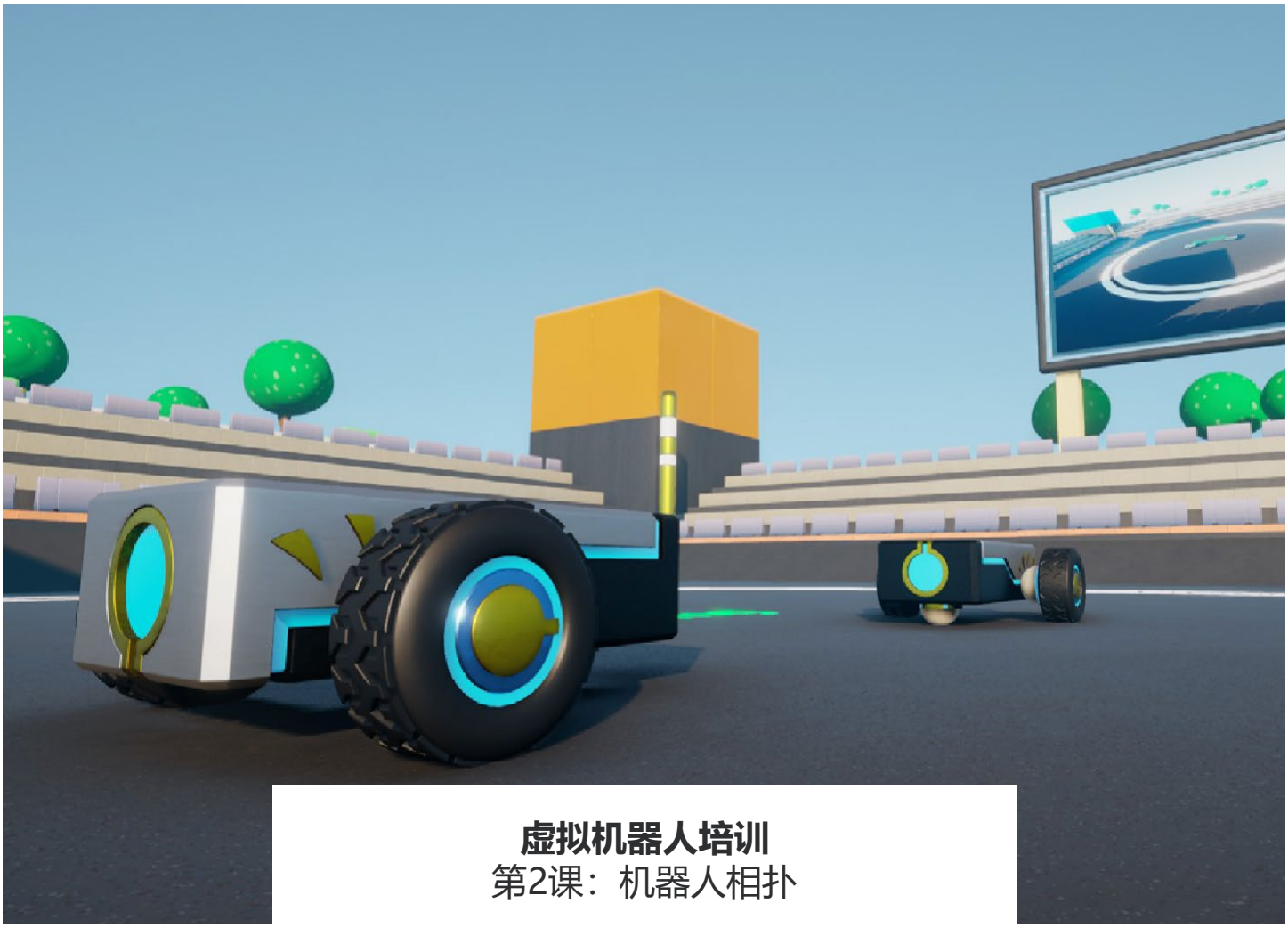
虚拟机器人培训

第2课：机器人相扑

UNREAL ENGINE

学生指南

目录



虚拟机器人培训
第2课：机器人相扑

目标

本课程将介绍为什么机器人需要传感器，如何使用传感器接收信息，如何使机器人根据信息做出决策。

本次课程所用的沙盒为机器人相扑挑战。如果你还不太了解相扑，这项运动需要你留在场内，尽力将对手逼出场外。快看看你的机器人身手如何吧！

开始入门

入门指南

如果你还是初次安装和使用虚幻引擎，请在继续本次学习活动之前先完成入门指南。该指南包含了安装虚幻学习包项目文件的指示，能帮助你完成本次学习活动。

你可以在此处获取入门指南！

开始入门续

课程内容

我们建议学生按顺序学习课程，以便充分理解虚幻学习包，了解如何在虚幻引擎环境中支持机器人创建和马达/传感器编写。

虚拟机器人培训通过应用物理学探索了机器人工程设计的概念，使学生能够立即获得代码反馈，并传授他们所需了解的原则/代码语言知识，以便应用于未来接触的其他物理机器人系统中。

第2课：机器人相扑

简介

你想制作一个机器人相扑力士吗？我们也想！

不过问题已经来了。例如，如何让机器人在不接收额外输入的同时遵守规则呢？如何让机器人“看到”决定输赢的出界线呢？你马上就会明白了！

课程概述

在本课程中，我们会使用通过传感器控制的机器人，通过编程使它停留在相扑圈内。为此，你需要学习许多传感器的相关知识，包括光传感器提供的信息类型、以及如何根据实时反馈帮助机器人采取特定操作。

我们会重点聚焦以下技能：

- 理解对传感器的需求，以及传感器如何为机器人程序提供信息
- 探索传感器放置在机器人的不同位置时产生的效果。
- 使用阈值，将一定范围内的传感器值转换为布尔
- 使用条件语句，根据传感器输入执行特定操作
- 实现代码命令，帮助传感器确定机器人运动，包括：
    - 从光传感器处获取输入
    - 设置条件语句
    - 向每个马达发送运动命令
    - 设置循环操作

代码关联

本活动会介绍条件语句的重要性。你的机器人配备了一个可以检测明暗的传感器。为了确保它能够正常运作，我们会将实时传感器值和一个阈值进行对比，确定机器人是否能看到竞技场或出界线。使用条件语句之后，机器人就能根据前方是否为竞技场或者出界线，从而执行不同的操作。

开始授课

机器人构造：平衡与设计

在打开的工程文件中查看全新的机器人。该模型的构造是相扑战斗的理想选择，我们会说明原因的。

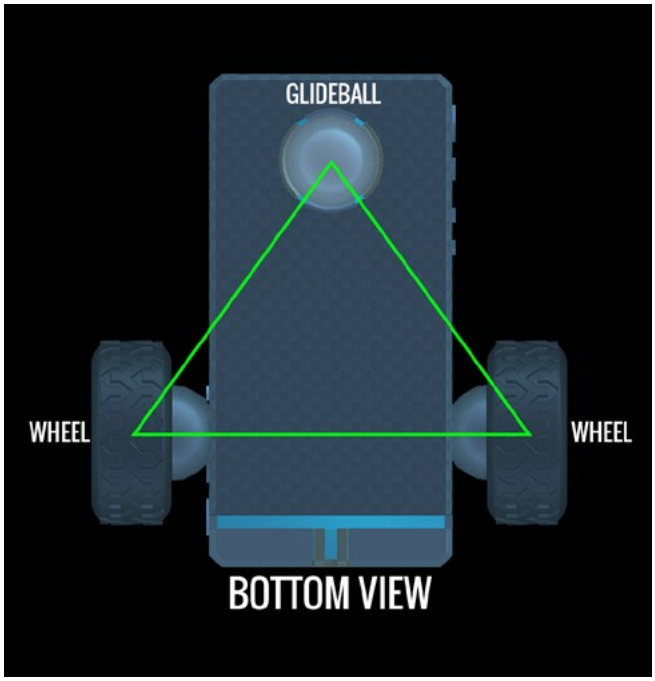


图2-1

当机器人的地面接触点位置决定了稳定性。在我们的示例中，两个轮子和滚球构成了一个三角形，三个接触点的平衡性较好。以下是可以考虑的综合因素：

- 当载具躯体上的轮子间距较宽时，机器人就难以侧翻
- 滚球离轮子越远，载具就越难以前后翻倒
- 载具躯体越扁平（例如水平），就能让质量集中在轮子上，使其难以翻倒。如果载具的躯体较高（例如垂直），载具质量就会位于轮子上方，导致机器人容易翻倒。
- 使轮子与载具躯体保持齐平，就能让载具的重量将轮子压在竞技场地面上，在推挤其他机器人时也可以增强轮子的牵引力。

确定目的

我们正在为特定的目的制作机器人。因为这是一场机器人相扑，我们可以构想一个环境，设定需要遵守的规则，并定义一些基本要求。

环境

- 机器人起初位于平坦的方形竞技场中
- 竞技场设有清晰可见的出界线

规则

- 机器人始终在移动
- 机器人会尽力留在竞技场范围内

要求

- 使用传感器来“查看”地面
- “看见”竞技场地面时径直向前行驶
- “看见”出界线时转向

实践练习

练习1-a：打开基础机器人

在本活动中，我们会打开工程文件，其中包含了一个竞技场中的机器人，目前它还无法移动。我们首先会添加光传感器，然后测试传感器值，再构建额外代码，控制机器人移动的时机。

- 在**内容浏览器**中，找到**“内容”** -> **“LearningKit_Robots”** -> **“Maps_Robots”**文件夹。
- **点击L2**文件夹。
- **双击**“Map_2-1_LineDection”打开它。
- 你现在看到一个拥有黑色边线的白色圆形竞技场，场地中间有一个机器人。

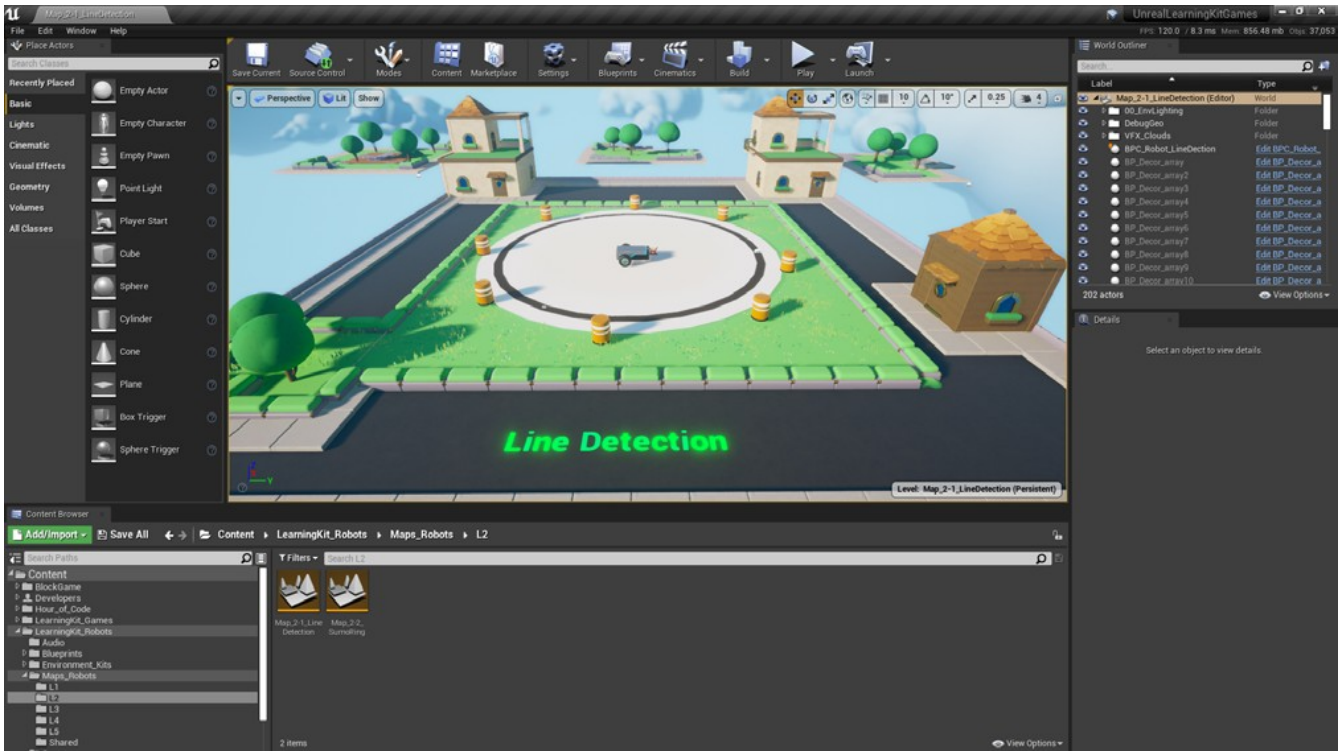


图2-2

练习1-b：为机器人添加光传感器

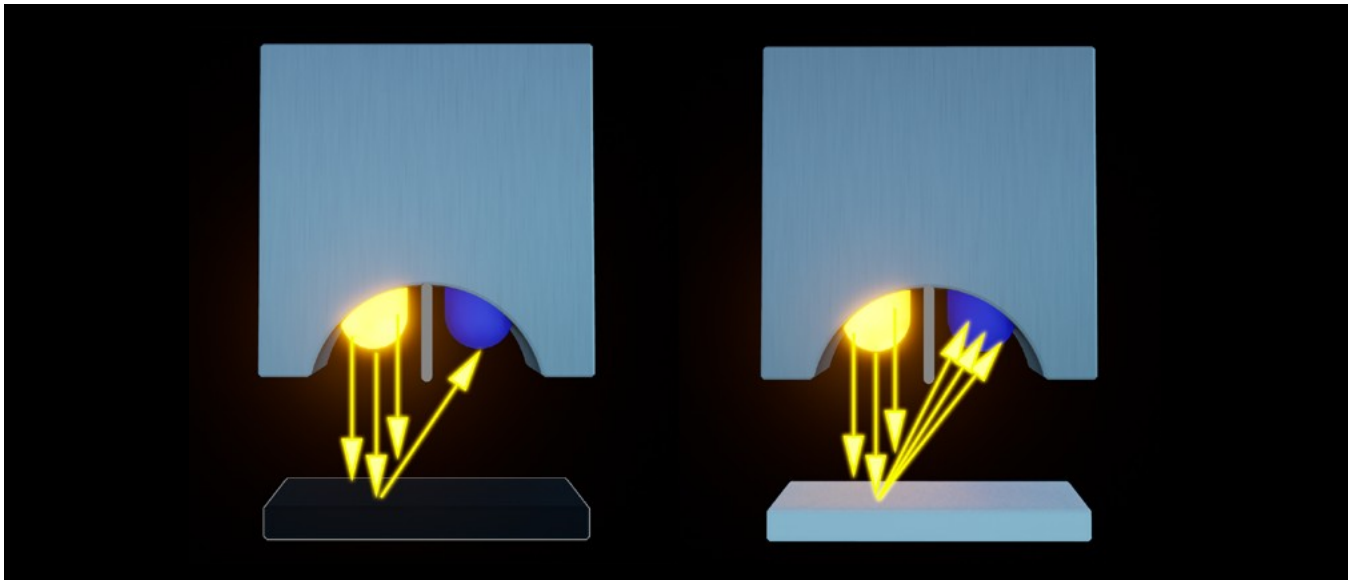
我们会在本活动中使用光传感器，传感器的位置会影响到准确度和编码操作的响应。

反光传感器原理

机器人入门学习包中最常见的光传感器类型是反光传感器。这意味着该传感器配置一个能够发光的LED，同时配备了接收器，以此测量“看见”的光量。

需要注意的是，发射器和接收器是通过障碍物隔开的，发射器发出的光不会直接传入接收器。接收器的主要输入来源就是附近表面的反光。为了确保效果，传感器必须安装在表面附近，并且直接朝向表面。由于暗色材质能够吸收光，接收器获得的返回值会低于亮色表面。传感器将返回“看见”光线的数值，我们可以使用该数值来确定该表面是否为白色地面或者黑色线条。

反光传感器



由于黑色表面吸收了大部分光线，所以接收器返回的值较低。

低值=暗色表面

由于亮色表面反射了大部分光线，所以接收器返回的值较高。高值=亮色表面

图2-3

在添加传感器之前，我们需要考虑到以下情况：

- 传感器距离表面的高度会影响到准确度
- 传感器在机器人前后端的位置会影响到机器人的转向操作，从而影响到可靠性
- 在相扑战斗中被其他机器人推出场外时，传感器的前后方位置会影响到其是否能够正确地做出反应

我们在蓝图视口中添加了一个助手，让你能查看传感器的理想位置。你可以寻找下图这个Cleverlike标志。

图2-4

- 在**视口**中**点击机器人**并选中它（你应该能够在屏幕右侧的**细节**面板和**世界大纲视图**中，看到**BPC_Robot_LineDetection**显示为激活的**Actor**）
- 调整机器人朝向，直到你能看到助手。**专业建议**：在选中机器人时按下**F**键就可以放大机器人。

机器人的躯体应该是平的（不应过高），轮子和滚球应该构成三角形，可以将它当做三轮车的三个触地点，这种车就连小孩都能安全骑行。

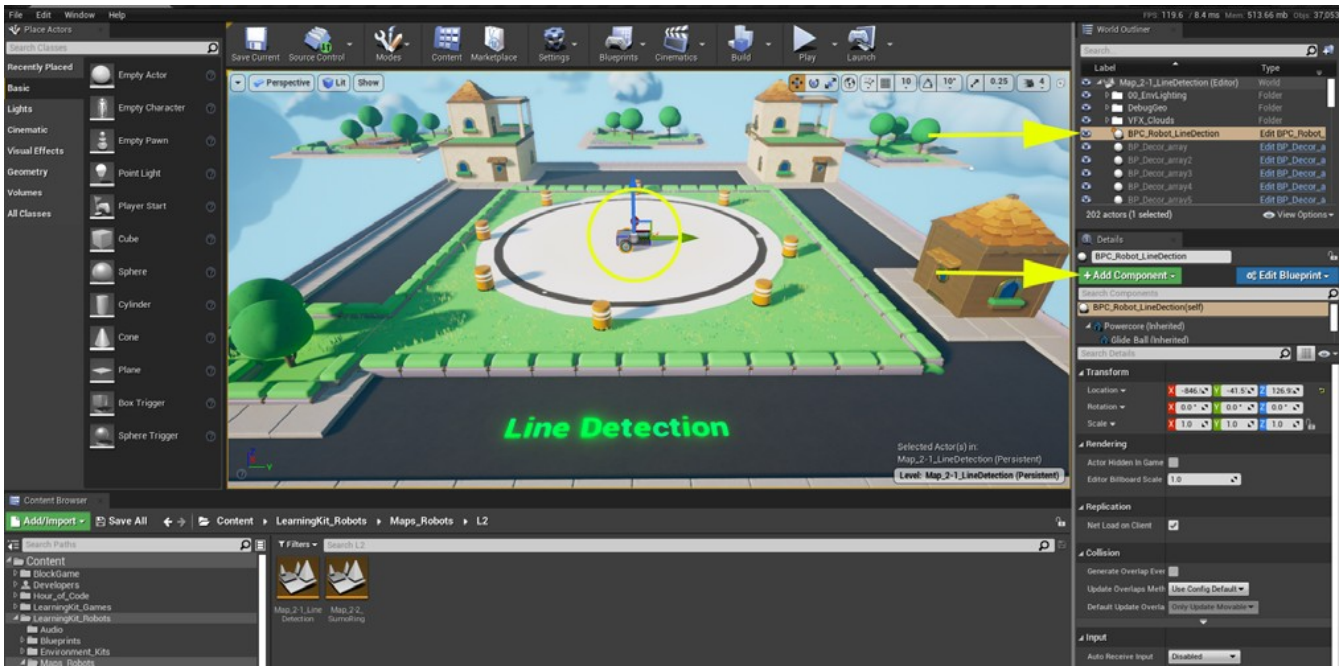


图2-5

添加光传感器

- 在细节面板中：点击**编辑蓝图**，选择**打开蓝图编辑器**（图2-6）

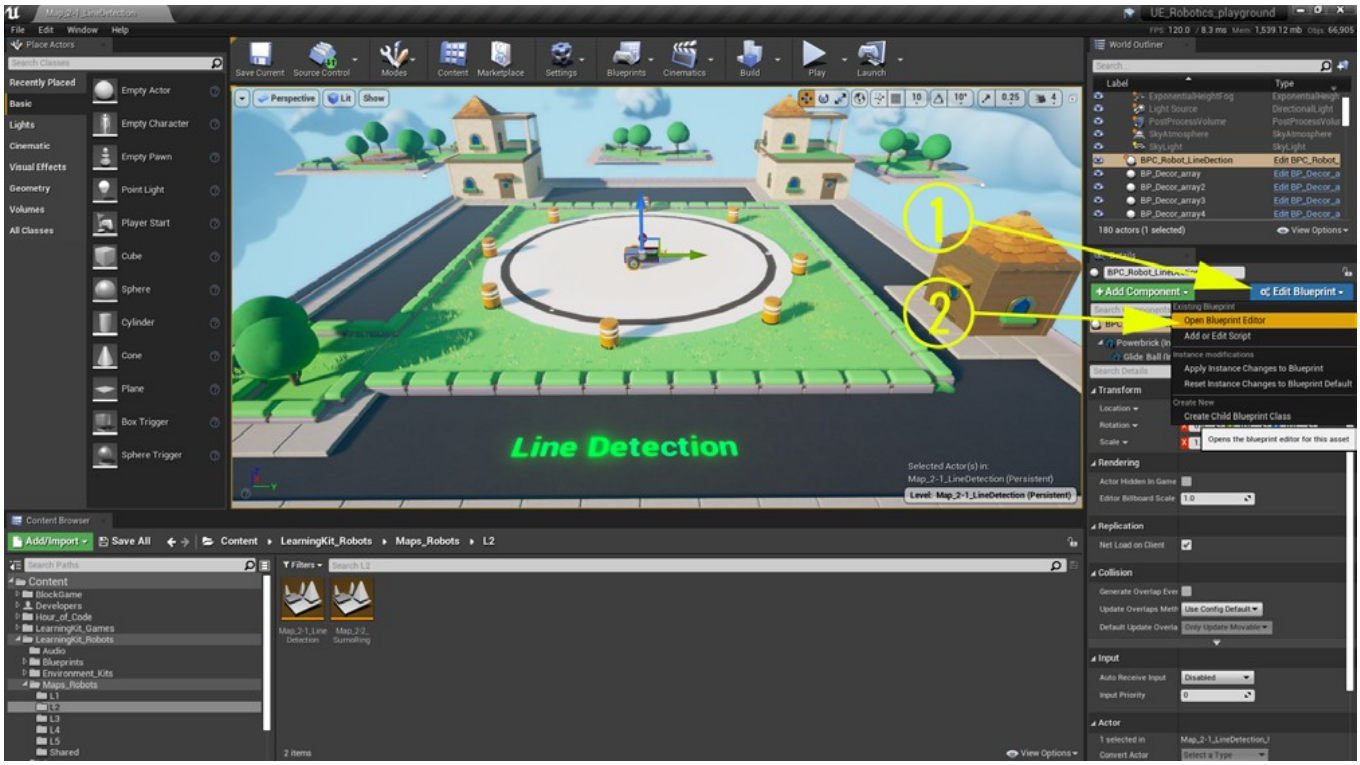


图2-6

- 你会在新打开的虚幻引擎标签页**BPC_Robot_LineDetection**中看到**事件图表**

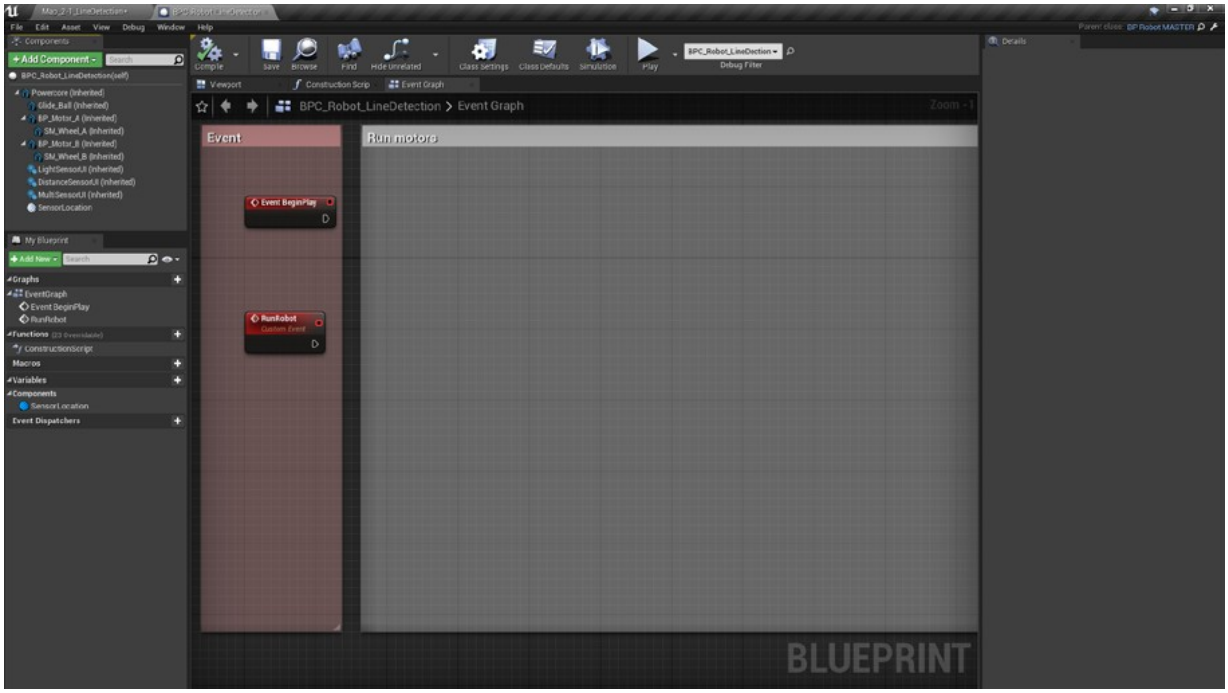


图2-7

- 在本次练习中，你会发现这里没有代码，只有左侧的条件方框里有“事件开始运行”和“运行机器人”节点。右侧的“运行马达”方框里也没有代码。

要为机器人添加光传感器，请遵循以下步骤：

- 在界面左上角的组件面板中：**点击绿色的添加组件**按钮
- 在**搜索栏**中输入**传感器（Sensor）**
- 点击选中**BP Sensor Light**

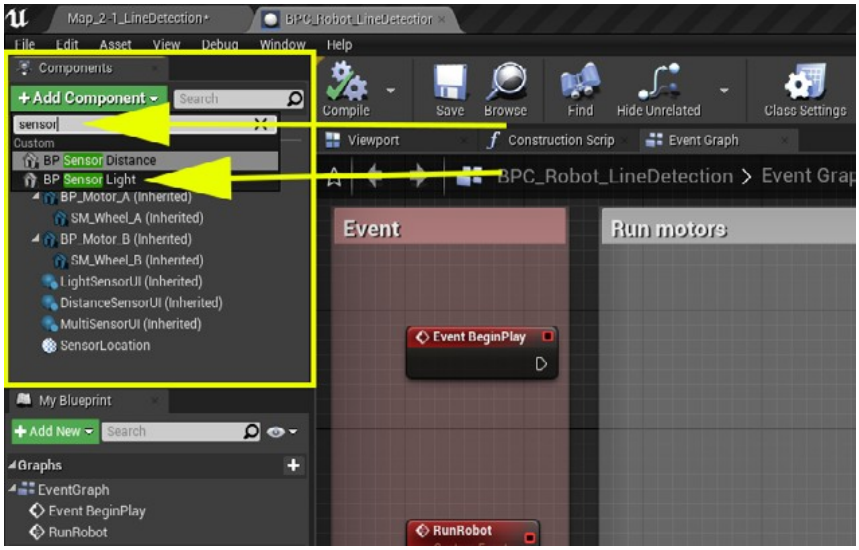
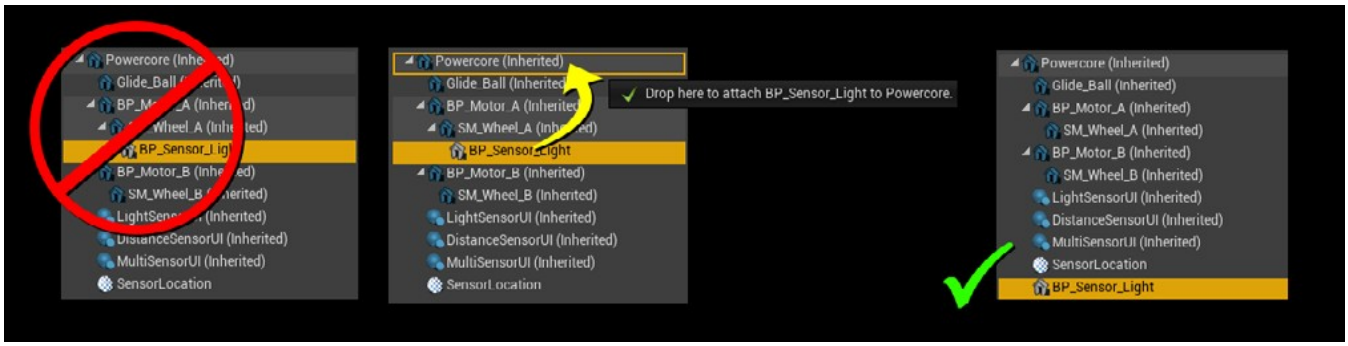


图2-8

- 这样就会将新组件添加到底部的组件列表中
- 如果组件没有出现在**Powercore**下方，那你就是将它添加在其他组件下方了。你需要**点击**并**拖动BP_Sensor_Light**，将它拖到自己依赖的**Powercore**中。
  它随后会出现在组件列表下方。如果位于其他组件下方，就会成为该组件的“子项”，而不是Powercore（机器人）的子项



如果你的传感器显示为子项，就应该将它直接移动到**Powercore**下方。

将**BP_Sensor_Light**拖放到**Powercore**组件上

BP_Sensor_Light应当和其他**Powercore**下的直接组件同级。

图2-9

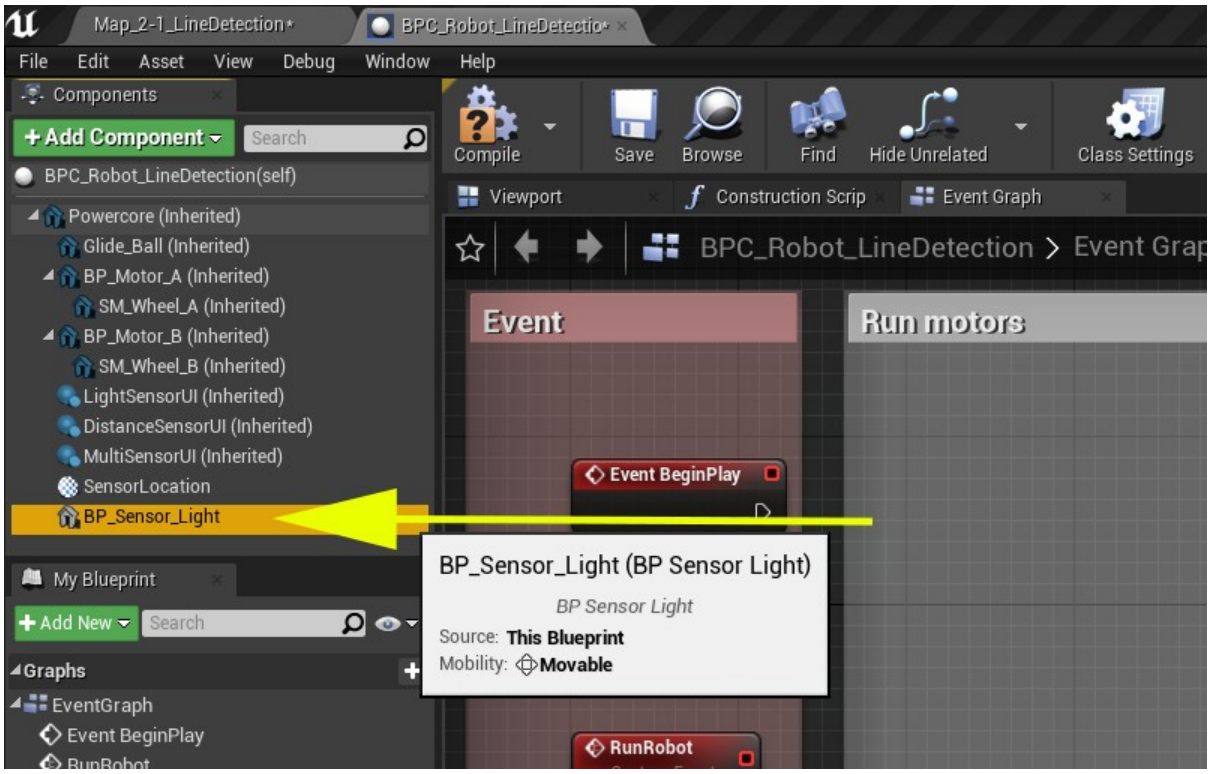


图2-10

- 要查看传感器相对于Powercore的位置，**点击视口**标签页（见下图）。

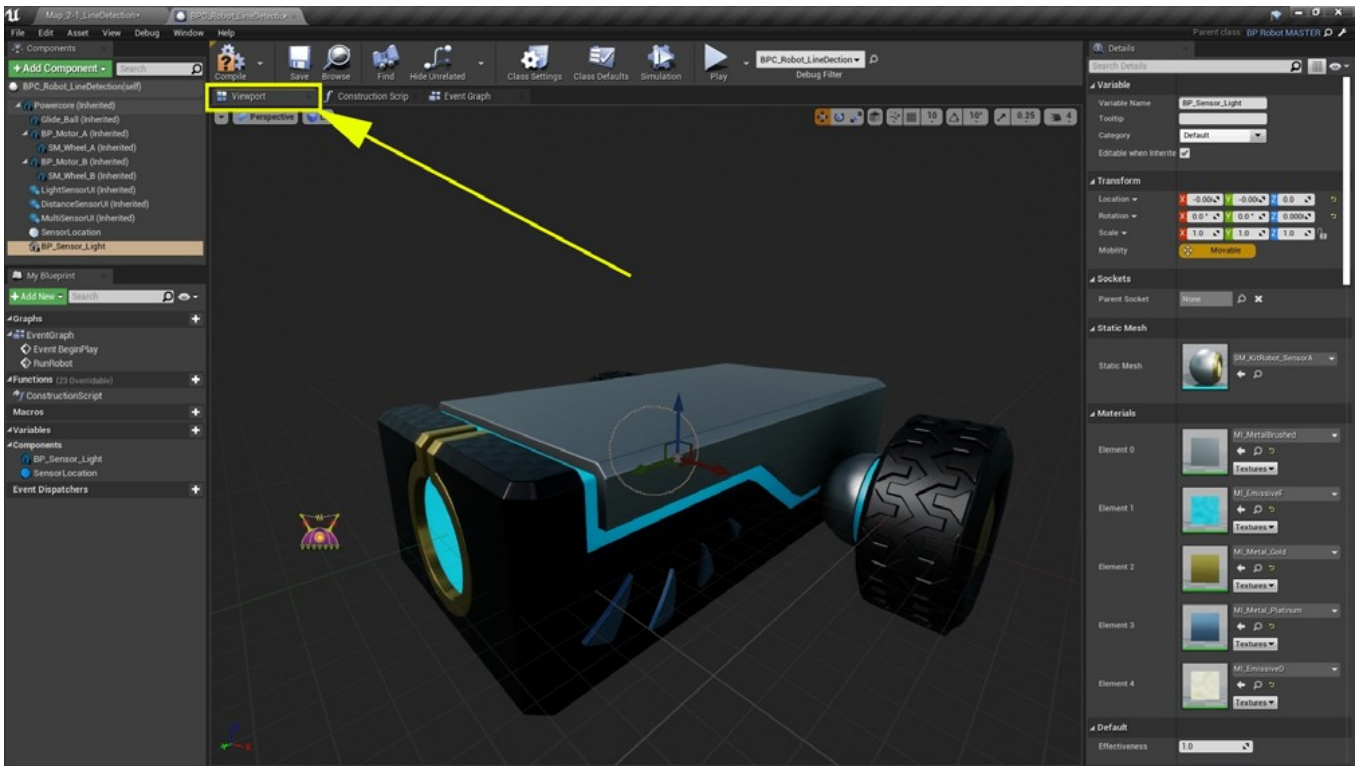


图2-11

- 传感器位于机器人的Powercore内部。选中它后，你会看到机器人内部有一个淡黄色的轮廓
- 我们需要将传感器移动到更合适的地方，让它更高效地发挥作用。
- 为此，我们需要使用视口窗口中的对象移动工具，就在工具栏的右侧
- **点击第一个工具**，使用**选择和平移对象工具（移动工具）**，或者**按下W键**（工具会显示为一个三向箭头），我们可以拖动工具的底部，向上下左右**移动**传感器，直到光传感器覆盖Cleverlike标志。这就是我们的起始位置
- 还是在视口窗口中（工具栏右侧），点击第二个工具，或者**按下E键**，使用**选择与旋转对象工具（旋转工具）**，**旋转它**，使其朝向地面
- 注意，会在右侧细节面板的变换部分中，选中的对象（光传感器）的设置会不断**变化**。你正在使用**移动工具**（W）和**旋转工具**（E）更改**位置**和**旋转**设置。如果你很了解放置对象的理想位置，还可以在方框中**输入值**来定义设置。
- 你的机器人和传感器现在就会像图片中所示：

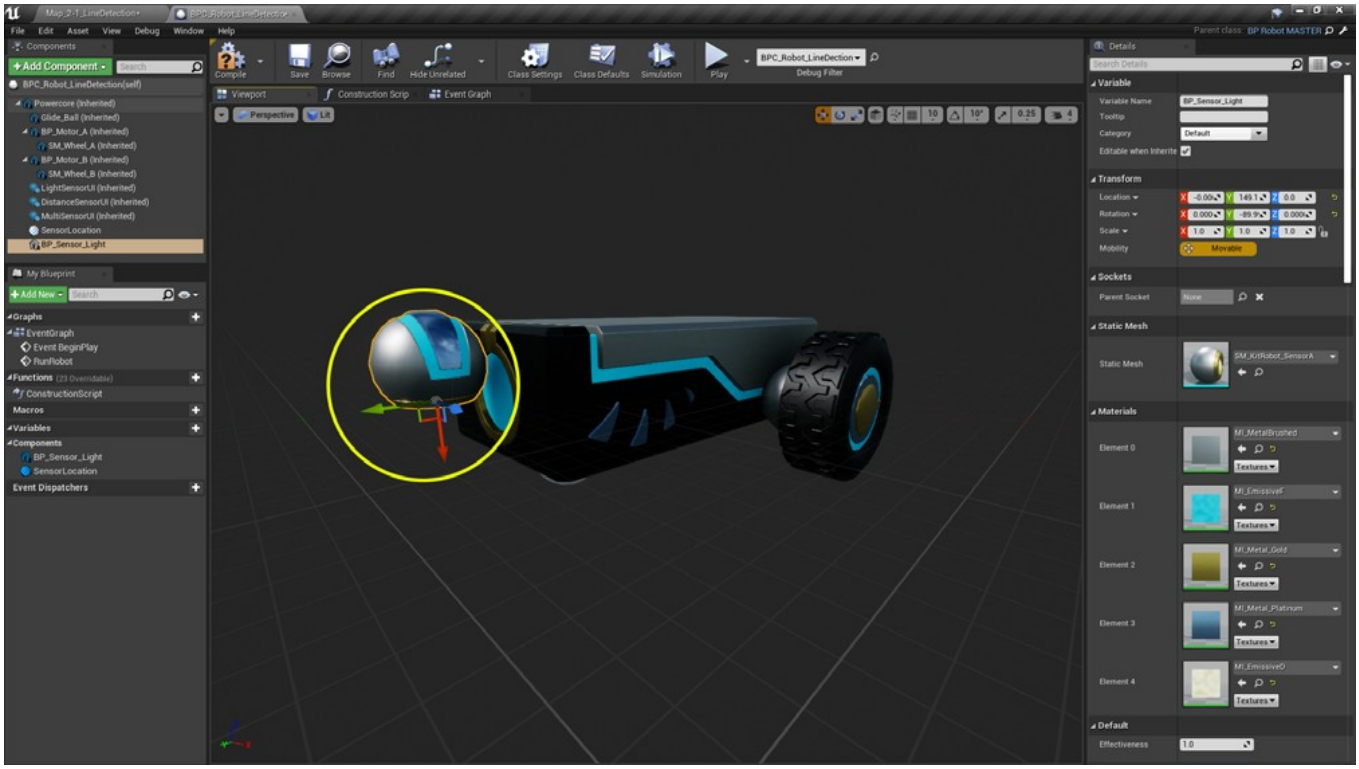


图2-12

- 选中**BP_Sensor_Light**时，注意**右侧的细节面板**。我们之前提到，如果比较方便，你可以在位置栏位中输入数值。

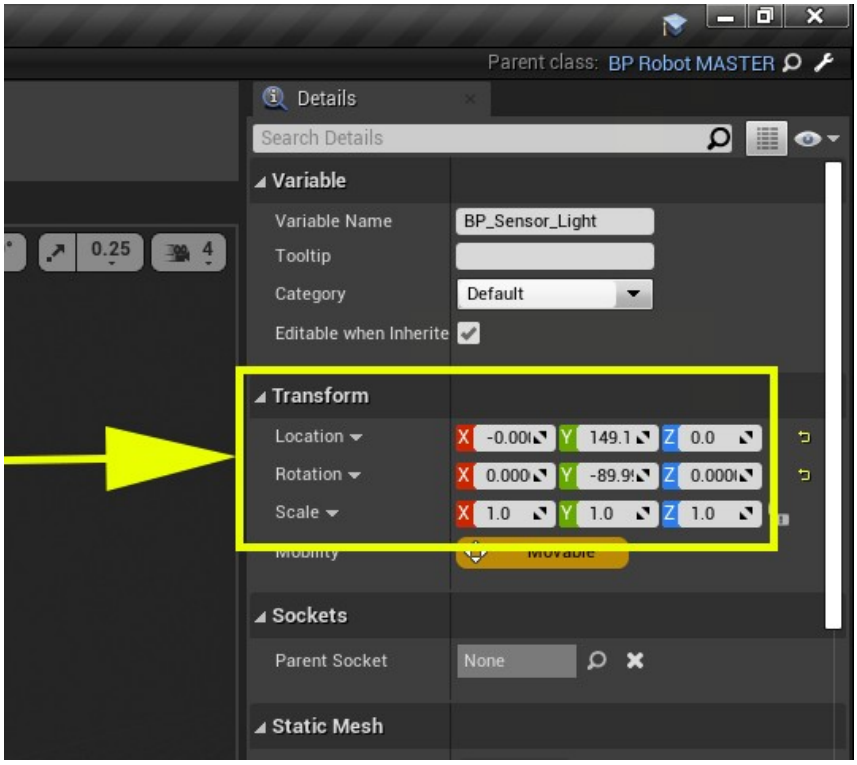


图2-13

- 传感器高度由你决定。要手动调整传感器的离地高度，你可以使用移动工具。用**鼠标点击**并**拖动**垂直方向的箭头，直到你将它摆放在理想的高度上。需要注意的是，在你上下拖动传感器的时候，位置的Z轴值会不断变化。

最后，我们需要将改动编译到机器人的内部代码中。

- 在工具栏的左上角**点击编译**，随后**点击保存**。

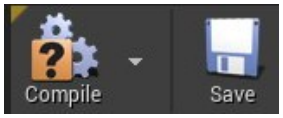


图2-14

- **点击**屏幕顶部的**Map_2-1_LineDetection**标签页，回到关卡中并放大机器人（鼠标**滚动**或按下**F键**）。
- 现在视口就会在机器人前方显示传感器。见图2-15。

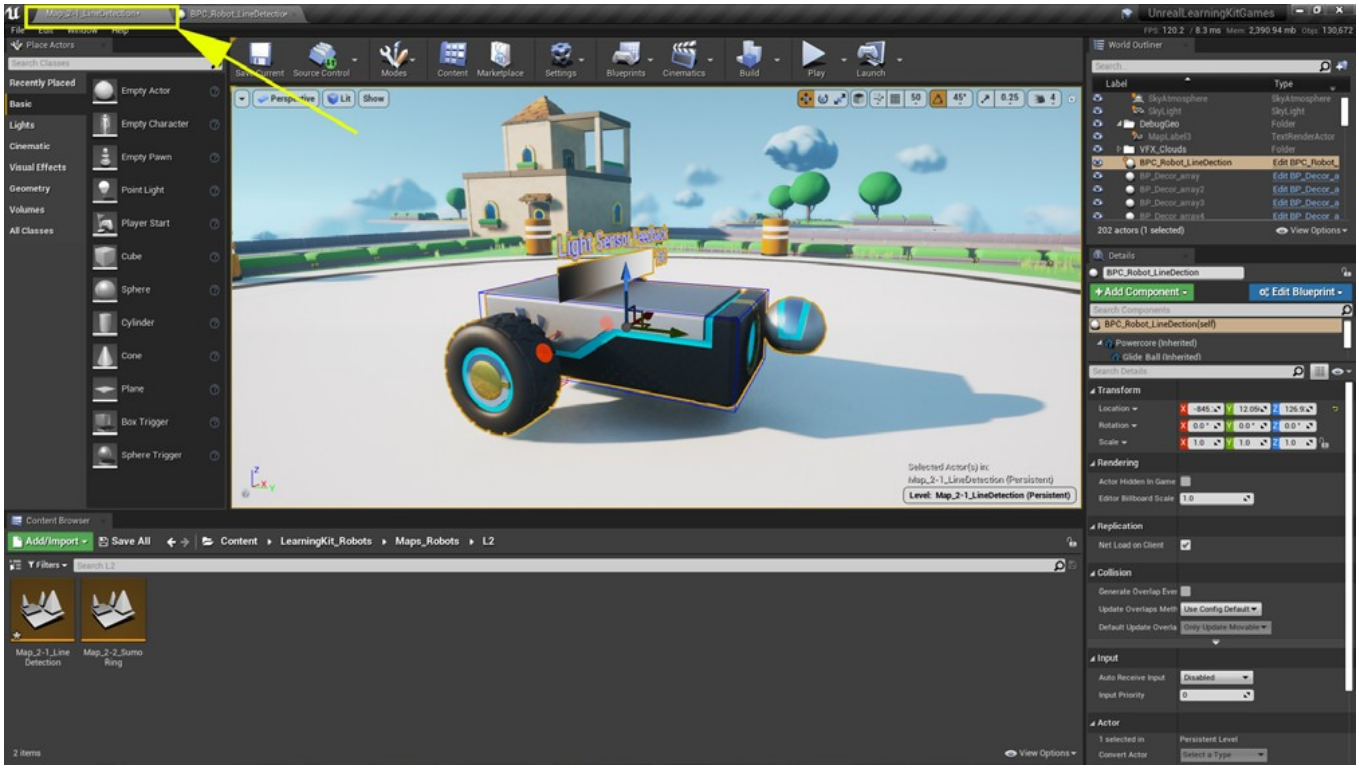


图2-15

练习2：获取传感器输入

传感器的信息

由于机器人必须检测到竞技场地面和出界线之间的差异，我们添加了一个光传感器来检测这种差异。

传感器类型

- 我们会使用光传感器来测量反光。亮色或白色表面反射的光比暗色表面多，因此光传感器会在亮色表面上返回较大的数值，在暗色表面上返回较小的数值。

传感器位置

- 载具行驶的时候，传感器必须朝向地面
- 传感器必须靠近地面，才能获得准确的读数
- 传感器必须位于机器人正前方，才能避免驶出界外

传感器连接到机器人身上的理想位置之后，我们需要为机器人的代码添加命令，使它能够返回输入以进行决策。现在我们要在蓝图编辑器中添加光传感器代码，让机器人能够使用它。

注：*要确保你的学校和班级使用了记事本或者共享文档系统，在后续步骤中记录光传感器的值*

打开蓝图编辑器

- 在关卡**视口**中，**点击机器人**
- 观察**细节面板**（右侧），确保它显示出你正在操作**BPC_Robot_LineDetection**
- 在细节面板中：**点击编辑蓝图**，再**打开蓝图编辑器**（你的屏幕现在会在新激活的虚幻引擎标签页BPC_Robot_LineDetection中显示**事件图表**）

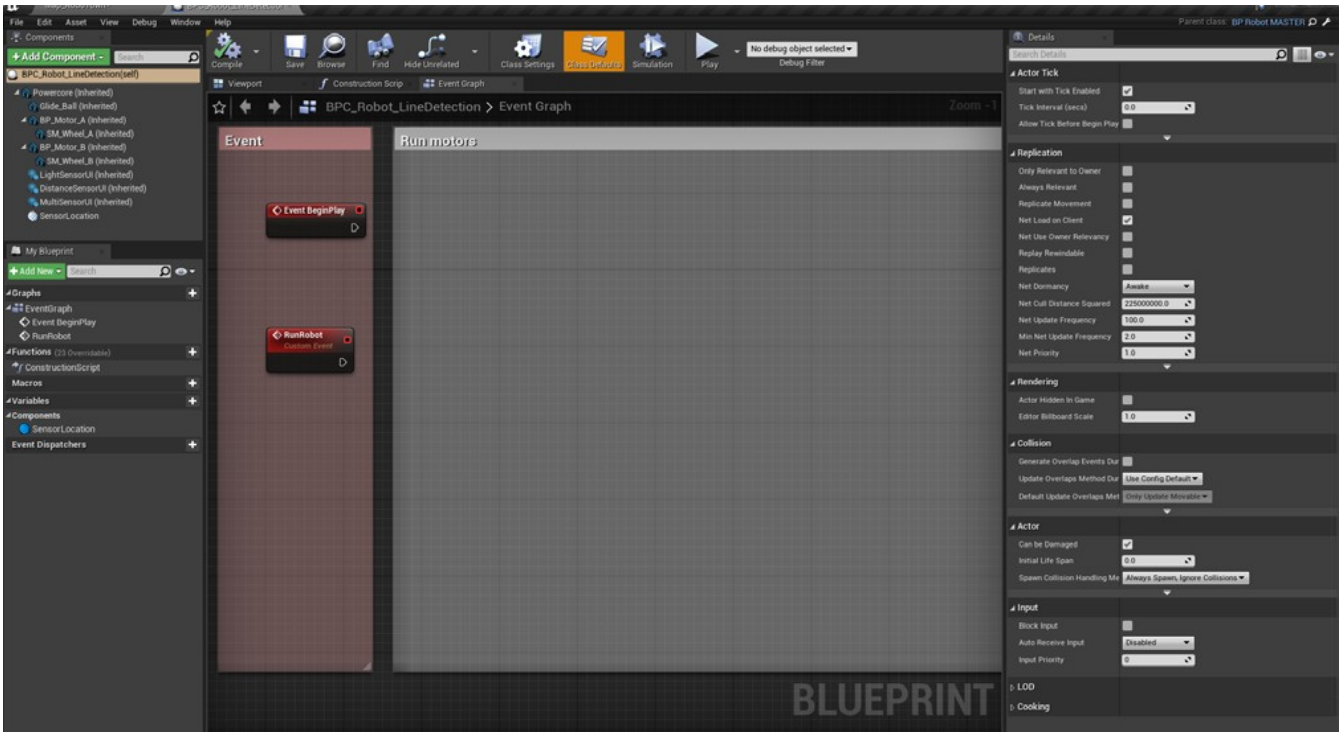


图2-16

- 在练习2中，你会**发现**这里依然没有代码，只有左侧的条件方框里有事件开始运行和运行机器人节点。右侧的“运行马达”方框里也没有代码。
- 从**事件开始运行**节点处**点击输出执行引脚**，**向右侧拖出**，创建一根线条，打开一个弹出窗口，你可以在这里搜索我们需要的函数。
- **输入**“**Run Robot**”，搜索**运行机器人**函数。

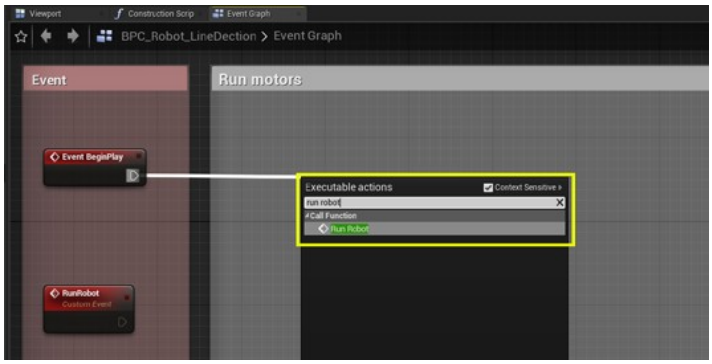


图2-17

- 选中“运行机器人”时，按下回车键或点击它，在图表中添加一个函数。这样会在图表中添加一个“运行机器人”函数，并自动在输出执行引脚和输入执行引脚之间创建线条，将节点连接起来。这样就会在运行游戏时调用“运行机器人”事件（下图）

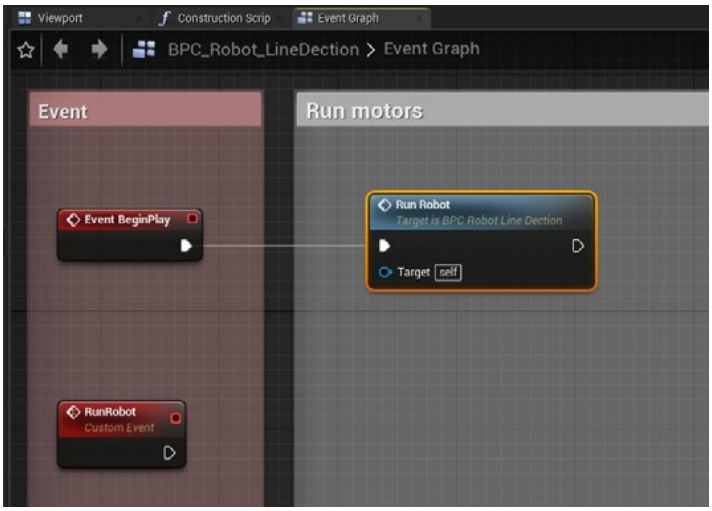
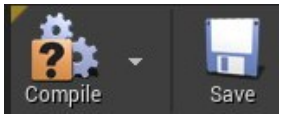


图2-18

*每次添加代码之后都要记得**编译**和**保存**!*



现在我们将传感器连接起来，这样就能在调用“运行机器人”时运行它

- 在**添加组件**面板中，点击**BP_Sensor_Light**，并将它拖入**图表编辑器**

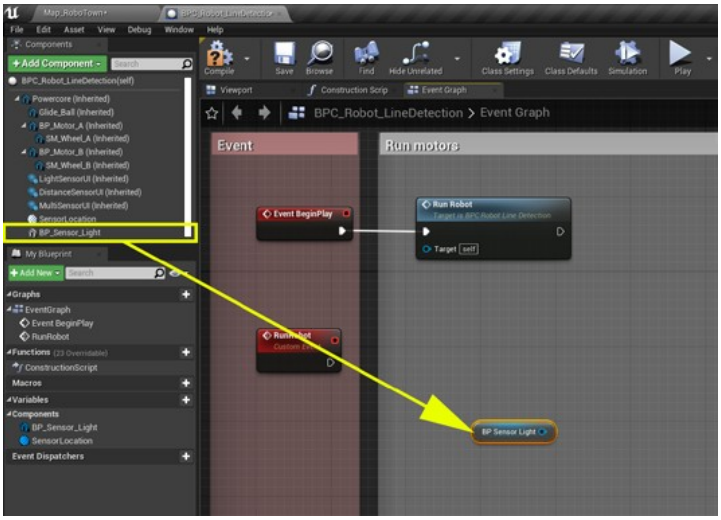


图2-19

为了运行传感器，我们需要将传感器和节点连接。

- 从蓝色的光传感器输出引脚处**向右拖出线条**，并输入在搜索栏中**“运行光传感器”**并**点击它**，将光传感器连接起来

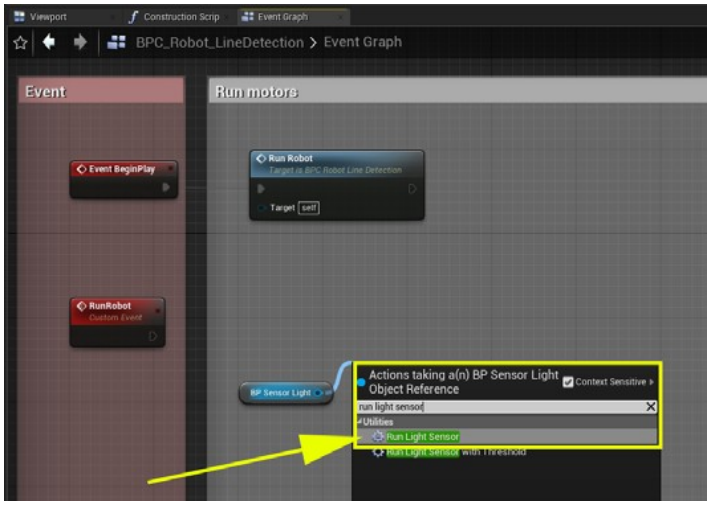


图2-20

- 这样就会在蓝图中放置一个运行光传感器节点，并且它的传感器输入引脚会和BP_Sensor Light节点相连。见图2-21

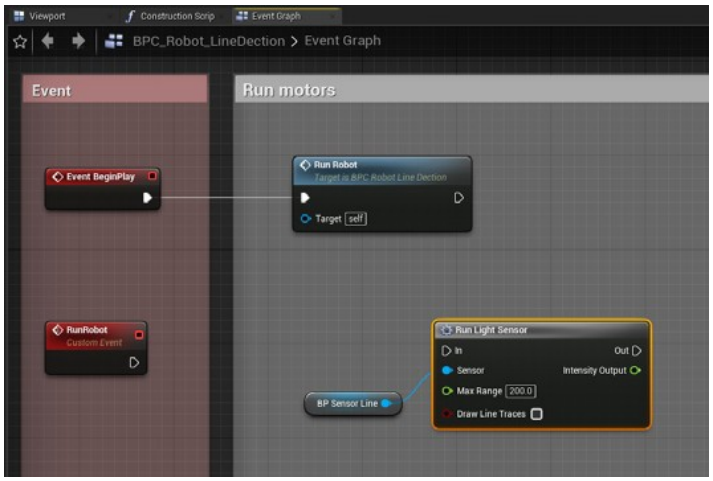


图2-21

最后，我们需要将“运行机器人”条件和“运行光传感器”节点连接。

- 从“运行机器人”的执行**输出引脚**处**拖出线条**，连接到**“运行光传感器”**节点的**输入引脚**中
- 如果你翻了个错误，需要**断开线条连接**，**按住Alt键**并**左键点击线条**或**引脚**即可

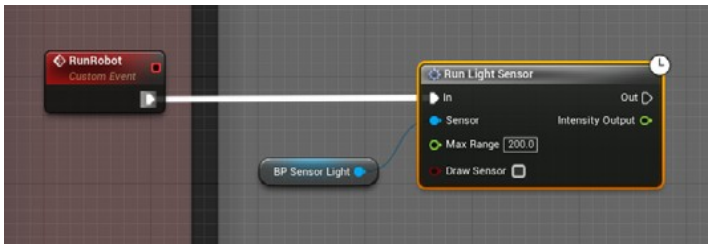


图2-22

该过程将传感器节点放入了代码中，并让它在我们运行游戏时开启。然而，我们当前的代码会在运行命令之后立即停止，而我们无法得知传感器“看到”的内容。为了透过传感器看到内容，我们要创建一个循环。

首先，我们能够看到传感器用于观察的线条。

- 将**绘制传感器设置为开**，从而打开传感器线条

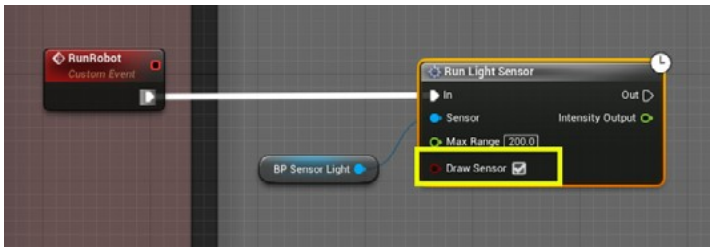


图2-23

- 接下来**点击**你在上一步加入的、蓝色的**“运行机器人”节点**，选中它之后按下**Ctrl+C**进行复制
- 按下**Ctrl+V**进行粘贴
- 将它**拖动**到**运行光传感器**节点的**右侧**，在运行光传感器之后再次调用它
- 最后，从**运行光传感器**的**输出引脚**处**拖出一根线条**，连接到新**运行机器人**节点的**输入引脚**中
- 现在代码就会变成这样：

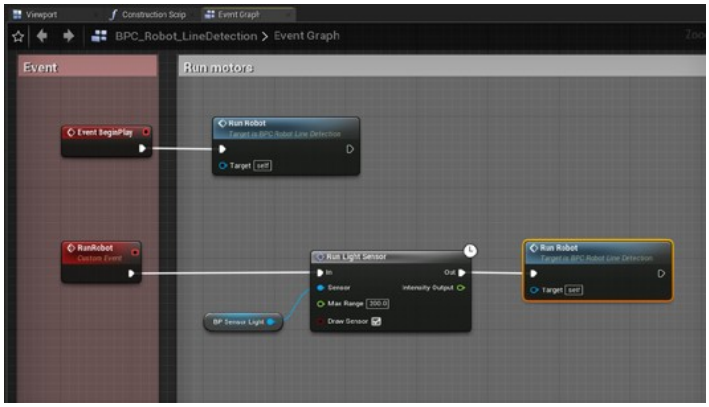
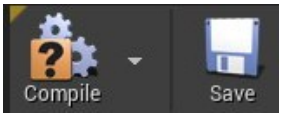


图2-24

- 现在运行游戏时，你就能看到剧烈的数值变化了
- 编译并保存新代码



我们现在打开它，看看效果如何。

- **点击Map_2-1_LineDection**标签页
- 你可以在**视口**中使用**移动工具**，在竞技场里手动移动机器人

运行游戏时，光传感器反馈读数会显示为机器人上方的一个条，表示传感器正在运行。强度读数会显示为红色条，并且覆盖了从深黑色（0）到灰色和白色（100）的渐变着色。这就是反光量的参考。

1. 如果它看到的光线**较少**，返回值**较低**，就会显示更多**红色**条。
2. 如果它看到的光线**较多**，返回值**较高**，就会显示更多**灰白渐变部分**。

- **观察你看到的值**，随后**停止**运行工程，必要时可以**记录竞技场的值。**
- 重复不断地**运行**、**观察**和**停止**，查看返回的值
- 在**视口**中使用**移动工具**，在表面上手动移动机器人，留意竞技场内部的强度读数/数值变化

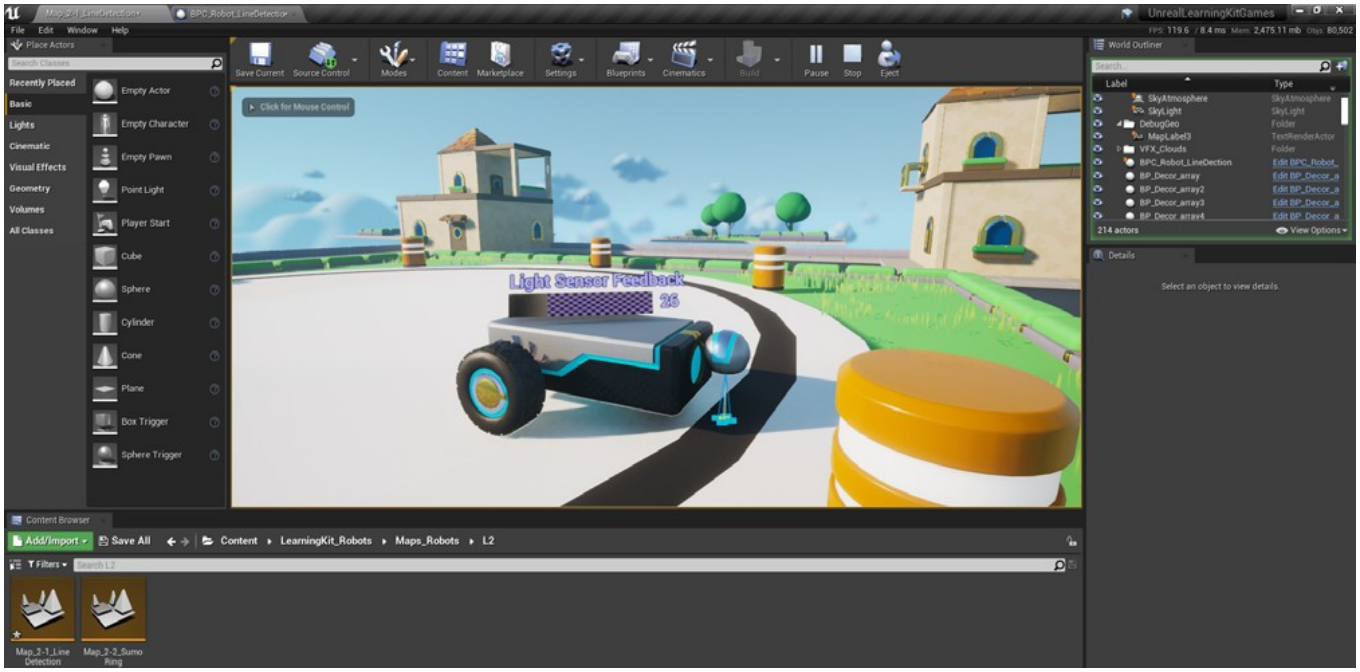


图2-25

- **记录**传感器看到白色竞技场地面时的数值，然后移动机器人，**记录**看到出界线时的数值
- 必要时重复运行、观察和停止，查看返回的值
- 排查错误，确保数值在0（暗）到100（亮）的范围内具有足够的分布差异，能够稳定地反映出竞技场和出界线的区别
- 如果需要，你可以调整传感器的垂直位置使其贴近地面，或者移动至高处以准确地查看表面值

练习3——确定所需的阈值

- 什么是阈值？阈值是一个单一的数值，能够告知机器人的条件语句何时执行特定的操作。条件会询问传感器是否看到了一个大于或等于（>=）阈值数的值，从而触发一个操作。如果传感器看到的值小于（<）阈值数，则会触发另一个操作。因此，阈值就是我们的“分界线”，能够让机器人判断自己是否位于场内，还是已经超出了黑色的出界线
- **参考笔记**，其中记录了上一步中光传感器提供的竞技场和出界线值
- 使用该公式确定阈值：**((高值 – 低值) / 2) + 低值 = 阈值**（可浏览指南末尾的资源部分，获取阈值计算表）
- 例如，如果高值（竞技场）=90，低值（出界线）=10，公式就写作 **（（90-10）/2）+10=50**
- **观察并记录**阈值

练习4：开始条件语句（向前驱动）

在本部分中，我们会添加代码，在线条传感器返回一个高于阈值的值时，让机器人向前移动。

打开蓝图编辑器

- 选中机器人——在**视口**中**点击机器人**（右侧的细节面板中会显示**BPC_Robot_LineDetection**为活跃**Actor**）
- 在细节面板中：**点击编辑蓝图**，随后**打开蓝图编辑器**（你的屏幕现在会在新激活的虚幻引擎标签页BPC_Robot_LineDetection中显示**事件图表**）

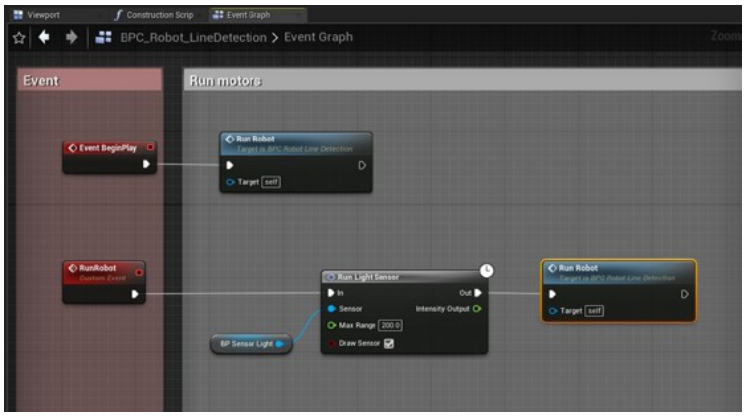


图2-26

命令机器人前进

当机器人位于竞技场内时，我们需要它能够行驶（运行马达），接下来就要添加这部分代码。

- **点击并拖出运行光传感器**节点的**绿色强度输出引脚**，新建一根线条
- **输入**“<=”
- 并选择**浮点**<=**浮点**。

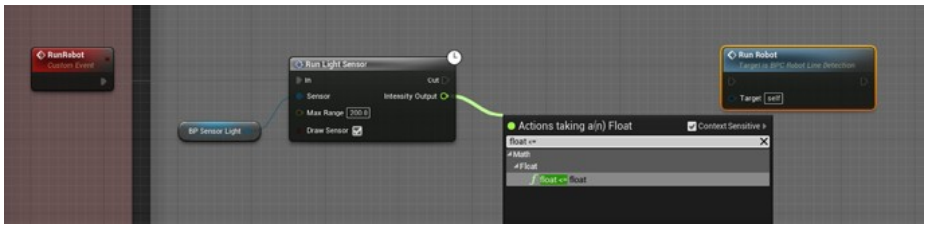


图2-27

- 接下来**点击并拖出运行光传感器**节点的**白色输出引脚**
- **输入**“**if**”（搜索“if”）并添加**分支**节点。（在虚幻引擎中，分支函数等同于其他编程程序中的if函数）

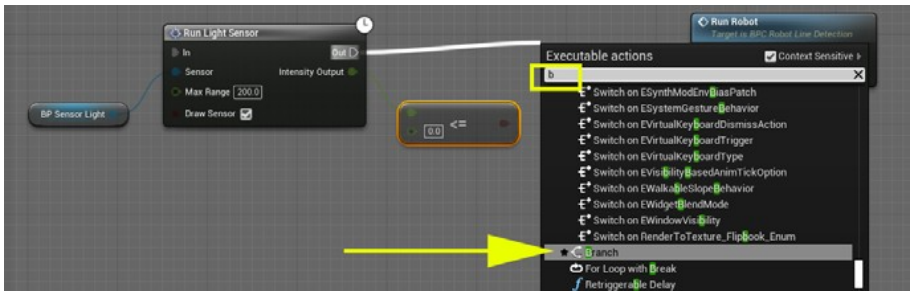


图2-28

- **分支**会自动连接在**运行光传感器**和**运行机器人**节点之间，代码流程会判断条件是否为**True**。这并**不是**我们想要的效果
- **按住Ctrl并点击True**引脚，将线条断开，并拖入到**False**引脚中

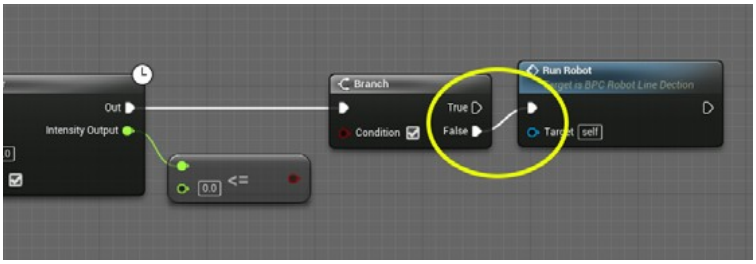


图2-29

现在我们需要设置条件。

- 为此，我们要**点击并拖动<=节点**的**红色引脚**，将它和**分支**的**红色引脚**连接
- 然后将<=方框里的值设置为计算好的阈值数值。我们这里的阈值是50

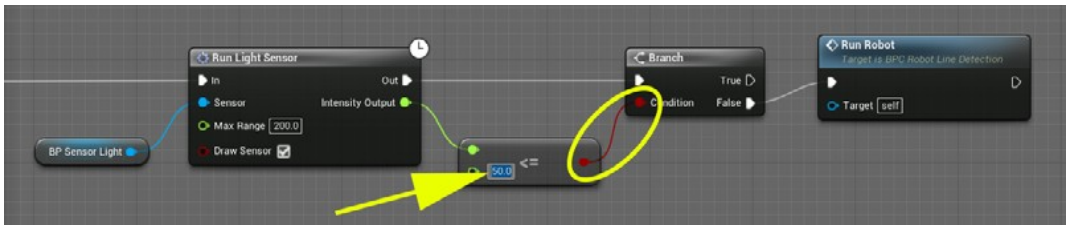


图2-30

接下来我们要编写代码，让机器人能够前进

- 我们需要在**分支**和**运行机器人**节点之间留出空间。选择**运行机器人**节点，将它**移动**到**右侧**
- 现在我们要添加代码，让马达能够转动轮子。
- **点击并拖动分支节点**的**False引脚**
- **输入“Sequence”**搜索，随后**选择它**或**按下回车键，**添加一个**序列**节点

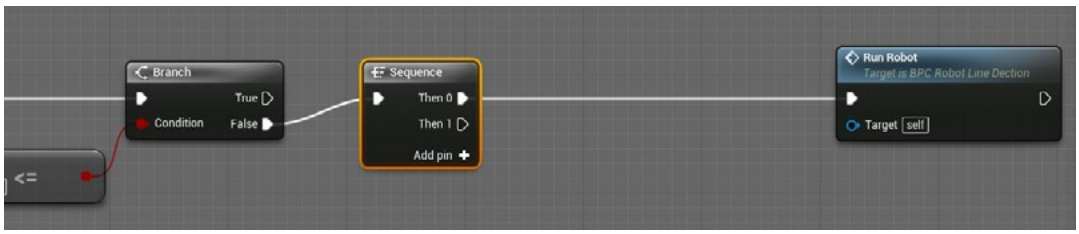


图2-31

什么是序列节点？为什么要使用它？

序列节点会执行一系列命令，从顶部输出引脚开始，到底部输出引脚结束。简而言之，它会按顺序执行从“**Then 0**”到“**Then #**”引脚的代码。

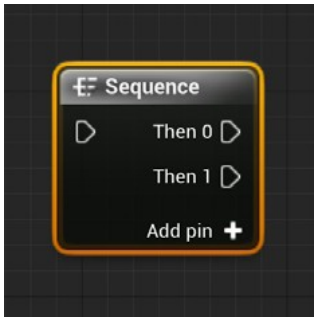


图2-32

我们需要从**Then 0**和**Then 1**引脚处运行马达

- **点击**并从每个**Then**引脚并从中**拖出**
- **搜索**“**运行马达**”，为每个**Then**引脚添加一个**运行马达**节点

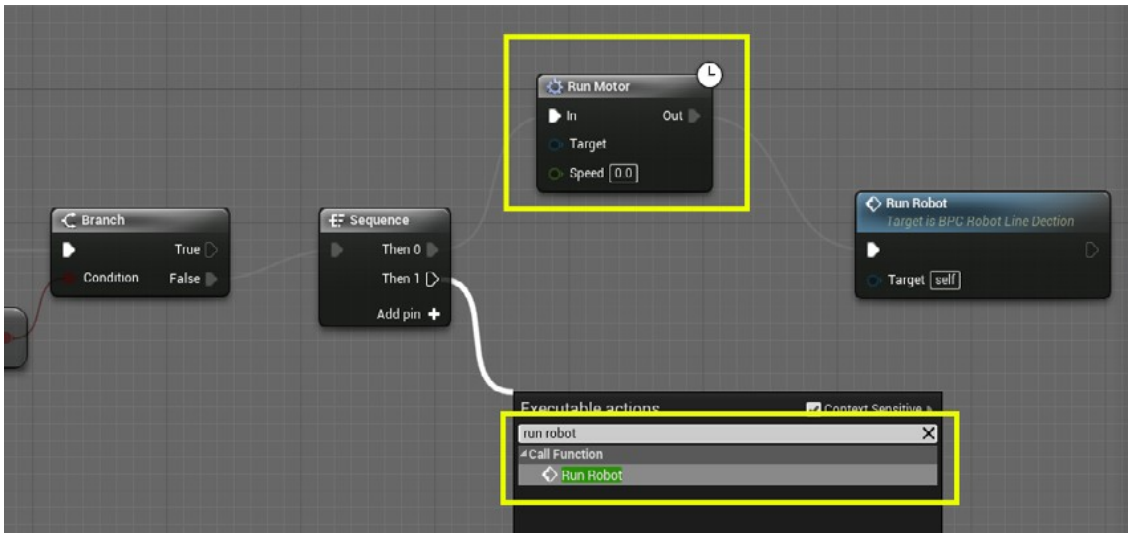


图2-33

确保**输出节点**和**运行机器人节点**相连。

- **点击**每个**输出引脚**，**拖出白色线条**，与**运行机器人节点的输入引脚**连接

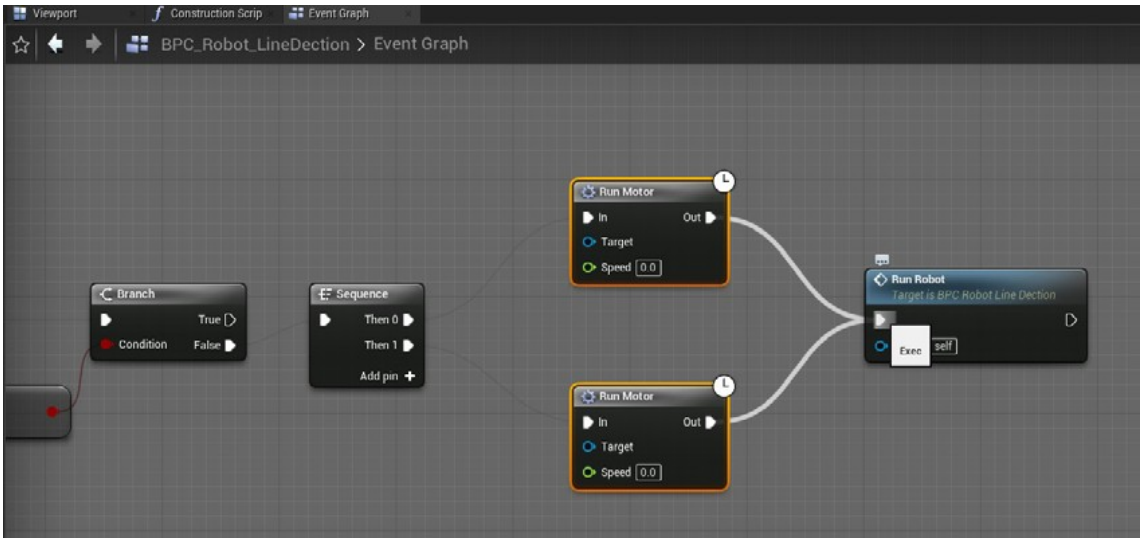


图2-34

接下来我们要为代码设置目标马达，从而设置运行马达的速度和方向。

- 在**组件面板**中，点击**BP_Motor_A**，将它拖动到顶部（上方）**运行马达**节点的**蓝色目标引脚**处

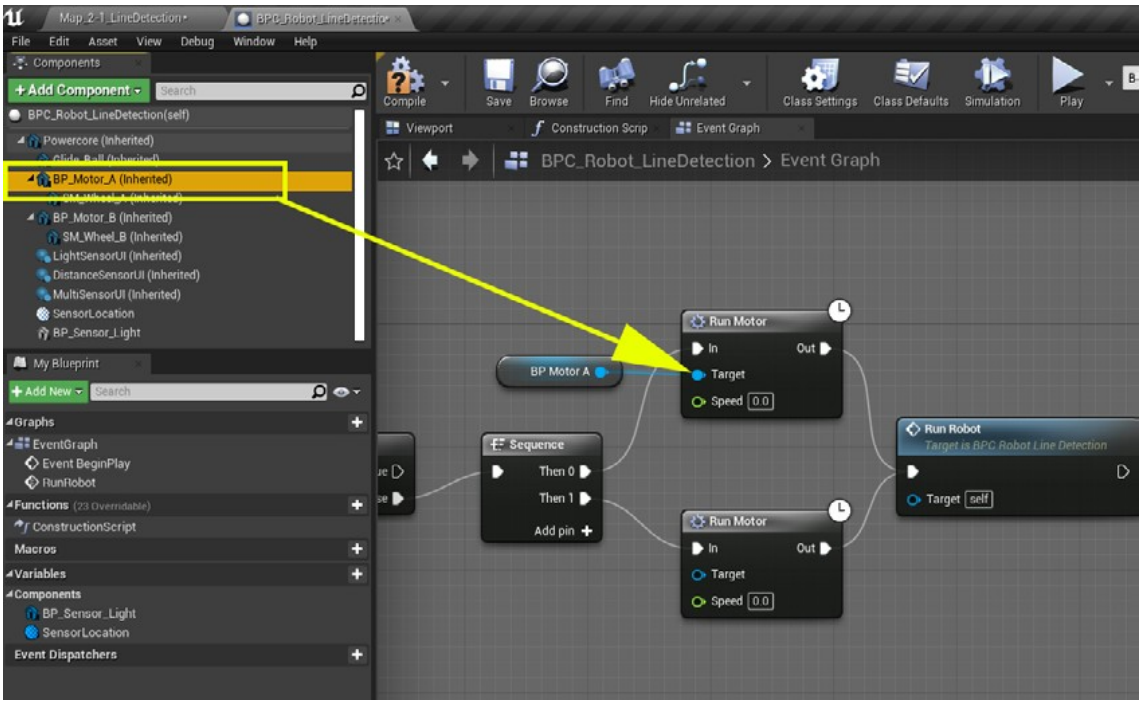


图2-35

- 要设置第二个马达，在**组件面板**中，点击**BP_Motor_B**
  将它拖动到底部（下方）**运行马达**节点的**蓝色目标引脚**处

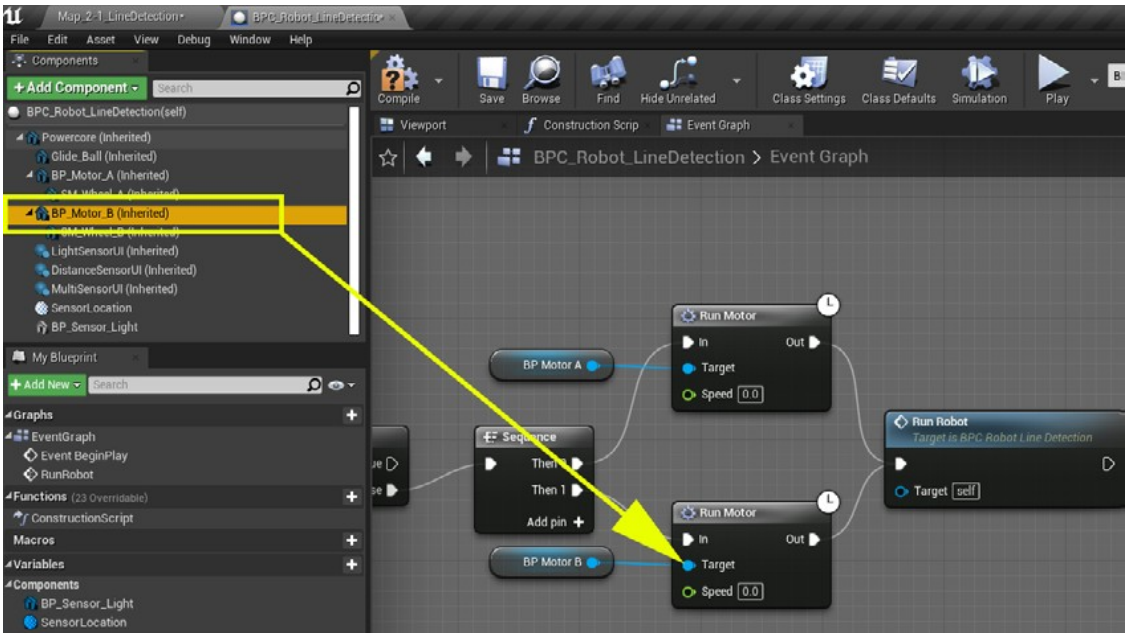


图2-36

现在我们只需要设置运行马达节点的速度即可。

记住，**BP_Motor_B**是反向的，因此必须将速度设置为负值。

- **点击速度数值框**，**输入**你所选择的**数值**
- 将**BP_Motor_A**的速度设置为**10**，将**BP_Motor_B**设置为**-10**（注：速度一栏允许的值范围为-100到100）
- **编译**并**保存**代码

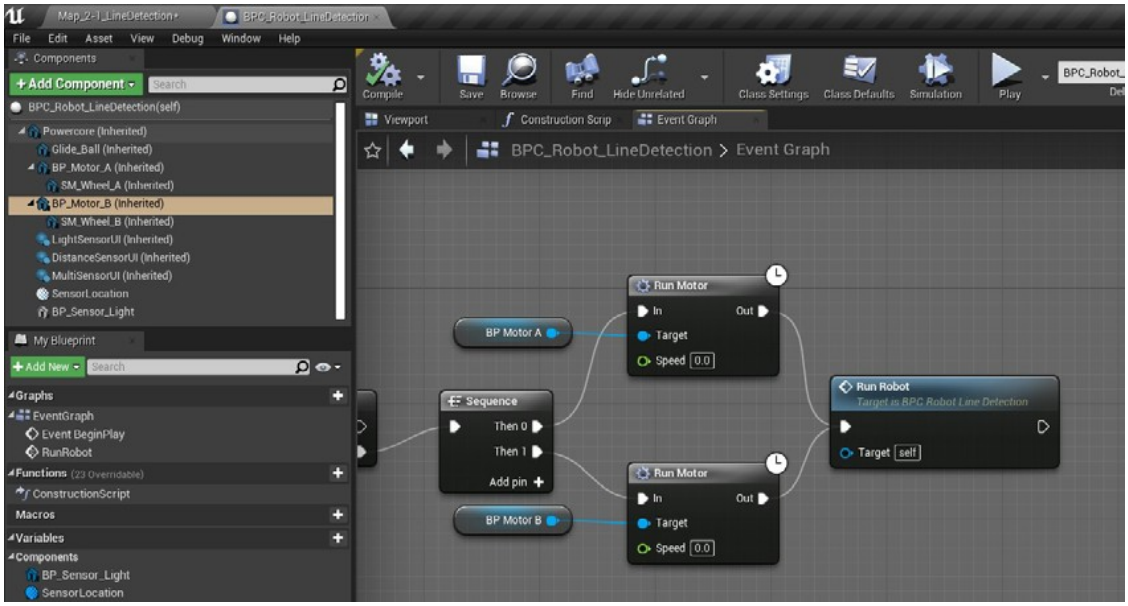


图2-37

完成之后，代码应该是这个样子（见图2-38）

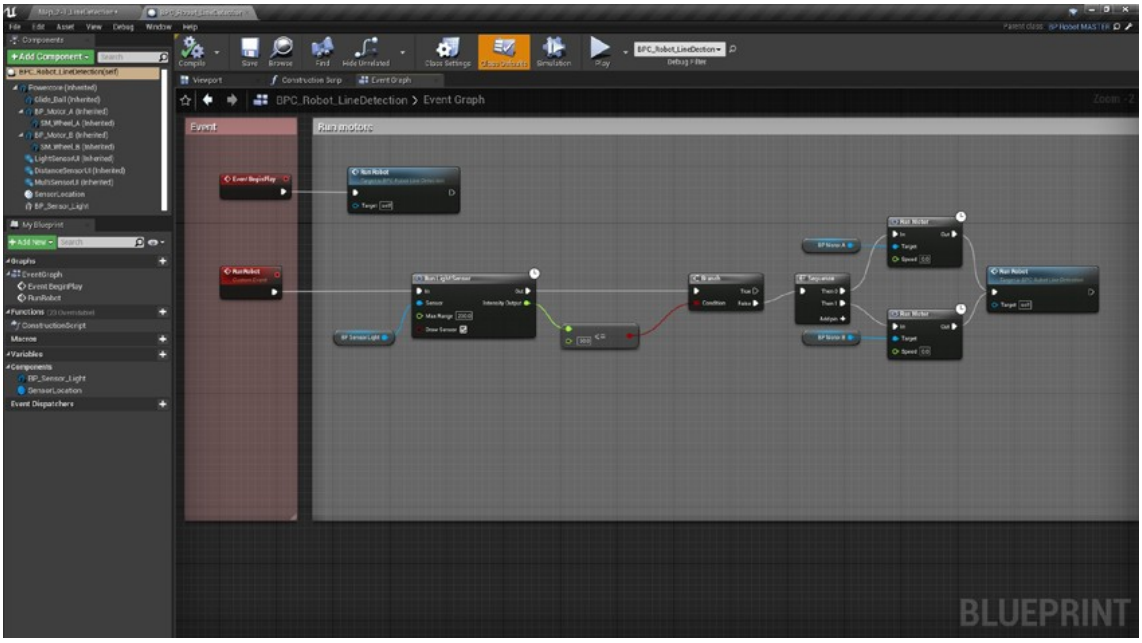


图2-38

回到**Map_2-1_LineDetection**标签页中，确保机器人位于竞技场内。如果你需要移动机器人，可以点击机器人并使用**移动**工具，将它放置在竞技场内，并确保它高于地面

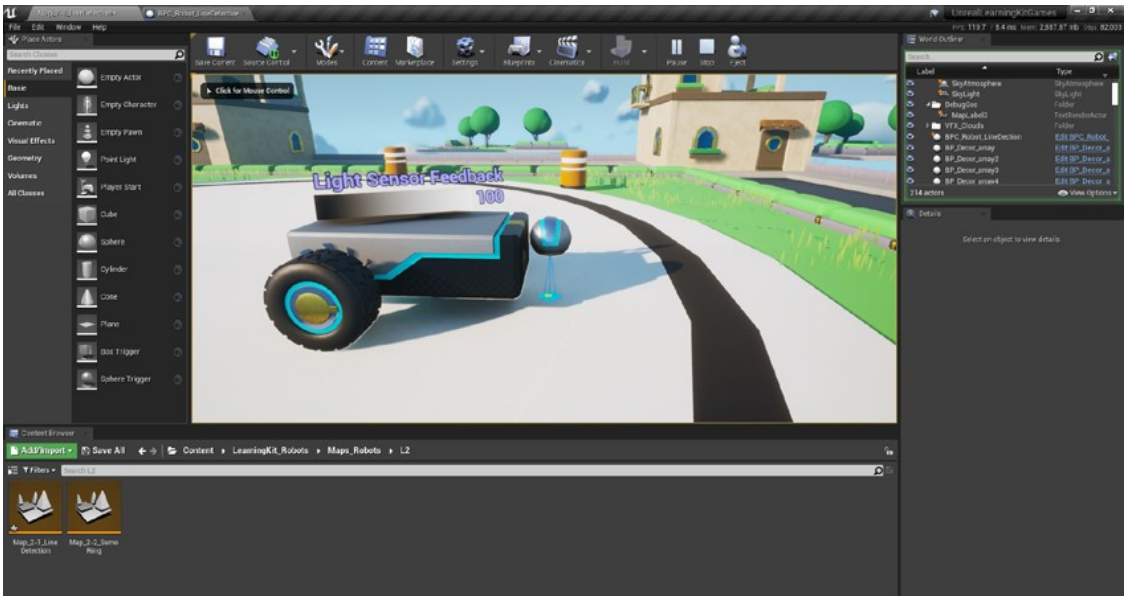


图2-39

- **运行**机器人游戏
- 现在机器人就可以向前移动并靠近线条了
- 然后呢?

问题排查

按下运行键之后，机器人就会向前方行驶。

- 如果你的机器人一动不动，请确保你遵循所需的所有步骤，在蓝图中将机器人节点全部连接起来
- 确保你编译并保存了蓝图
- 如果机器人向后移动或发生转向，请确保马达A和马达B的速度值设置无误

练习5：第二条件

停止和转向

在本练习中，我们会为**True分支**添加代码，使机器人在触及边界线时停下。

传感器输入响应

当机器人能够读取传感器的输入值时，它就可以根据传感器的反馈执行特定的操作。我们希望它能持续判断，如果能看到竞技场地面，则继续向前行驶；当它看见出界线时，就应该停下并转回竞技场内。

活动：我们现在要在蓝图节点中使用条件语句（分支节点）
根据传感器输入执行操作。如果传感器输入小于等于阈值，则表明它看到了出界线。如果输入值大于阈值，则表明看到了竞技场地面。

- **点击Map_2-1_LineDection**标签页
  第1步：我们首先要编辑机器人的操作命令：
- 在**视口**中，**点击机器人**
  第2步：我们需要编辑代码，所以打开蓝图：
- **点击细节面板**（一定要确保你编辑的项目为**BPC_Robot_LineDetection**）
- **点击编辑蓝图**
- 在**下拉菜单**中，点击打开蓝图编辑器

出界操作

第3步：接下来，我们要更新条件语句，使其在检测到出界线时发送新的马达控制操作：

- 我们要使用图2-38中的代码，所以检查一下，确保当前代码与其相同。为了方便读取代码，我们可以添加注释，所以现在就来添加吧。
- **点击**并选择**序列节点**，然后按住Ctrl键，同时点击**后方的所有节点**（选中的节点会带有橙色边框）
- 按下**C键**创建**注释框**
- 在注释框中**输入**文字，将这组代码命名为“**Drive Robot Forward**”
- 随后**点击注释框并向下拖动**，为后续添加的新节点腾出空间

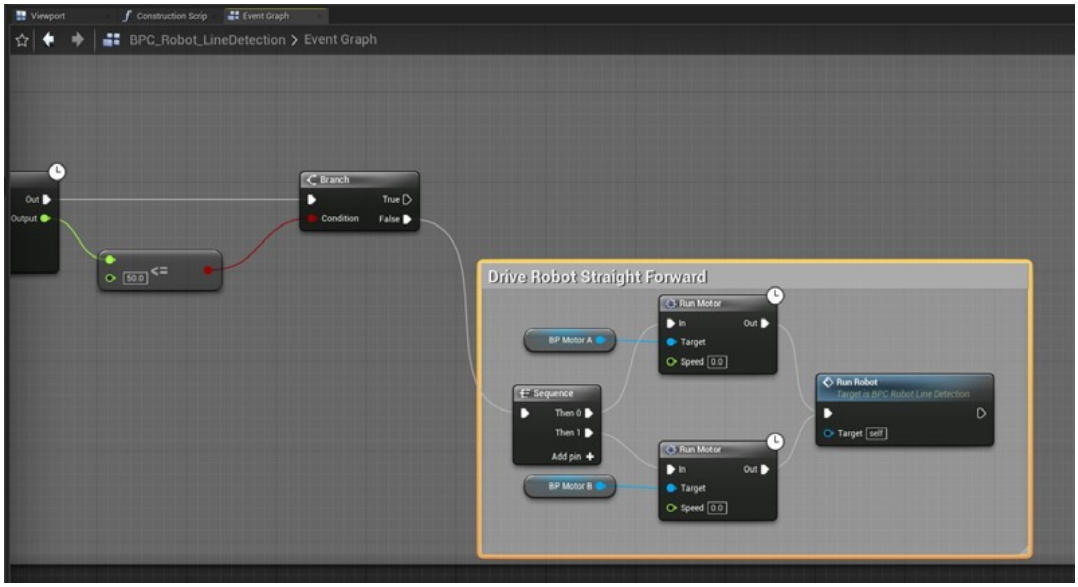


图2-40

在机器人看见出界线时停止移动

为了让机器人停下，我们需要让两个马达同时停止运行。我们已经拥有了部分需要的代码，所以先复制这一部分吧。

- **选中BP_Motor_A和BP_Moter_B节点**（点击选中额外节点时一定要按住Ctrl键）
- 按下**Ctrl+W**进行复制
- 复制的节点会出现在鼠标悬停的位置
- **点击并拖动节点**，将复制节点移动到**Drive Robot Forward注释框上方**

（见图2-41）现在我们要添加两个**停止马达**节点。

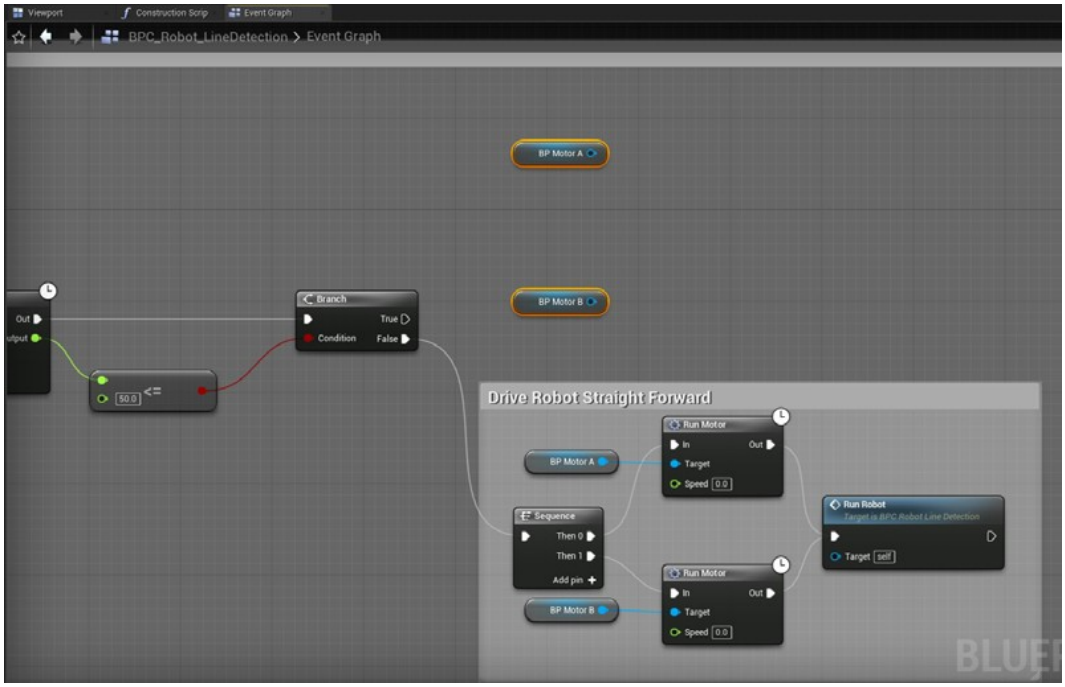


图2-41

- **点击分支**的**True引脚**并**向右拖出**
- **输入**（搜索）“**停止马达**”
- **选择停止马达节点**

这样就会在事件图表中放置一个停止马达节点。

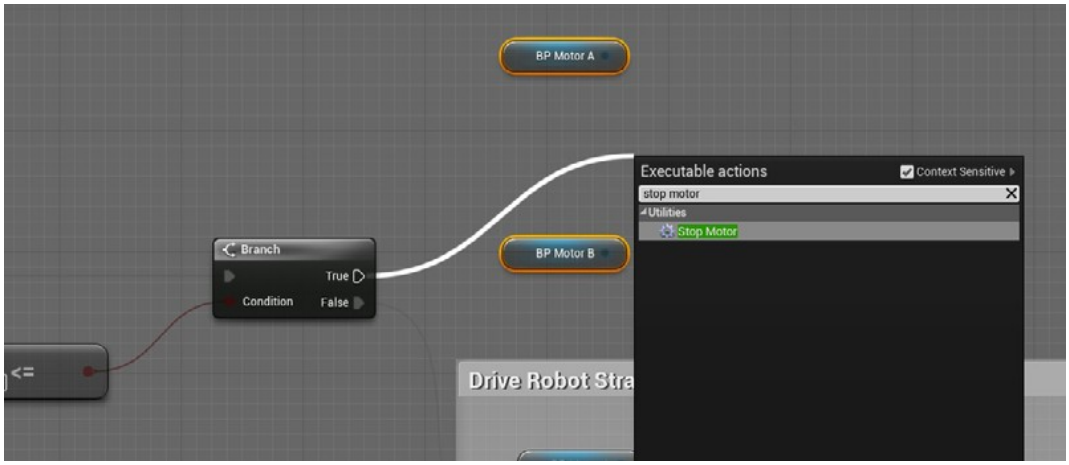


图2-42

要复制**停止马达**节点：

- 选择（**点击**）**停止马达**节点（会显示橙色边框）
- **右键点击**它，在弹出菜单中选择**复制**（Ctrl+W）
- 现在你就有两个**停止马达**节点了
- 如下方示例所示（图2-43），**点击复制的节点**并**拖动到右侧。**要将它们连接起来，**点击第一个节点的白色输出引脚**，**拖出线条**并连接至**第二个节点的输入引脚**，如下图所示（图2-43）

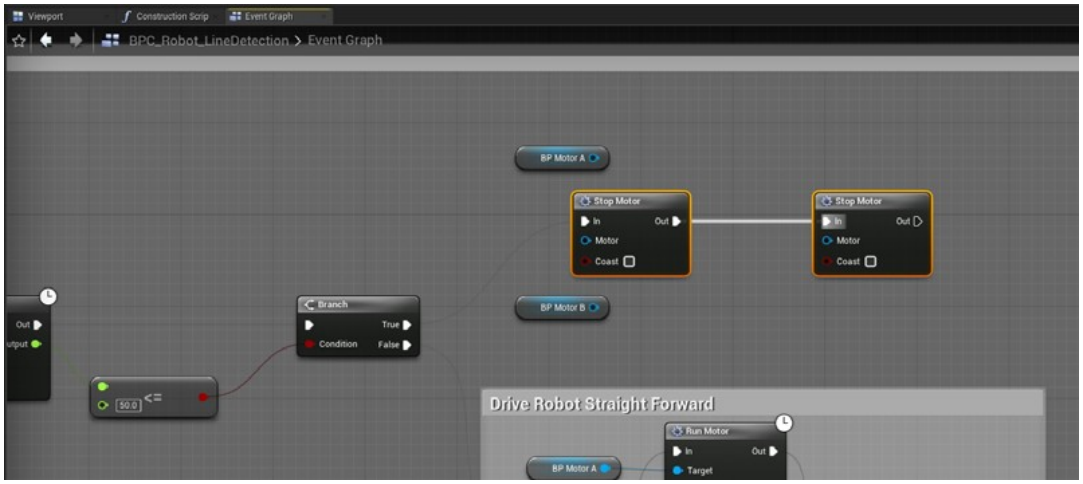


图2-43

现在连接节点：

- **点击BP_Motor_A的蓝色引脚并拖出**，连接至**停止马达节点**的**Motor引脚**
- **点击BP_Motor_B的蓝色引脚并拖出**，连接至**第二个停止马达节点**的**Motor引脚**，见图2-44

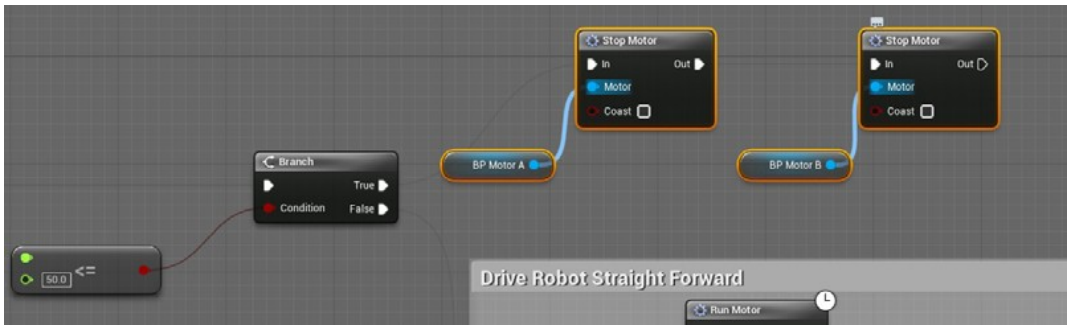


图2-44

为了完成这一部分，创建一个注释框，在其中包含用于停止机器人的代码。

- **选中我们创建的所有代码**，按下**C键**（**点击并拖动**框选BP Motor A、停止马达、BP Motor B和第二个停止马达这四个节点）
- 为这个注释框命名，**输入**“**Stop Robot**”并按下**回车键**
- 代码应该如图2-45所示

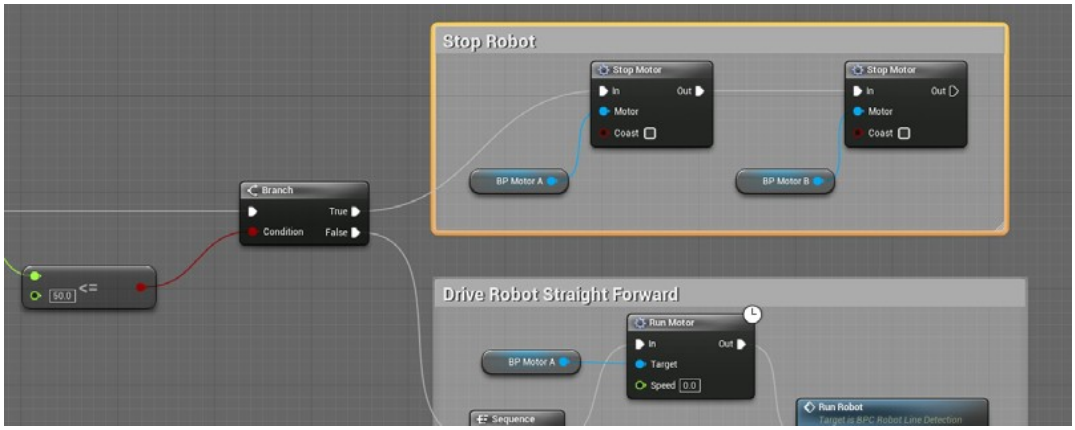
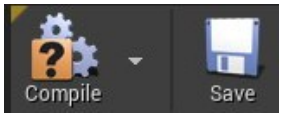


图2-45

- **编译**并**保存**蓝图



- 点击**Map_2-1_LineDection**标签页
- **运行**游戏并测试代码
- **修正代码**，直到机器人在竞技场内时，能够在未触及出界线的时候向前移动

完成的代码应该如图所示（见图2-46）

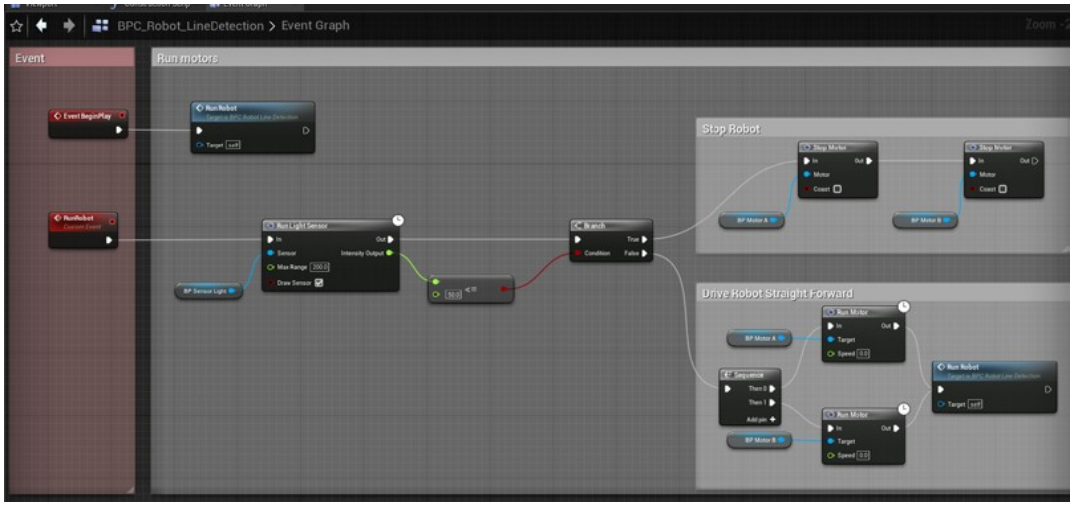


图2-46

观察：

- 机器人靠近出界线时会发生什么？
- 它会留在场内吗？
- 如何让机器人留在界内？
- 如果机器人触碰到了出界线，它需要识别出环境中的哪些区别？

机器人的条件移动

练习6：看到出界线时

在机器人相扑比赛中，机器人必须留在场内并保持移动。我们需要在机器人看到出界线时定义一种转向移动方式。目前，你的机器人看到出界线时会停止移动，我们只需要添加一个节点，让机器人转向并继续运行即可。

在构建语句的时候，我们必须将它分配到条件语句分支的正确路径上。逻辑应当是“**如果光传感器值大于阈值，则向前行驶。如果光传感器值小于等于阈值，则转向**（左右转皆可）。”

请思考为了完成转向而需要为每个马达分配的具体操作。应该使用锚点转向还是旋转转向呢？朝哪个方向转向呢？应该转多远？

第1步：选中机器人，编辑蓝图代码：

- 在**视口**中，**点击机器人**

第2步：在蓝图中编辑代码：

- **点击细节面板**（一定要确保你编辑的项目为**BPC_Robot_LineDetection**）
- **点击编辑蓝图**
- 在下拉菜单中，**点击打开蓝图编辑器**

第3步：添加机器人转向命令

- 从图2-47的代码开始，我们要修改**Stop Robot**注释框中的代码

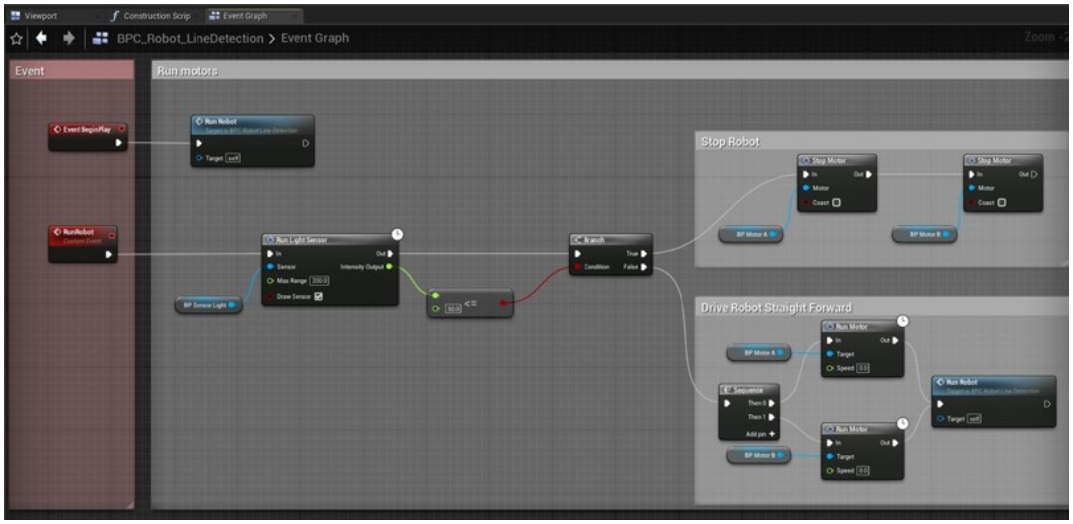


图2-47

首先，我们将**Stop Robot**注释框重命名为“**Turn Robot**”：

- **右键点击Stop Robot框**
- 在**节点注释**部分中，将文本编辑为“**Turn Robot**”

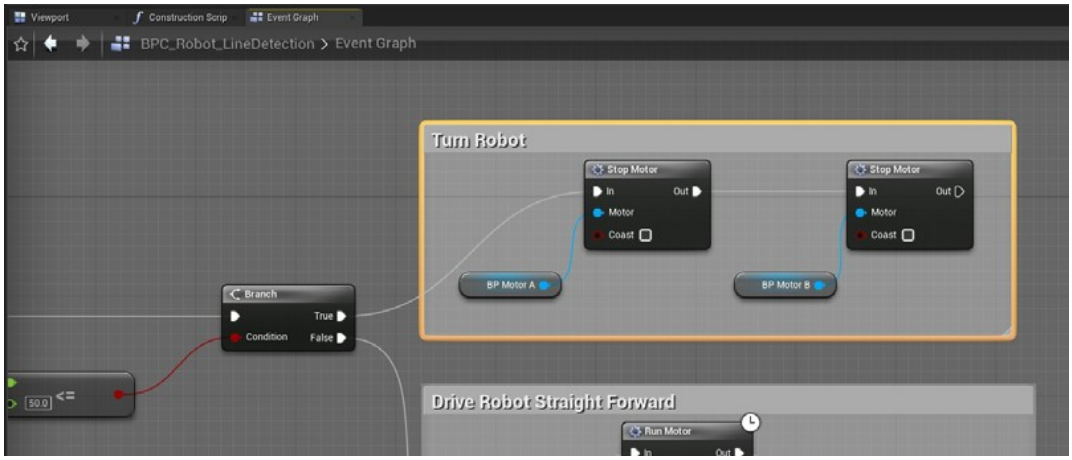


图2-48

- 在接下来的几步中，你可以自由选择机器人的转向方式
- **由你选择：删除**一个或全部两个**停止马达**节点。下方示例展示的是删除一个**停止马达**节点的情况。要删除一个节点，只需要点击节点将其选中，然后按下**DELETE**键。*如果不行，请确保项目已经停止运行*

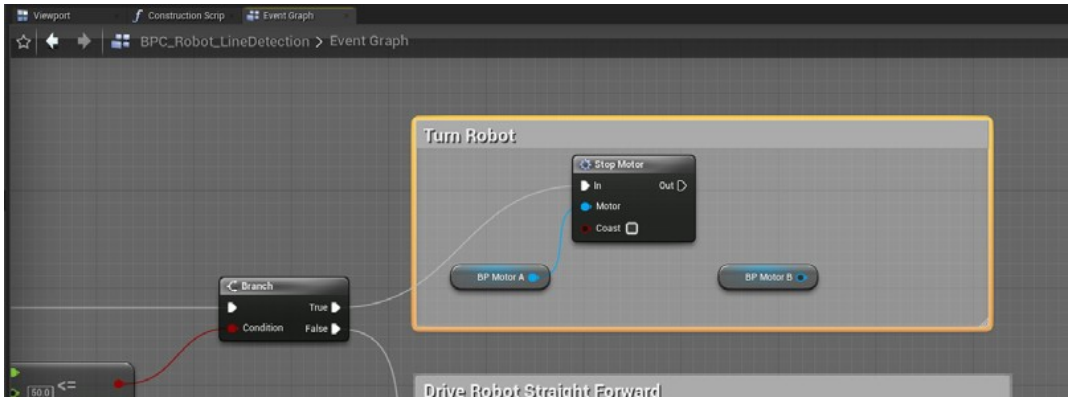


图2-49

- **由你选择**：将删除的停止马达节点替换为新的“**Run Motor for Time**”节点，并将它和你选择的**马达**连接
- **点击马达的蓝色引脚并拖出**，搜索“**Run Motor for Time**”，添加一个**Run Motor for Time**节点

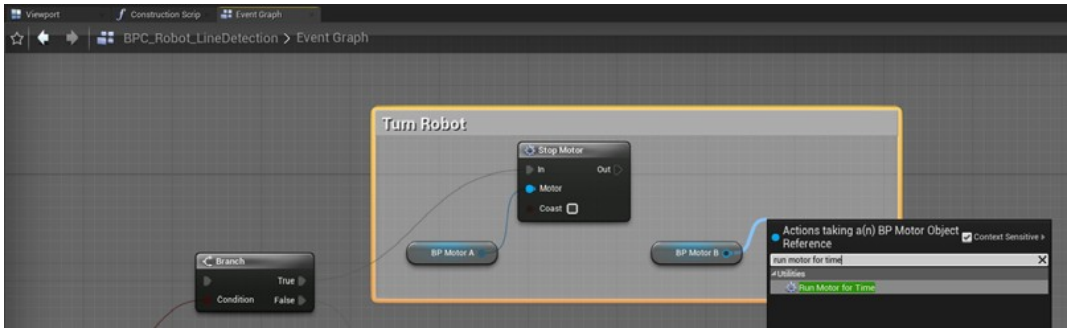


图2-50

- 下图展示了一个和**BP_Motor_B节点**连接的**Run Motor for Time**节点
- 一定要连接所有的**白色执行**引脚，这样代码才能正确执行

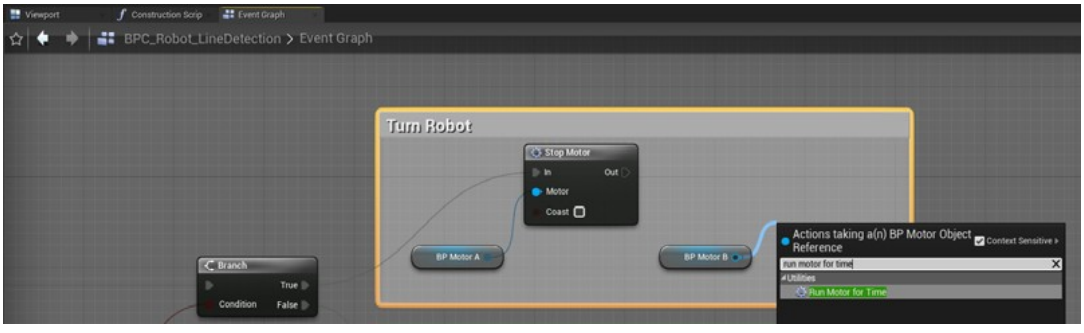


图2-51

- **更改**马达的**时间数值**（秒）
- **更改**马达的**速度数值**（-100到100）
- **打开/关闭Coast或not Coast**选项（控制函数结束的时间）。注意，当机器人看到出界线时，你需要让机器人在竞技场内向后转身
    - **时间**=马达运行的秒数
    - **马达速度**=马达旋转的速度（-100到100）
    - **Coast**=马达是否随惯性逐渐停止，还是立即停止旋转

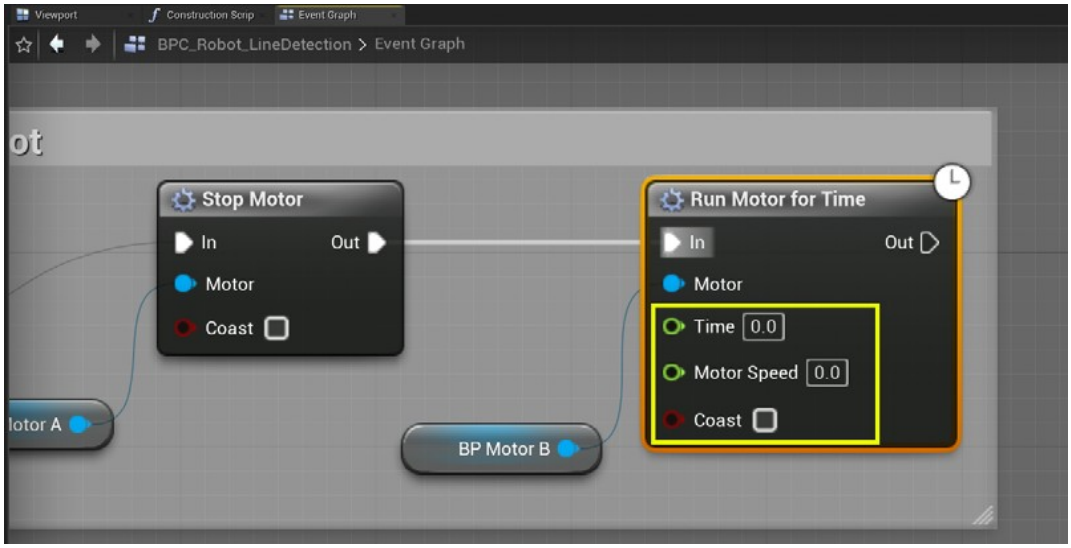


图2-52

第4步：保存，运行，排查错误，重复步骤，直到机器人能够在碰到出界线时成功转向：

- **编译**并保存蓝图
- **点击Map_2-1_LineDection**标签页
- **运行**代码
- **修正代码，**直到机器人在竞技场内时向前移动，并在看见或触碰到出界线时转向返回场内

练习7：持续移动和无限制留在场内（战斗模式）

思考：如何添加代码实现持续向前移动，并且能够无限次地转向？
我们需要让相扑机器人持续移动，但始终留在场内。
我们的机器人已经可以在看到竞技场地面时向前行驶了，只需要在完成转向后继续运行程序即可。为此，我们要在转向结束后添加一个运行机器人节点。

第5步：循环

- **复制并粘贴运行机器人**节点，或者**复制**图表中已有的节点，将它连接在**Turn Robot**序列之后
- 将**Turn Robot**中的最后一个节点输出的白色执行引脚和新放置的**运行机器人**节点的输入引脚连接
- *注：你可以点击注释框并拖动边缘，从而改变注释框的大小。*

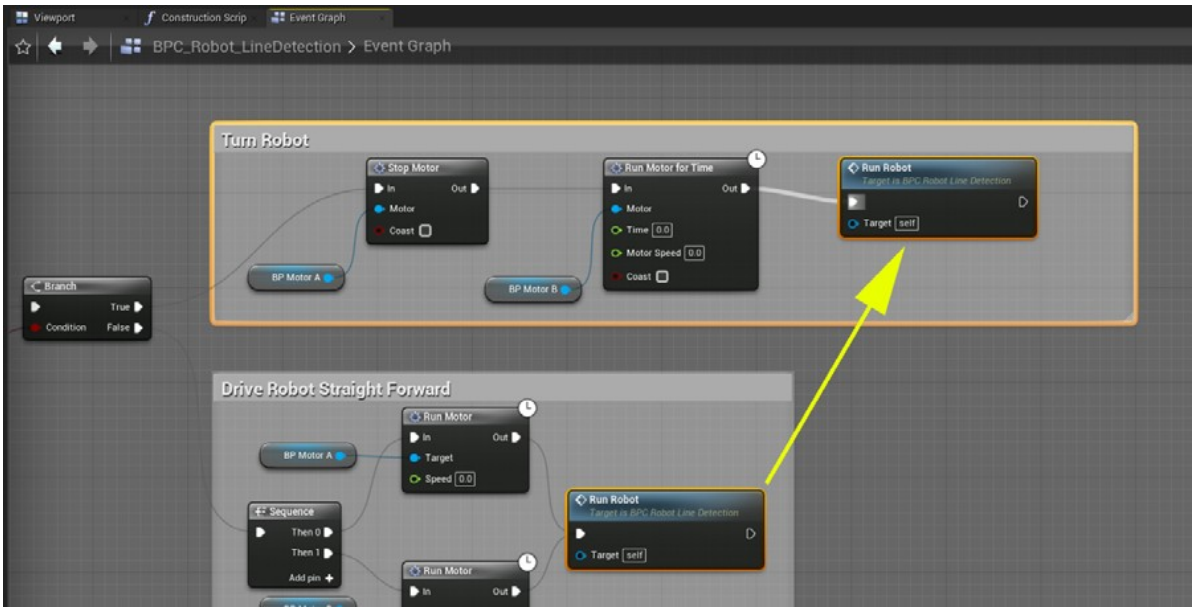
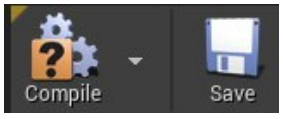


图2-53

编译并保存蓝图



- **点击Map_2-1_LineDection**标签页
- **运行**游戏
- **观察**行为并**记录结果**

练习8：测试并修正以获得稳定的行为

通过测试，确保机器人能够稳定地留在场内：

- 如果机器人***无法***继续移动，请调试代码并进行修正

准备好让自己的机器人和其他相扑机器人切磋一下吗？来搏斗吧！

练习9：相扑战

该将你的作品投入实战了。我们会复制一个机器人，并观察它们相互搏斗。

活动：复制机器人，让两个机器人在竞技场中搏斗

为了能和另一个机器人对战，我们要复制你当前的机器人：

- 在关卡**视口**中**选中机器人**
- **右键点击**并选择**浏览至资产**
- 这样就能在**内容浏览器**中直接找到**内容-> “LearningKit_Robots” -> “Blueprints” -> “Robots” -> “BPC_Robot_LineDetection”**资产

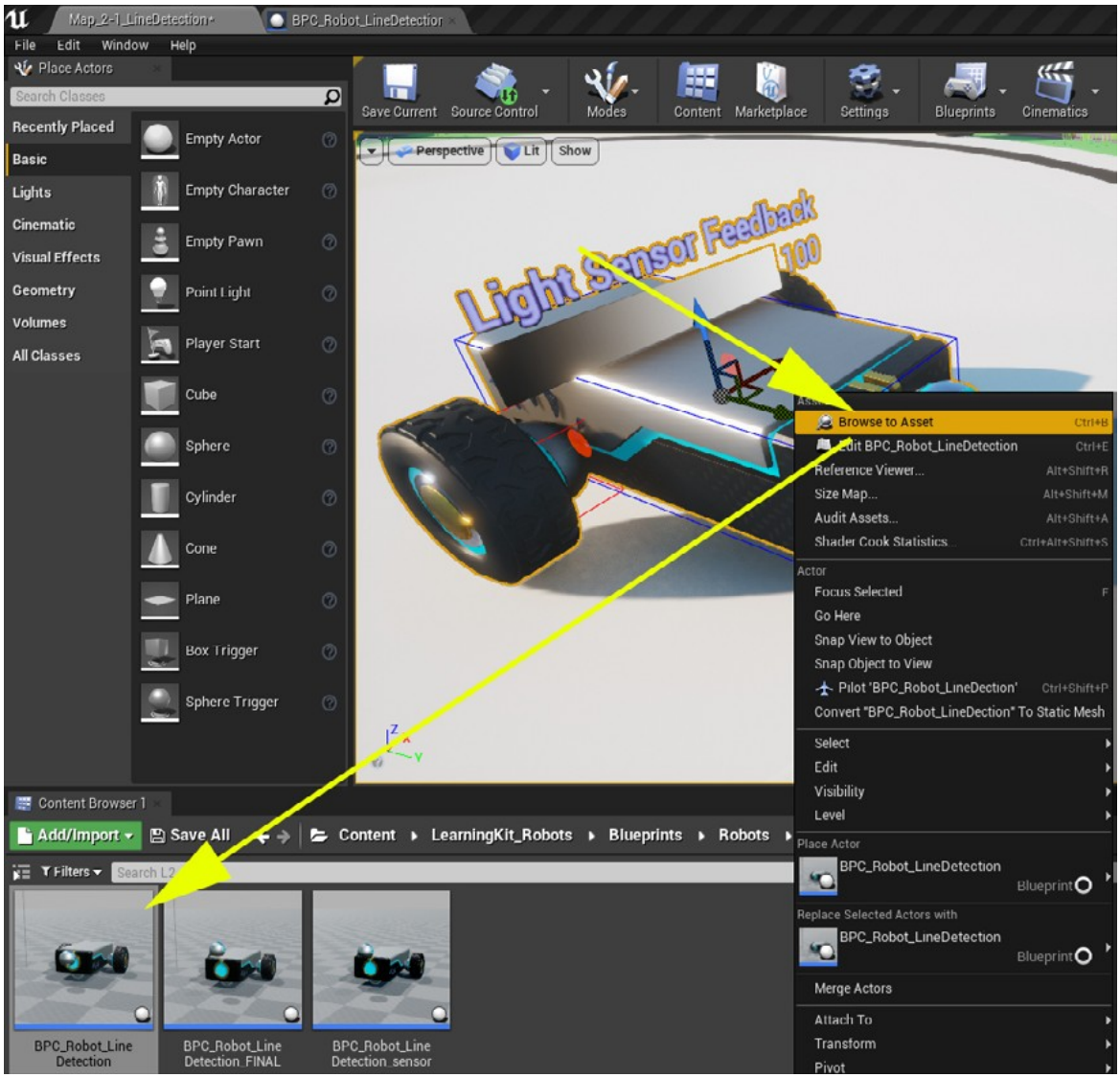


图2-54

- **选中BPC_Robot_LineDetection**
- **右键点击**它
- 选择**复制**。这样就会复制机器人资产，新资产可以立即重命名，具体可见下一步

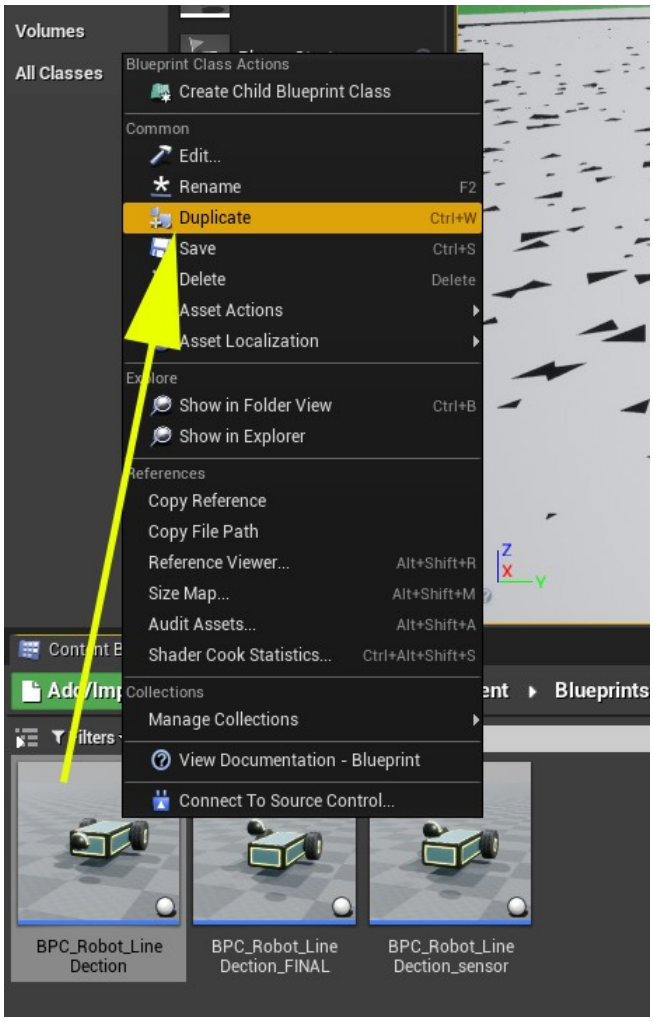


图2-55

- 将新机器人资产**重命名**为“**BPC_MyRobot_01**”，或者其他配得上相扑的好名字
- 如果你没有选中新机器人资产，则可以**右键点击**它，选择**重命名**
- **输入新名称**

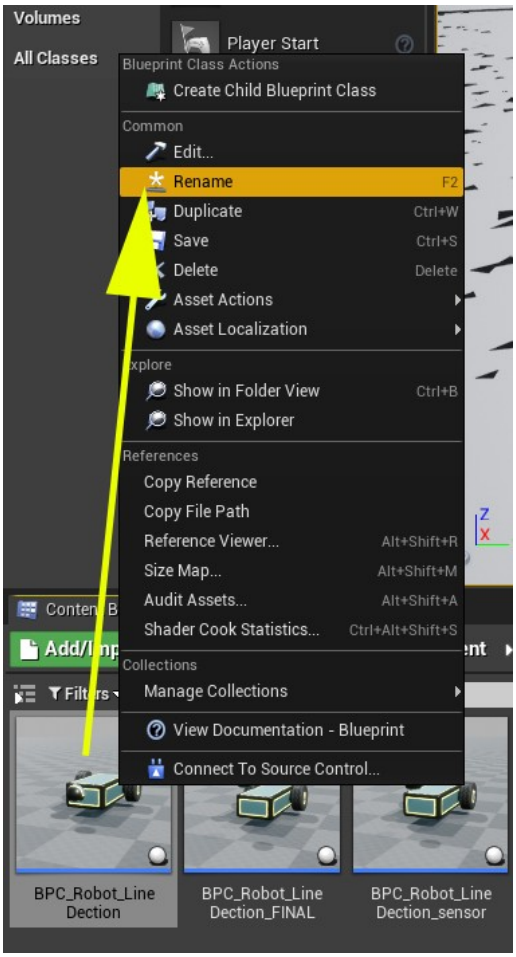


图2-56

- 从**内容浏览器**中，将**新机器人资产拖动**到关卡视口中，将它放入相扑竞技场
- 如果你看到了**Actor放置警告**，点击**确定**即可。你需要确保新放置的机器人一开始就完全位于地面上方

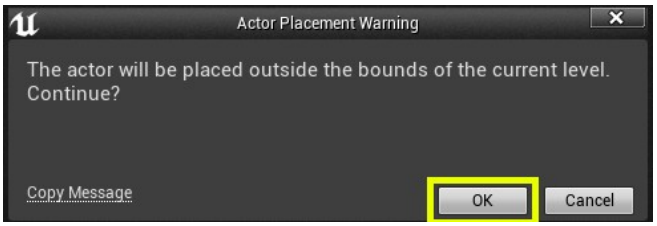


图2-57

- 一定要使用**移动工具**将它**设置在地面上方**，这样它才能留在地面上，或者稍稍位于地面上方
- 见图2-58

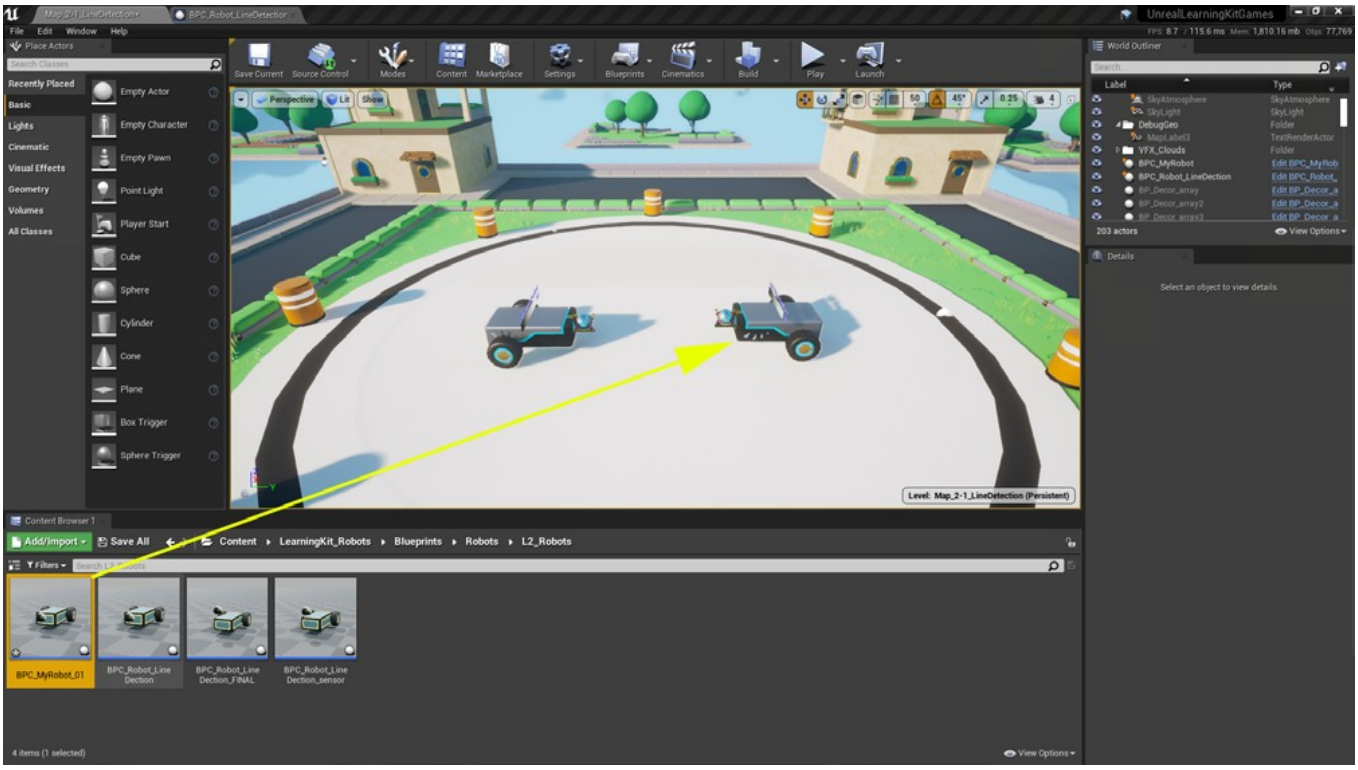


图2-58

- 接下来，**打开**界面左上角的**文件菜单**，选择**保存全部**（或者按下**Ctrl+Shift+S**）

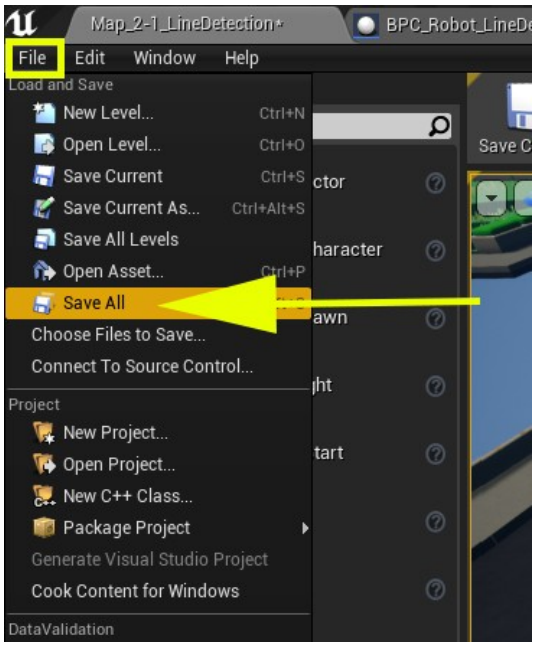


图2-59

让比赛双方各就各位，命令它们开始战斗。

让两个不同的机器人背靠背，或者设置成其他起始位置：

- 使用**移动工具**，**点击每个机器人并拖动**到理想的起始位置。
- 在每个机器人的蓝图中，**为每个机器人编写代码**，使用条件语句和循环，使它们被其他机器人前后推挤时尽量向前方移动，同时注意竞技场地面，对出界线及时做出反应并转回竞技场地面。

回顾

- 哪些方法很有效?
- 存在哪些挑战?
- 为什么你依然需要亲手实践?

资源

工作表和/或代码说明

阈值计算表

基础教程

打开虚幻引擎工程之后，点击**帮助**菜单，选择**教程**，然后点击**基础**。

拓展活动

- 尝试让机器人行驶得更快。机器人加速后会出现哪些新的挑战呢?
- 让学生共享机器人进行相扑战。可以浏览“资源指南——复制粘贴其他学生的机器人代码”文件
- 装饰机器人，在设计中添加特别的细节，也许能让它在相扑战斗中更有优势
- 如果你能为机器人安装更多马达来执行一个操作，你会加装什么呢?

评价

评价标准

学习概念	成果优异	熟练精通	基本完成	有待培养
机器人设计概念	可以说明机器人传感器组件及其相互运作原理。工程可展示出课程中对传感器的使用，以及在机器人上最佳的安装位置。演示或学生文档能够展示出组件的定义。	展示出对机器人的传感器的运用。能向教师充分表达对机器人组件的理解。	能够让机器人对传感器做出响应，但无法提供有意义的理解。展示出对硬件组件的基本理解。	无法证明对机器人传感器及其功能的理解。无法展示出对硬件组件的理解。
软件概念	展示出复杂的命令组合和工程目标。能够说明使用传感器、传感器位置和理想的机器人位置或操作所需的命令组。学生展示或文档展示出了命令或命令组。	学生理解使用传感器、传感器位置和理想的机器人位置或操作所需的命令组。学生展示中提及了命令或命令组。	学生可以在提示下基本理解使用传感器、传感器位置和理想的机器人位置或操作所需的命令，但未能在演示中展示。	无法证明学生对命令及其功能的理解。学生展示中不包括或未提及命令。
代码概念	可以调试代码错误，展示了复杂的代码组合和独特用法。可以说明大部分代码。学生演示或说明文字中包含了代码。	展示出对默认设置、循环和条件命令的理解。学生在演示中至少引用了每种命令类型的代码各一条。	学生可以在提示下基本理解代码，可能未在学生的展示中提及。	无法证明学生对命令代码及其功能的理解。学生展示中不包括或未提及代码。
现实概念	能够提出在现实世界中运用机器人、代码和移动的创意。展示出相关理解并能够予以记录。	理解在现实世界中应用代码、移动和机器人的部分用例。在学生展示中包含至少一个示例。	对现实世界中的代码、移动和机器人应用具备基本理解，可能可以在提示下口头说明，但未包括在学生展示中。	无法证明学生对现实世界中运用代码、移动和机器人的用例有所理解。
挑战活动（机器人相扑）	展示出创新想法，或者为相扑机器人模型和行为新增了工具。能够复制机器人，以便和其他机器人作战。	成功编写代码，使机器人能够在相扑区域中移动，并在检测到出界线时转向。	基本理解如何为相扑机器人编写代码，但无法让相扑机器人留在场内。	不理解相扑机器人的功能，无法展示出传感器的运作原理，无法为相扑机器人活动编写代码。

学生指南

虚拟机器人培训

第2课：机器人相扑