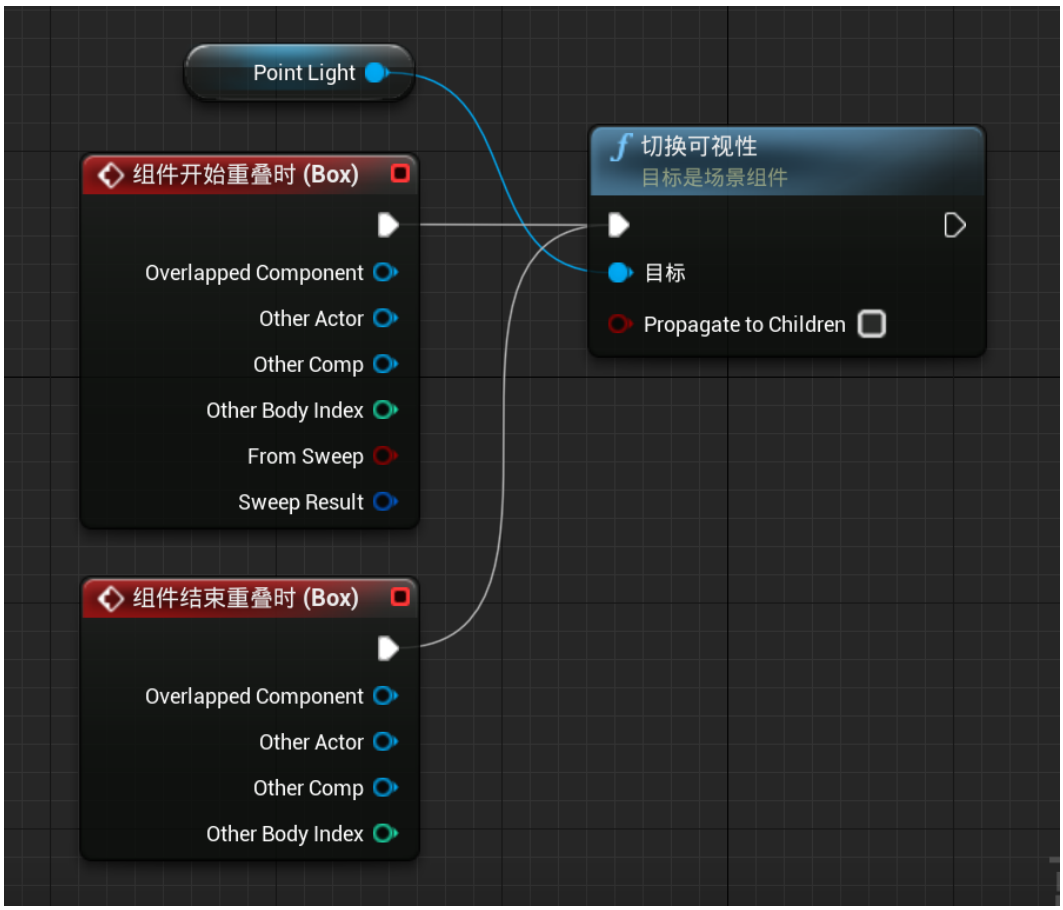
将会被触发执行。

继续添加后续节点，原理与前几个小节一样，接触后改变灯的开关状态。按住“组件开始重叠时（Box）”事件节点右侧的执行引脚到空白处，松开鼠标出现关联菜单，在搜索栏中输入“toggle”（或者“切换”），选择“渲染”>>“切换可视性（PointLight）”。这个节点控制 Light 蓝图中已添加的 PointLight 组件的可视状态。



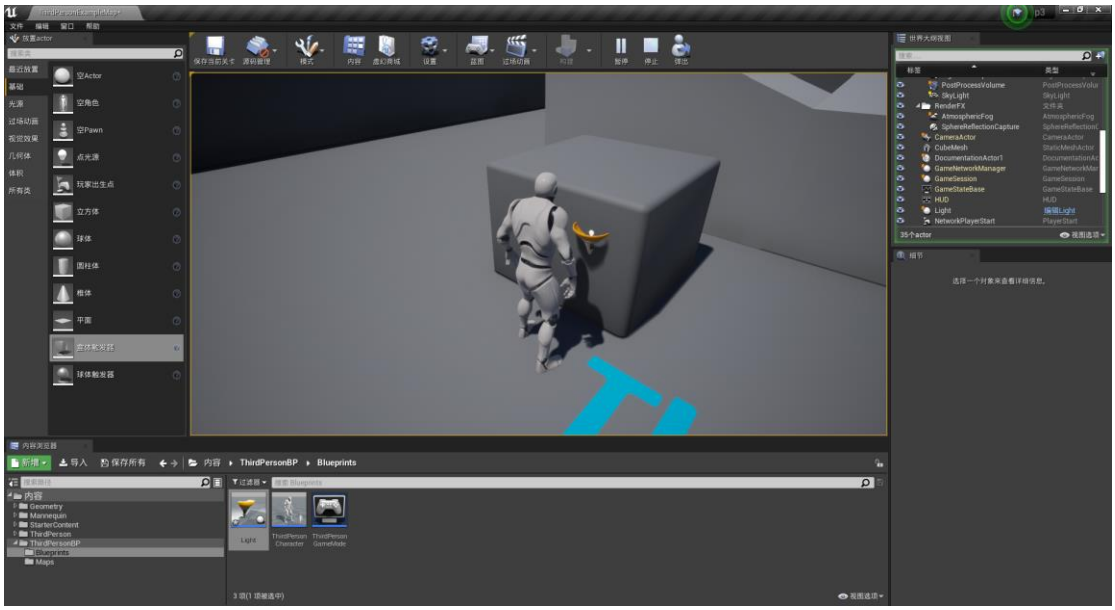
事件图表中新增了两个节点。

同样道理，添加“组件结束重叠时（Box）”碰撞事件节点，并做好相应的引脚连接。



编译 Light 蓝图，然后运行游戏。Light 中 PointLight 默认为打开状态，当接

触 Light 后，PointLight 关闭，远离 Light，PointLight 打开。



作业：创建第一人称游戏，实现第 2~3 课的内容，并指出第一人称游戏和第三人称游戏在功能实现的区别。