第 2 课 蓝图节点

1. 认识蓝图（Blueprint）

虚幻引擎中的蓝图——可视化脚本系统，是一个完整的游戏脚本系统，它为设计人员提供了一般仅供程序员使用的所有概念及工具。

（1）Blueprint 概述

Blueprint 的工作原理是通过各种用途的节点构成图表来进行工作，这些节点包括针创建实例对象、函数和事件，然后把这些节点连接到一起，从而创建复杂的游戏性元素。

Blueprint 节点中的参数和返回值以引脚的形式存在，只需知道节点的功能，不需要知道节点的内部实现，可以让未接触过编程的人制作出游戏。

Unreal Engine 4 C++为程序员提供了基于蓝图功能的语法标记，可以对蓝图语法功能进行扩充。

（2）Blueprint 类型

常见的蓝图类型有关卡蓝图（Level Blueprint）、蓝图类（Blueprint Class）、蓝图宏（Blueprint Macro）和蓝图接口（Blueprint Interface）。

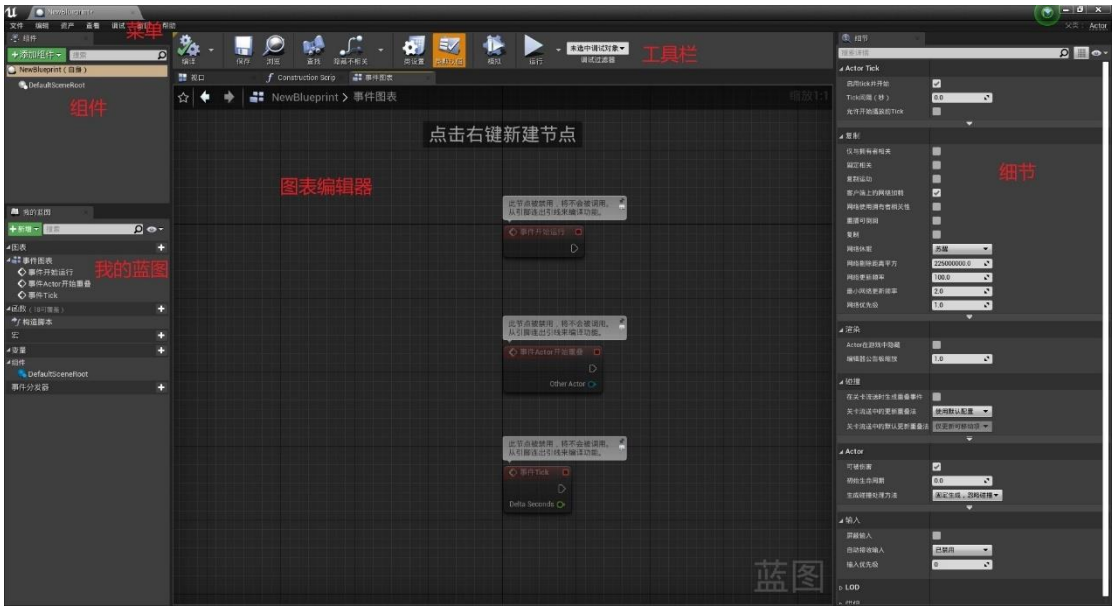
Level Blueprint 描述关卡的全局事件，包括关卡中的游戏元素以及功能。通常情况下，每个关卡有一个 Level Blueprint。

Blueprint Class，简称 Blueprint（蓝图），用于描述用户自定义的 Actor，这些 Actor 被创建后可以作为实例（对象）放入关卡中。

Blueprint Macro 是存放了一组节点序列的宏，让用户方便调用。

Blueprint Interface 与编程中的接口概念一样，提供了公共函数接口，允许通过该接口访问多种类型对象。任何使用该接口的蓝图必须实现接口函数。

（3）Blueprint 编辑器（Class 蓝图编辑器）



菜单和工具栏：基础操作和常用操作

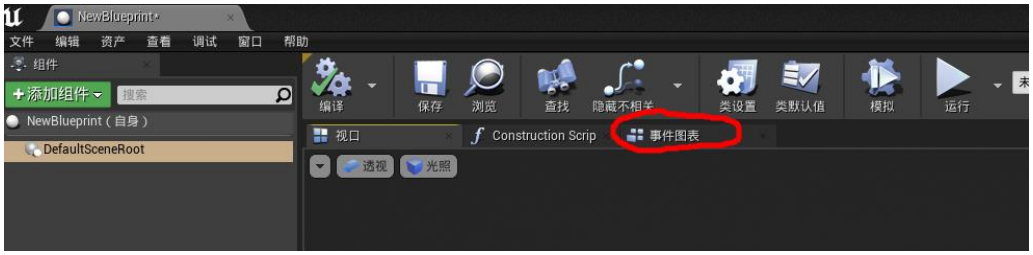
组件面板：蓝图包含的组件，组件之间的层次结构。

我的蓝图：蓝图中的图表、函数、宏、变量、脚本，是蓝图的大纲视图，包含蓝图中的所有元素。

图表编辑器：蓝图编辑的主界面，是蓝图的核心功能区域。其中视口页面显示 Actor 的视觉效果，construction script（构造函数脚本）用于定义创建或者更新 Actor 时执行的函数，事件图表用于定义游戏过程中执行的函数。

细节面板：编辑蓝图中所选项的属性。

2、事件（Event）类型节点

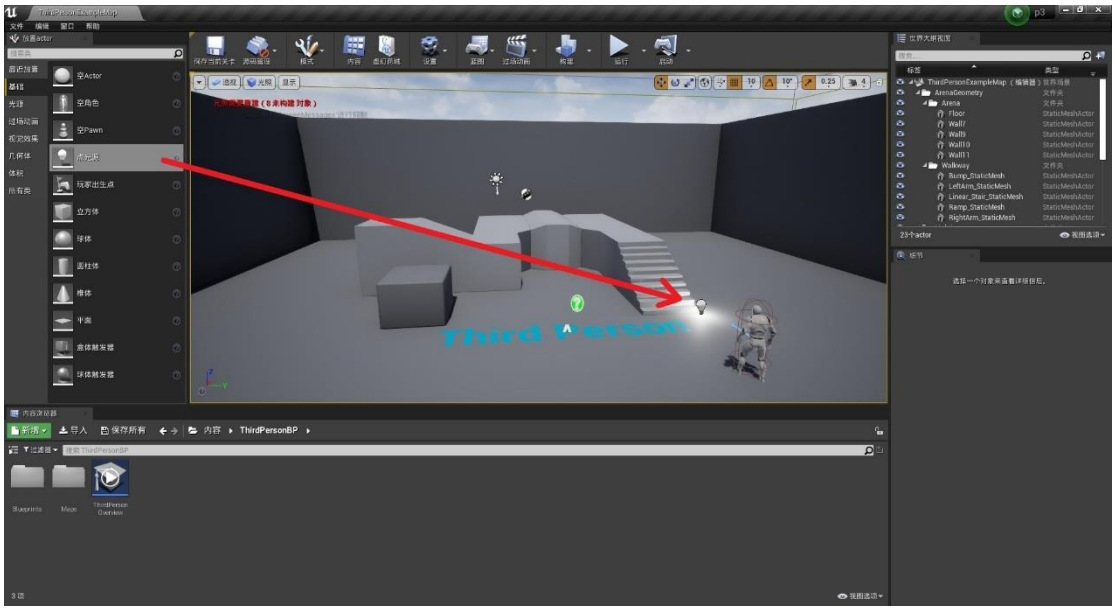


Event 类型节点是构建事件图表的重要元素，可以针对游戏中发生的特定事件（如游戏开始、关卡重置、受到伤害）进行回应。

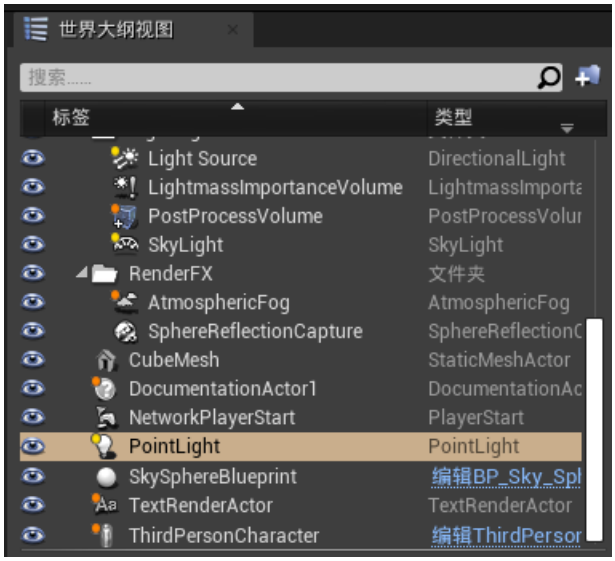
（1）事件开始运行（Event BeginPlay）节点

Event BeginPlay 节点描述当游戏开始运行时，在此 Actor 上触发的事件。游戏开始运行时，所有 Actor 均会自动触发各自蓝图中的 Event BeginPlay 节点。本节通过实现游戏运行时灯光关闭的效果来认识这个节点的用法和作用。

创建 Unreal Engine 4 第三人称工程项目。在左侧模式面板中找到“基础”“点光源”，拖动到视图窗口中。

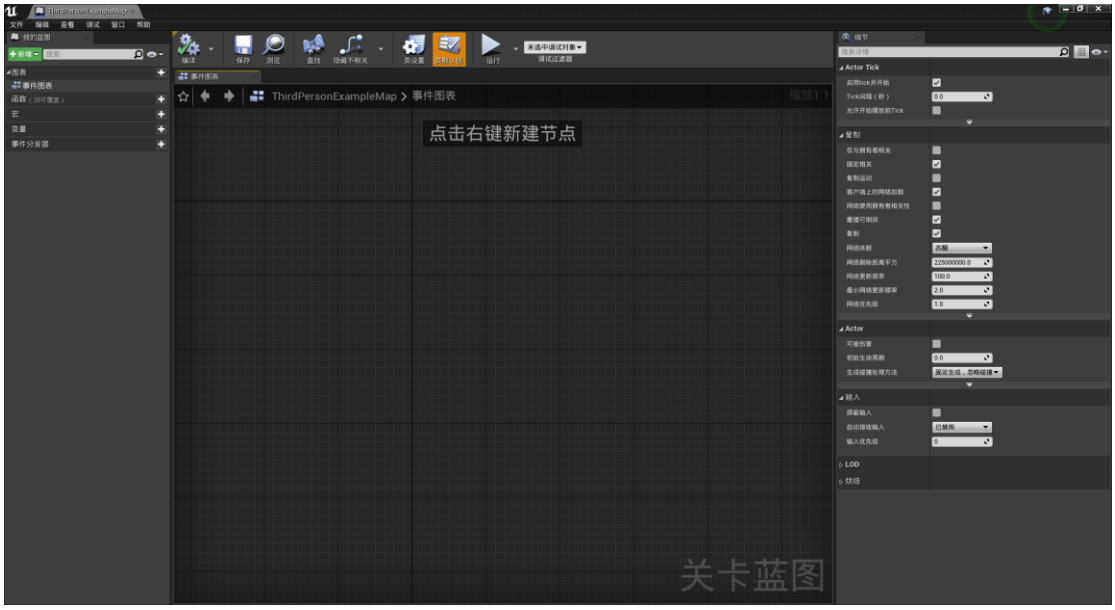


注意到，当添加点光源后，世界大纲视图中会出现该资源，名字为 PointLight（资源实际名字是英文）。



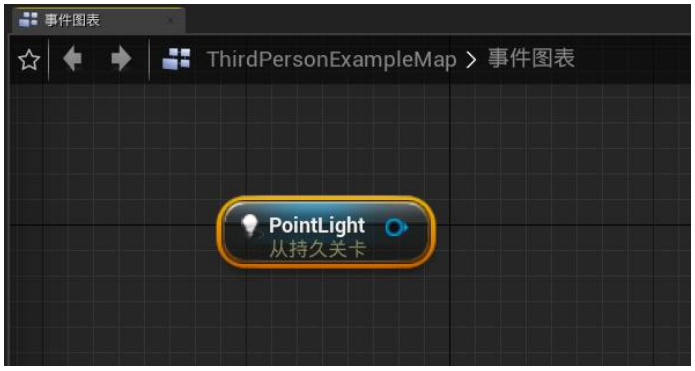
选择工具栏中的“蓝图”“打开关卡蓝图”，打开 Level 蓝图编辑器（关卡蓝图编辑器）。

Level 蓝图编辑器相比于 Class 蓝图编辑器，布局上没有组件面板。



关卡蓝图用于定义关卡打开时的事件。由于需要在关卡开始之时在关卡蓝图事件图表编辑器界面中，鼠标右键，出现关联菜单，搜索“Create a Reference to PointLight”选项(或者搜索“引用”)，在关卡蓝图中创建一个对 PointLight 的引用，该引用针对的对象是此前拖入视图窗口的点光源。

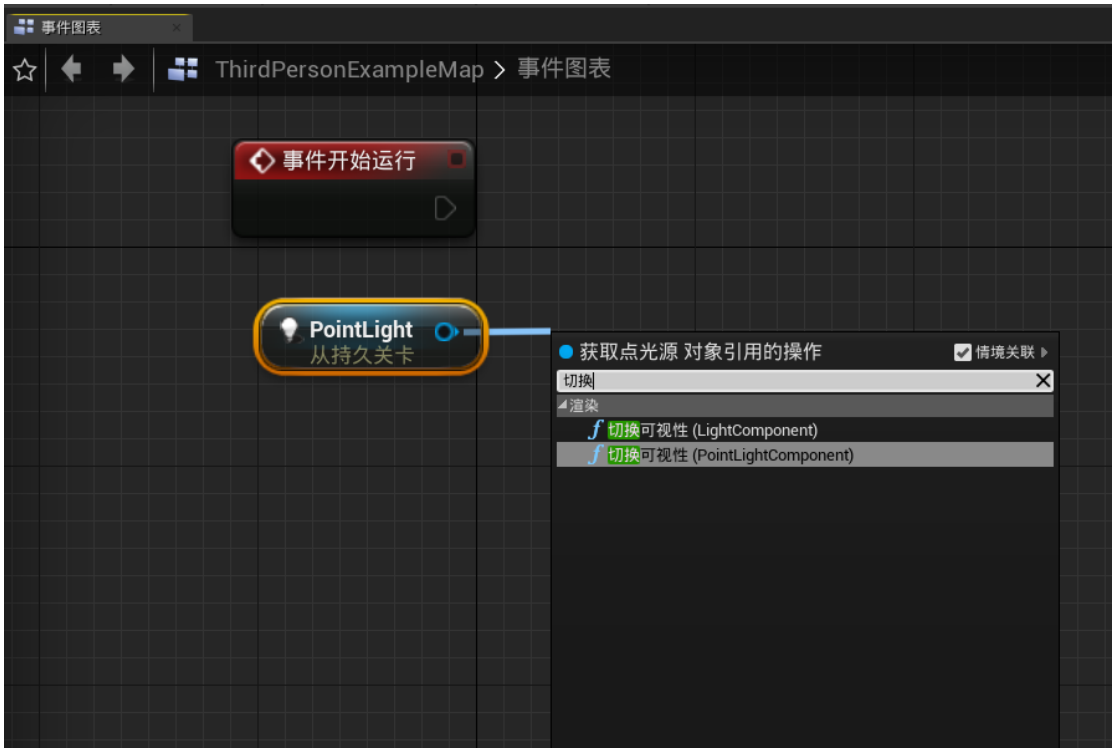
此时在事件图表中出现了一个显示为PointLight 的蓝色变量节点，该节点就是刚创建的对点光源 PointLight 的引用。蓝色变量节点属于 Object 类的节点。



在事件图表编辑器窗口空白处再次鼠标右键，搜索“Event BeginPlay”（或者“事件开始运行”），创建事件开始运行节点，红色节点表示事件节点。

回到 PointLight 节点，找到节点右侧的数据引脚（data pins）。数据引脚用来输入数据到节点或者从节点输出数据。数据引脚具有特定的数据类型（例如

整型、浮点型），可以与相同类型的变量或者另一节点中相同类型的引脚相连。现在按住鼠标左键拖动 PointLight 的蓝色数据引脚到事件图表编辑器的空白处，然后松开鼠标，出现关联列表，搜索“toggle”（或者“切换”），缩小搜索范围，找到并选择“切换可视性（PointLightComponent）”。



事件图表编辑器会出现两个新的节点“读取变量 PointLightComponent 的值”和“切换组件的可视性”，并且 3 个节点之间的数据引脚由蓝色连接线连接在一起。

这些蓝色连线是数据连线，它们把一个数据引脚连接到同种类型的另一个数据引脚上，代表着数据流向。数据连线显示为带颜色的箭头，用于可视化地表示

数据的转移，箭头的方向代表数据的移动方向。一般情况下，节点左侧的数据引脚表示数据流进节点，节点右侧的数据引脚表示数据流出节点。

三个节点之间的关系是，中间“读取变量 PointLightComponent 的值”节点从左侧“PointLight”变量获取 PointLightComponent 组件，然后调用“切换组件的可视性”节点，从而运行 PointLightComponent.ToggleVisibility 方法，切换 PointLight 类型变量的可视性。

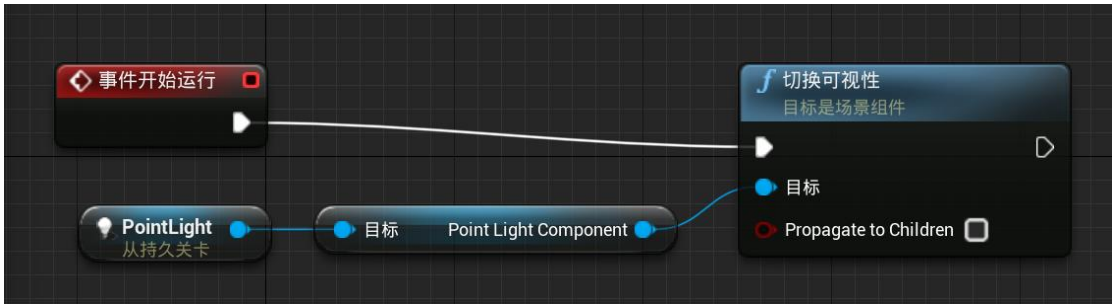
目前已经创建好实现游戏运行时关闭灯光的所有节点，但离实现效果还差最后一步。目前已有“切换组件的可视性”这个控制灯光开关的节点，但目前它只是被摆在了事件图表编辑器中，游戏运行时并不会被执行。

每个可执行的节点，左上角都有一个白色的五边形，称为输入执行引脚（Execution Pin），右上角的白色五边形称为输出执行引脚。把各节点的执行引脚连在一起，就形成了一个执行流。当激活一个节点的输入执行引脚时，执行该节点。一旦执行完一个节点，那么会激活输出执行引脚，以继续激活执行流中的下一个节点。

激活执行引脚的常见事件是“事件开始执行（Event BeginPlay）”节点，该节点右侧有一个输出执行引脚，当游戏运行时，节点被自动调用执行，在输出执行引脚产生一个信号，通知执行流的下一个节点运行。

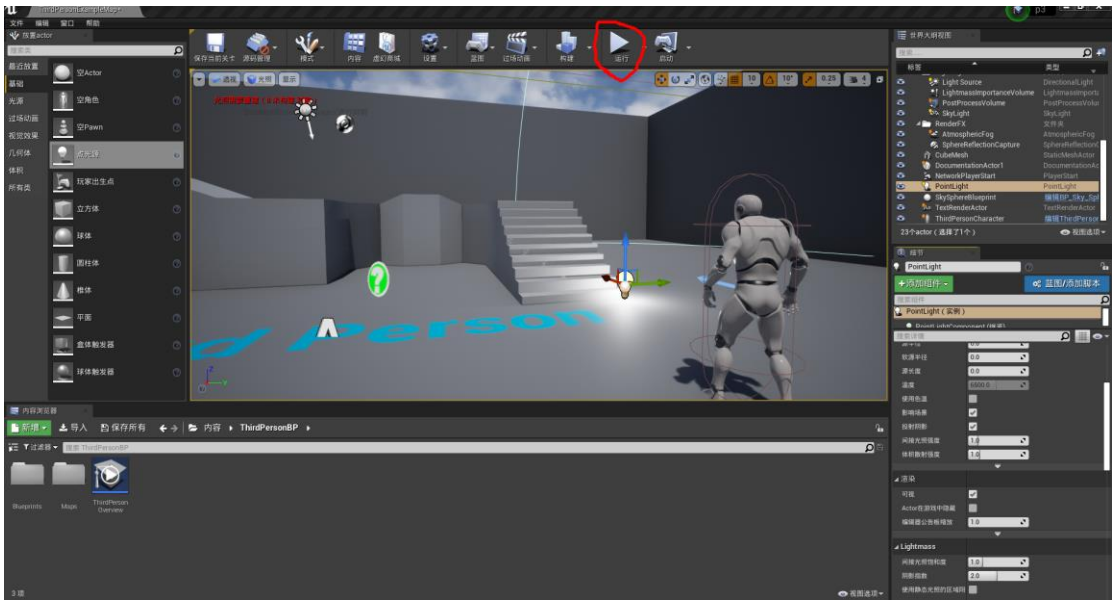
根据设计，游戏运行开始就需要执行“切换组件的可视性”节点，因此把“事件开始执行”节点的输出执行引脚与“切换组件的可视性”节点的输入执行引脚相连。

执行连线是白色箭头，箭头方向是执行流程的走向。



接下来编译Level蓝图。找到“编译”按钮，未编译时“编译”按钮上有一个问号，编译成功后显示为绿色的✔。

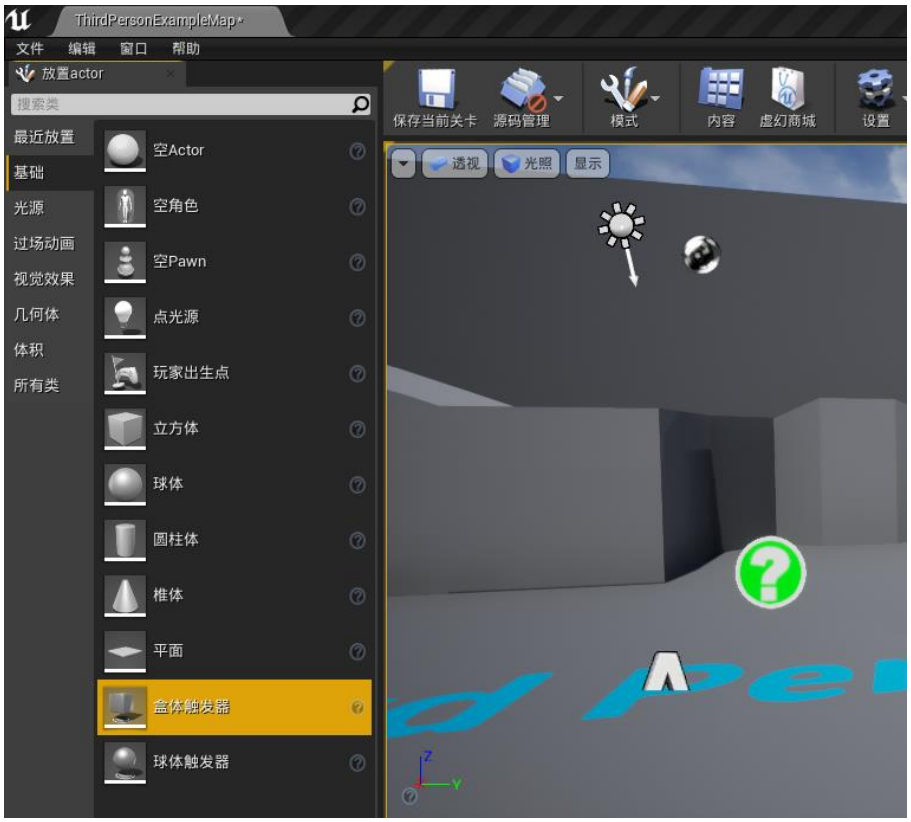
一切就绪后，切换窗口回到视图窗口（最小化 Level 蓝图窗口即可），点“运行”按钮运行游戏，在游戏还没运行前，灯光是亮的，游戏运行后灯光自动关闭。



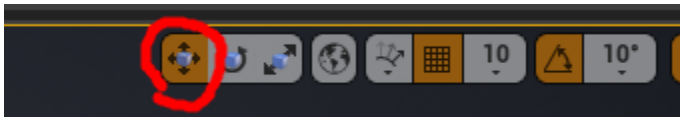
（2）OnActorBeginOverlap / OnActorEndOverlap 节点

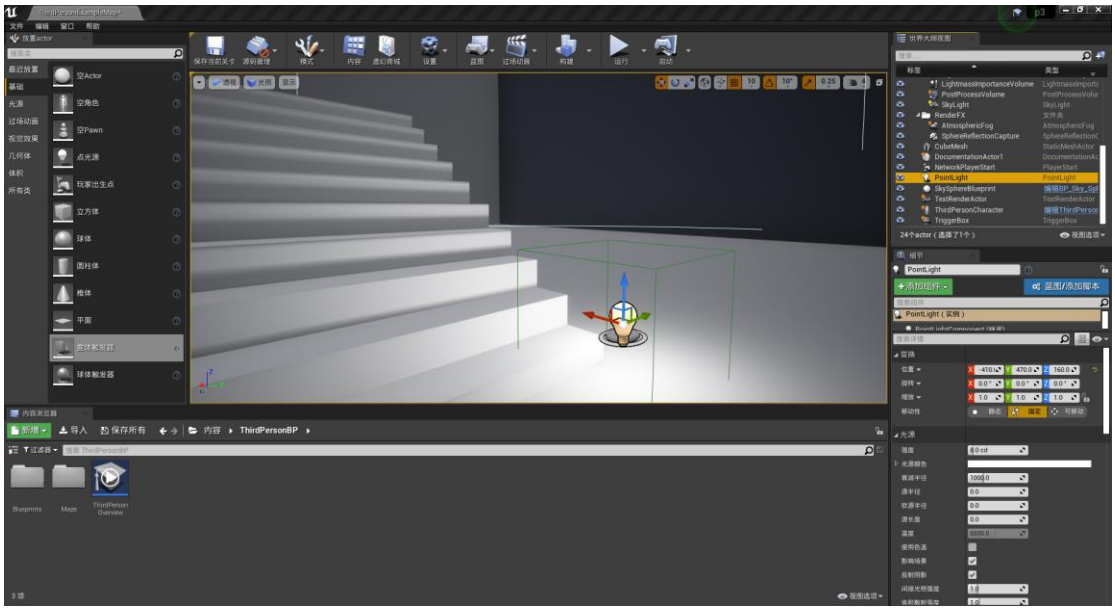
当两个 Actor 开始重叠到一起时（例如玩家进入了触发机关区域），包括一个 Actor 创建时的位置与另一个 Actor 重叠，触发 OnActorBeginOverlap 事件节点。此外，要触发事件还必须满足双方 Actor 都具有碰撞体积（例如玩家撞上了墙），且双方 Actor 都必须把“生成重叠事件（Generate Overlap Events）”设置为 true。本节实现一个通过碰撞使灯光切换状态的机制。

继续沿用上一节创建的 PointLight。从左侧模式面板选择“基础”“盒体触发器（TriggerBox）”。



把 TriggerBox 放置到场景中，通过平移操作让 TriggerBox 完全包围住 PointLight。



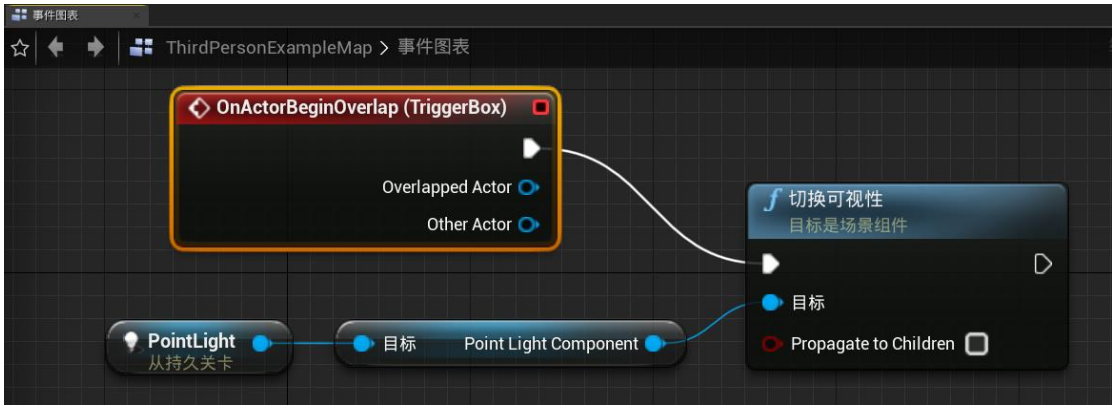


点不中 TriggerBox、移动不到合适位置，以及有强迫症的同学，在右上方“世界大纲视图”中找到 PointLight，记下位置信息，然后在“世界大纲视图”中找到 TriggerBox，直接修改位置，与 PointLight 的位置相同。如果要调整 TriggerBox 的大小，可以修改“缩放”，就在“位置”的下方。

在上一节的蓝图中，通过使用 Event BeginPlay 事件节点，使得游戏一开始运行，PointLight 就会熄灭，现在使用 OnActorBeginOverlap 碰撞事件节点，取代 Event BeginPlay 事件节点。

打开关卡蓝图，删除“事件开始运行”节点。然后回到视图窗口（主界面），在右上方“世界大纲视图”中鼠标左键单击选中 TriggerBox，再切换到关卡蓝图窗口，在事件图表编辑器空白处鼠标右键，选择“为 Trigger Box1 添加事件”>>“碰撞”>>“添加 On Actor Begin Overlap”。

事件图表编辑器中增加了一个红色的 OnActorBeginOverlap(TriggerBox)碰撞事件节点，括号里的名字就是指定的 Actor 的名字，表明当 TriggerBox 被检测到与其他 Actor 发生重叠碰撞时，该事件节点会被执行。

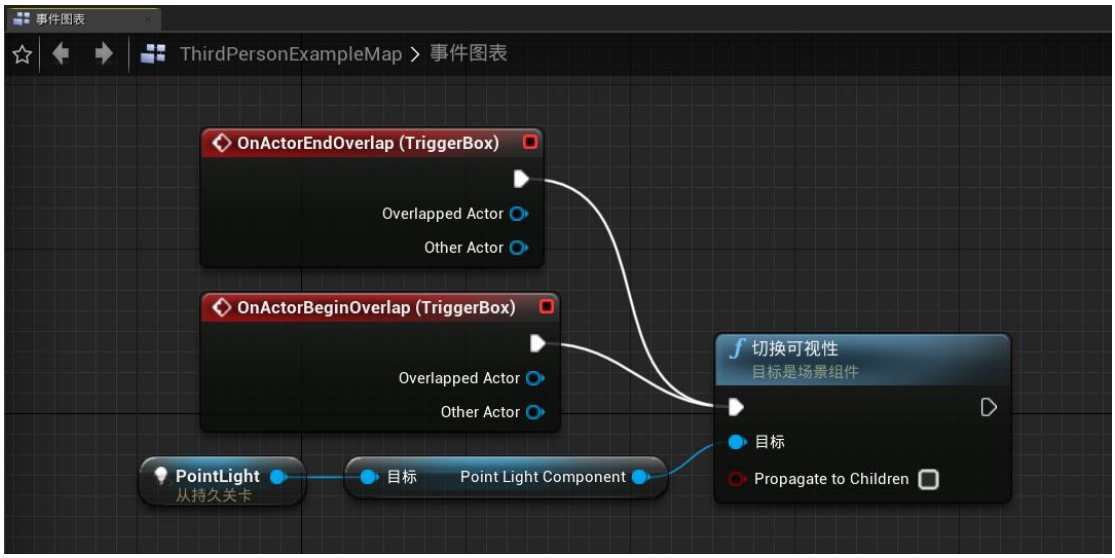


编译后运行游戏，当玩家靠近灯光时，触发 OnActorBeginOverlap(TriggerBox)事件，灯光切换为关闭；再次远离光源后再靠近，灯光切换为开启。

了解 OnActorBeginOverlap 碰撞事件后，就会推断出 OnActorEndOverlap 碰撞事件节点的功能，即当两个 Actor 停止重叠，或者其中一个 Actor 被销毁时，触发执行。同样的，要触发事件还必须满足双方 Actor 都具有碰撞体积（例如玩家撞上了墙），且双方 Actor 都必须把“生成重叠事件（Generate Overlap Events）”设置为 true。

参照 OnActorBeginOverlap 碰撞事件节点的添加方法，实现功能：接触时灯

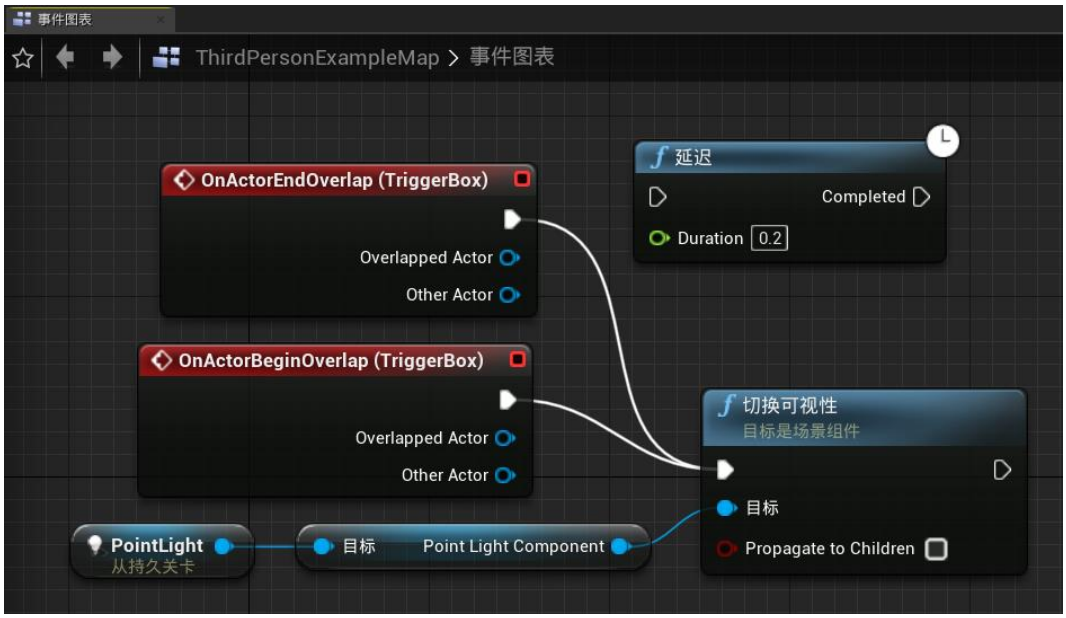
光关闭，离开时灯光开启。



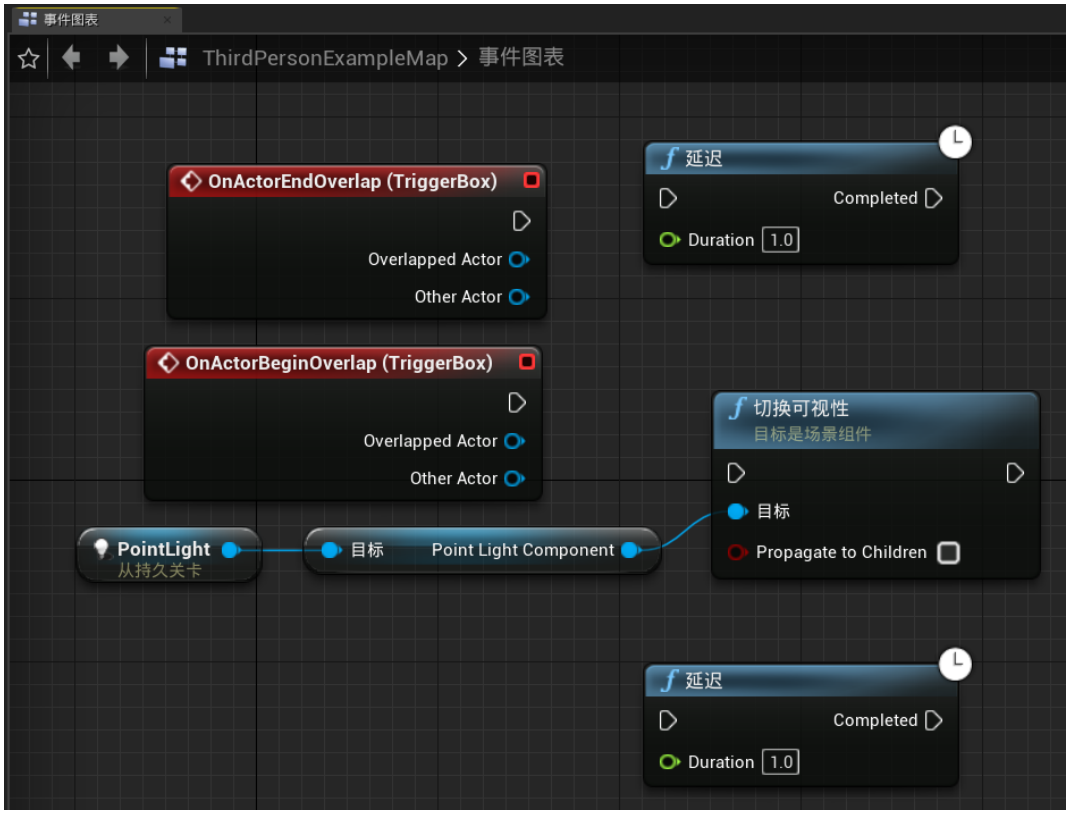
思考：如果反过来，要求接触时灯光开启，离开时灯光关闭，该如何实现？

还可以进一步丰富切换灯光开关状态的机制，让玩家接触时过一阵再开灯，离开后过一阵子再关灯。在关卡蓝图事件图表编辑器中，搜索“delay”（或者“延迟”），添加延迟节点。

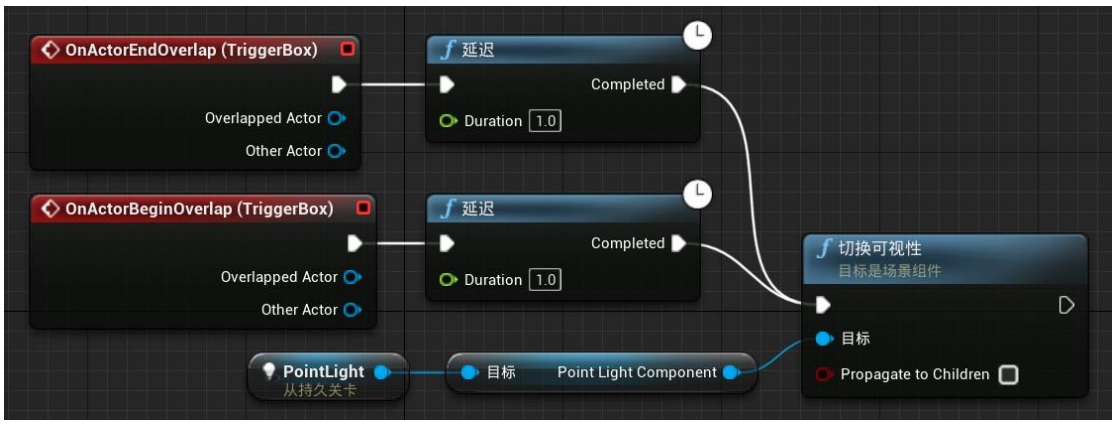
延迟节点用于延后下一个节点的执行时间，左侧 Duration 值用于设置延时时长，单位是秒，默认值是 0.2，设置时可以直接在后面的文本框中输入一个 Float 类型常数，也可以添加一个 Float 类型的变量并赋值给它。



添加两个“延迟”节点，并把 Duration 值都改成 1。可以通过 Ctrl+C 和 Ctrl+V 的方法进行复制粘贴。



修改执行引脚的连接顺序。可以通过拖动节点的执行引脚到指定的执行引脚，来改变原有的连接顺序；也可以按住键盘上的 Alt 或 Ctrl 键单击想要改变的连线，来删除或改变连线方向。适当调整节点的位置，让整个蓝图设计看上去美观。



编译运行游戏，当接触 PointLight 时，经过 1 秒延迟，灯光切换状态；当远离 PointLight 时，经过 1 秒延迟，灯光再次切换状态。