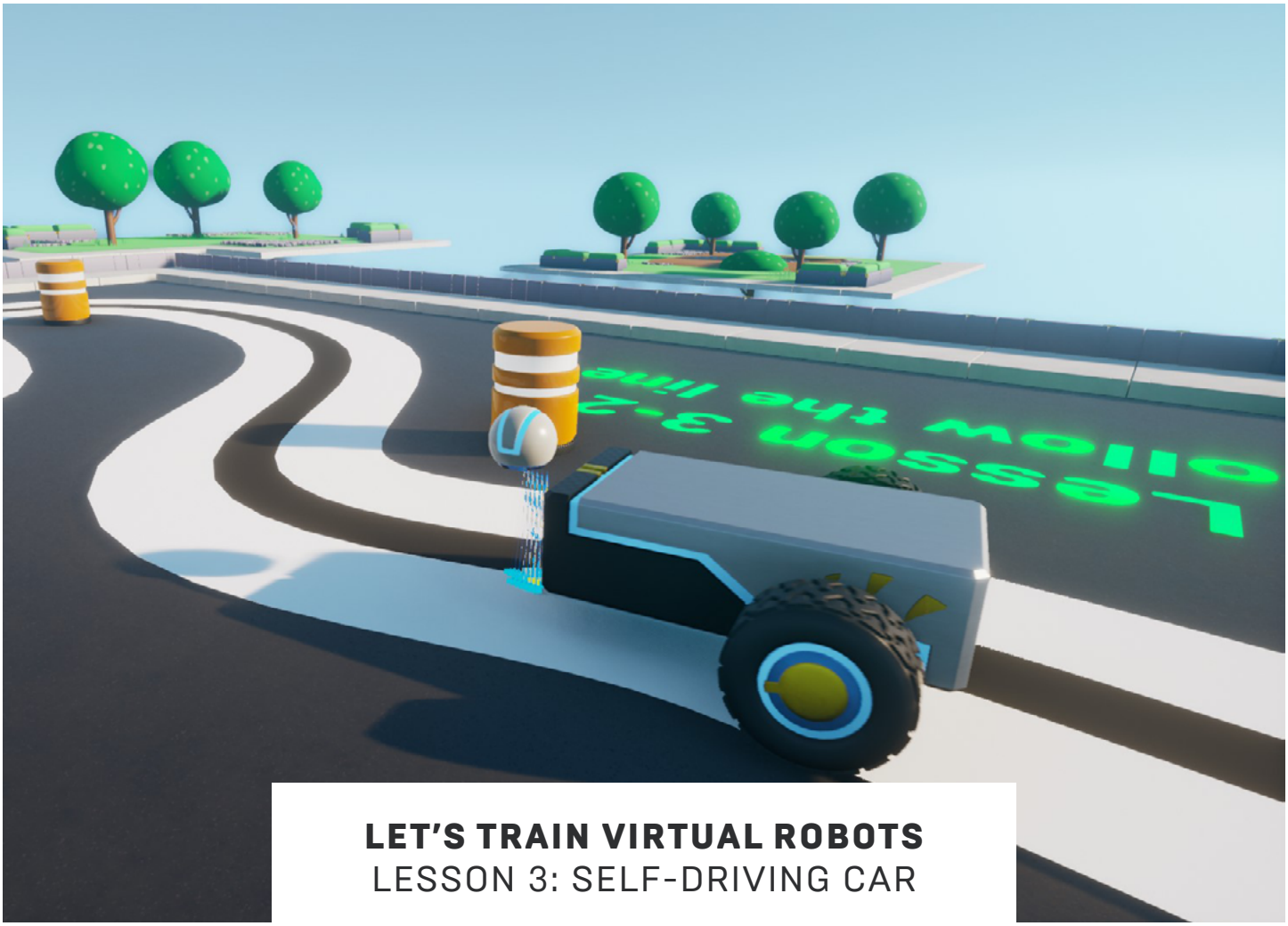
LET'S TRAIN VIRTUAL ROBOTS

LESSON 3: SELF-DRIVING CAR

UNREAL ENGINE

STUDENT GUIDE

TABLE OF CONTENTS



LET'S TRAIN VIRTUAL ROBOTS
LESSON 3: SELF-DRIVING CAR

OBJECTIVE

This lesson is similar to the previous learning in Lesson Two - Sumo Robots, but will show students how similar concepts can be used to accomplish new goals. It will also build their confidence before they attempt the extension/future activities that come later in the Unreal Learning Kit.

This lesson will again deal with sensors, specifically how robots can make decisions based on received sensor data.

Students will create a delivery robot, programming it to follow a line to its destination. In an extension exercise, students can expand their learning by building the robot with two sensors. With a little extra coding, they'll be able to make its line-following movements smoother and more accurate.

GETTING STARTED

Getting Started Guide

If you are installing and using Unreal Engine for the first time, please complete the Getting Started Guide before proceeding through this activity. The guide includes instructions for getting the Unreal Learning Kit project files installed so you can successfully complete this activity.

You can find the Getting Started Guide here!

Lesson Intent

We recommend all students go through all the lessons in order, so they gain a full perspective of the Unreal Learning Kit and how it supports both robot creation and coding motors/sensors in the Unreal Engine environment.

The Unreal Learning Kit explores the concepts of robotic engineering using applied physics. While giving students immediate feedback on their coding and providing principles/coding language knowledge that they can apply to other physical robotic systems they may work with in the future.

LESSON THREE: SELF-DRIVING CAR

Introduction

Have you seen Tesla factory assembly robots follow a path (as in How the Tesla Model S is Made | Tesla Motors Part 1 (WIRED),) or ridden in a self-driving car? How do these robots see the road or path and stay on it independently? Let's explore the world of autonomous vehicles by creating a delivery bot.

Line-following robots are your on-ramp to the wonderful world of autonomous vehicles. This simple exercise will give you an introduction to just one way that a vehicle can use the environment to navigate independently. To understand how a line-following robot works, you must understand what a light sensor detects, and how that information can be used to stay on or within a preset line or path.

In this activity, we will be coding a robot with a light sensor to follow a line, thus becoming a line-following robot.

Lesson Overview

To control the robot, we will learn how to turn a robot away from the line and back toward it again, ultimately following the line as the robot continues to move forward.

If the robot sees the line, it will turn slightly away from it. When it no longer sees the line, it will turn back the other direction to look for the line again and continue. This process must be repeated continuously to have the robot virtually stay "on" the line. The robot will have a bit of a "wiggle" effect as it travels along this line.

The code must contain a loop that includes a conditional statement where the sensor continues to test the result and make adjustments to turn toward or away from the line continuously.

Using the light sensor will require understanding that:

- The light sensor will return a numeric value to the robot
- Our conditional statement will test the current value of the sensor to determine if the number is within the low range (black surface = line) or the high range (light surface = floor).
    - If the value is in the low range, it means the robot is seeing the line, so it will be programmed to drive away from it
    - Likewise, if the value is in the high range, it is seeing the floor, so it will be programmed to drive toward the line
- The code will need to loop continuously, so it tests the sensor and controls the motors several times per second
- The robot will stop at its destination. This activity's current robot model will stop when it hits the invisible Blocking Volumes box. We do not have to code this action.

An extension of this activity explores how a second sensor can create a smoother action for the robot as it travels forward.

CODING CONNECTION

Completing this activity will help students understand the use of loops and conditional statements in a practical environment. This robot will have a light sensor to detect the difference between light and dark. The returned sensor values will test against a threshold value to determine whether it is seeing the white border (ground) or the line (path) as it travels along.

The conditional statement will help the robot to follow the line, while the loop will help it continually move until it reaches its destination.

Let's find that line!

STARTING THE LESSON

Robot Construction: Using a Light Sensor

In our last lesson, we used a light sensor to help the robot see the out-of-bounds line in the sumo ring. We will use the same orientation for our delivery robot but will change our code to help the robot use that same sensor to guide it along the path.

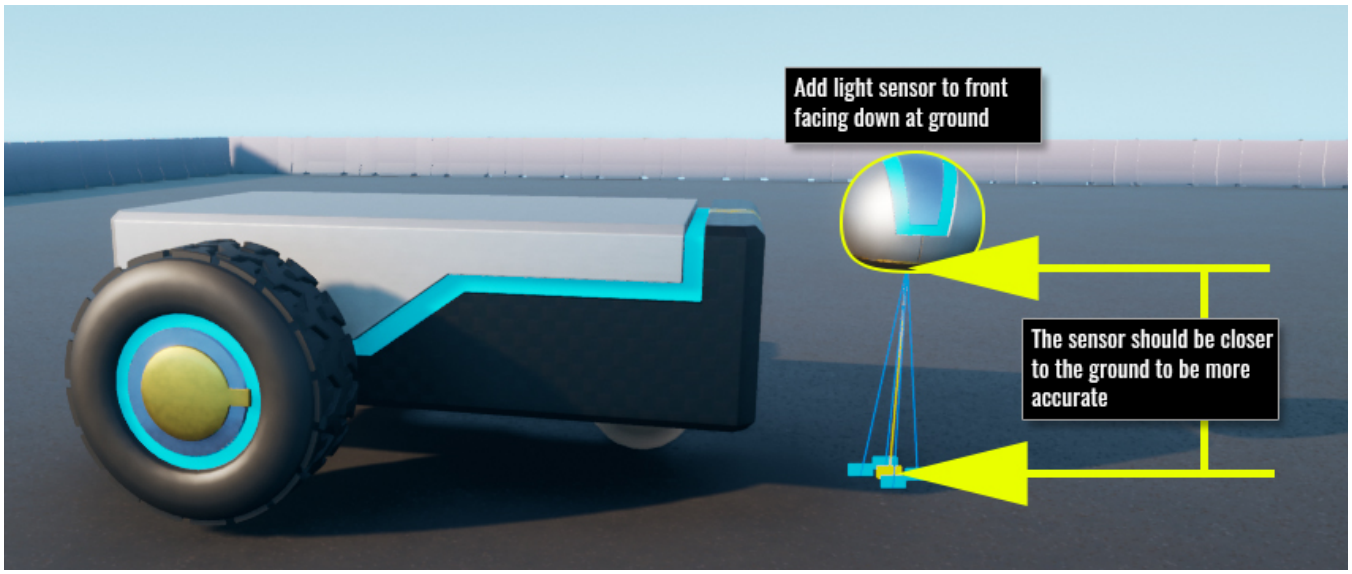


Figure 3 - 1

Defining a Purpose

Our robot's specific task will be following a line on the ground. We will identify the requirements for the environment and set any rules that must be followed to complete the task.

Environment

- The robot will be placed on the line facing in the direction it must travel.
- The line to follow will have smooth curves and will not overlap itself.
- The driving surface will be flat.

Rules

- The robot must drive autonomously via code with no user intervention.
- The robot will start driving immediately and continue until it reaches the end of the line.
- If the robot strays away from the line, it should be stopped and modified before trying again.

Requirement

- Uses a sensor to "see" the black line/path and the white border/ground.
- Drives away from the line when "seeing" the black line.
- Turns toward the line when "seeing" the white border/ground.

HANDS-ON EXERCISES

Exercise 1-a: Opening the Line-Following Map

In this activity we will open a project file with a starter robot in place at the start of the line. This robot does not function, so you will need to add the sensor and the coding to make it work. We will first add a light sensor to it, test the sensor values, and then build additional code to decide when we want the robot to move.

- In the Content Browser navigate to the Content -> LearningKit_Robots -> Maps_Robots folder
- Double click on the L3 folder to open it
- Then open Map_3-1_FollowLine by double-clicking it
- You should now see a map with a robot at the beginning of a curving black-on-white path extending in front of the robot, showing a destination parking spot beside two other vehicles

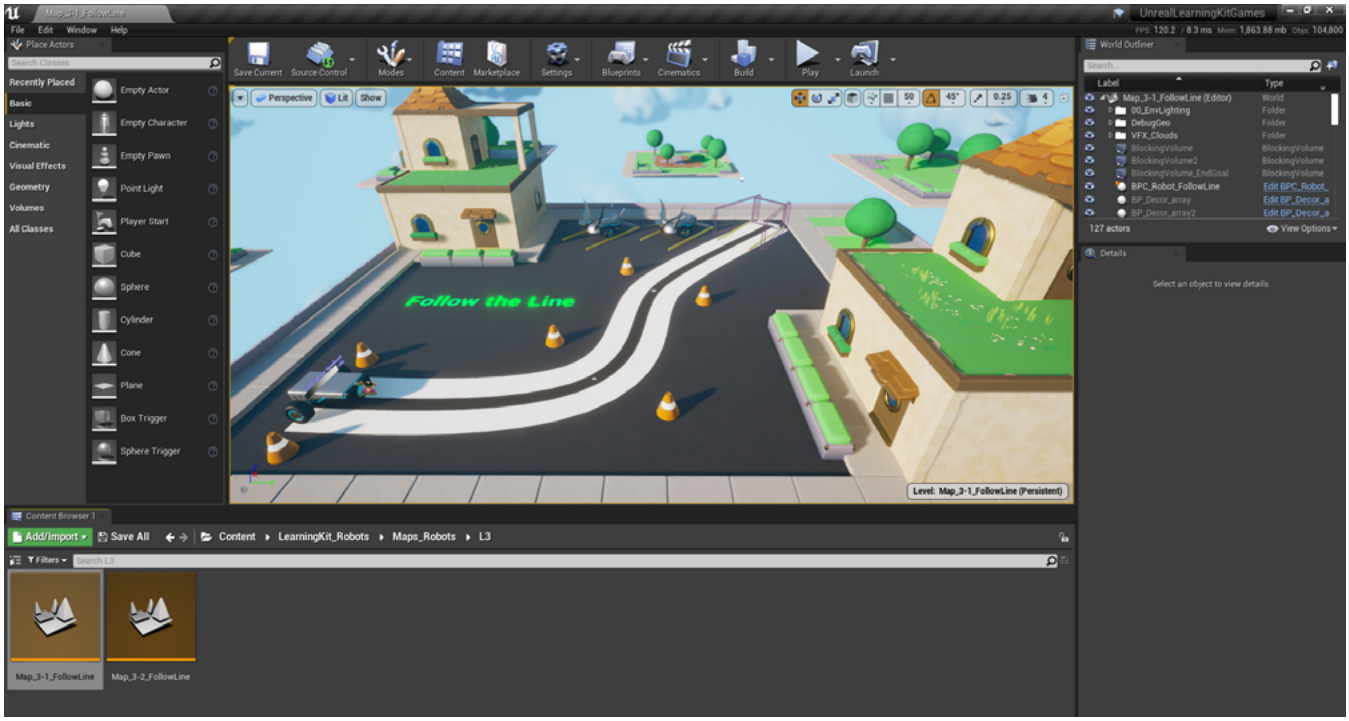


Figure 3 - 2

Exercise 1-b: Adding a Light Sensor to the Robot

- We will be using the light sensor in this activity. The position of the sensor will affect its accuracy and its response to the actions we will be coding

Before we add the sensor, we need to consider the following situations:

- The height of the sensor from the surface will affect its accuracy
- The front-to-back location of the sensor on the robot affects the turnaround action of the robot, which will affect its reliability

We have added a helper in the Blueprint viewport for you. Look for the Cleverlike logo, see image below.

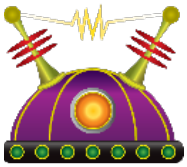


Figure 3 - 3

Edit the Robot

- Select the Robot - Click the robot in the Viewport (you should see BPC_Robot_FollowLine displayed as our active Actor on the right side of the screen Details panel and World Outliner)

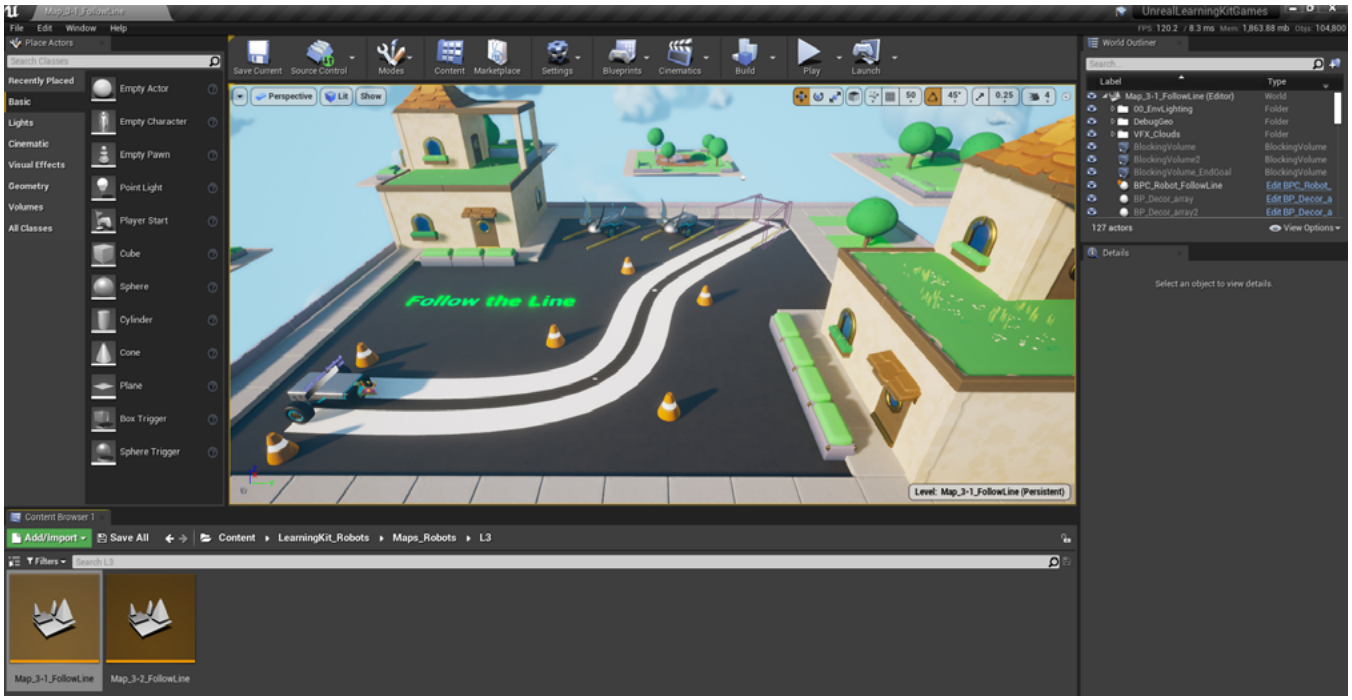


Figure 3 - 4

- In the **Details panel**: **click Edit Blueprint**, then **Open Blueprint Editor**
- You should now see the **Event Graph** inside the of **BPC_Robot_FollowLine** Blueprint

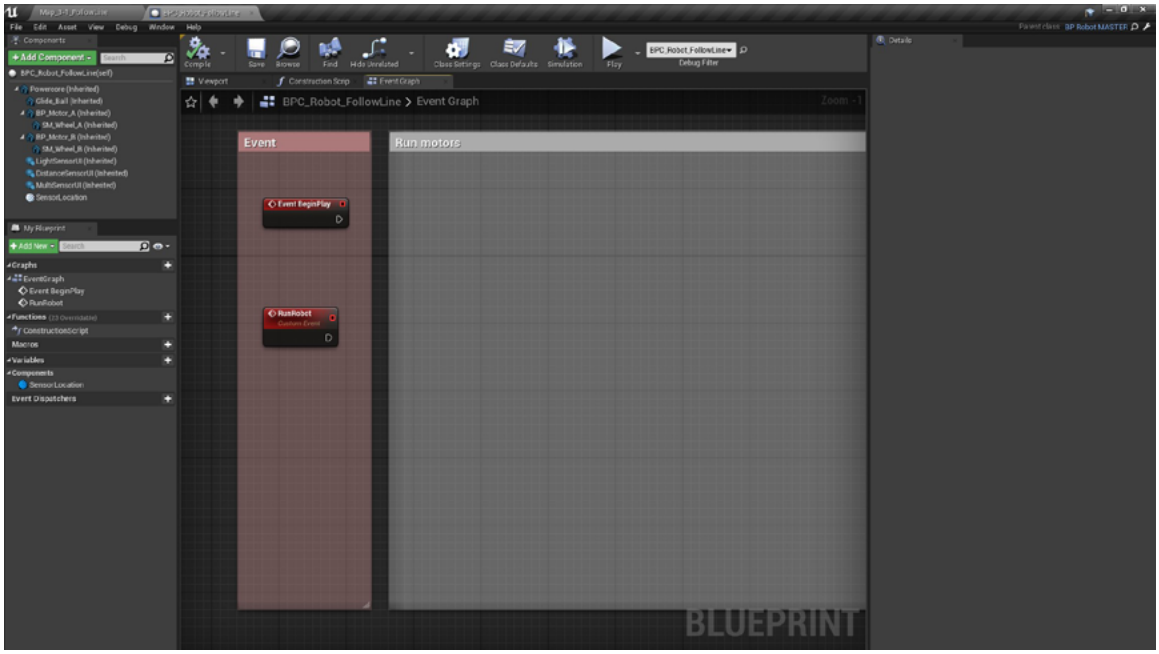


Figure 3 - 5

- For exercise 3-1, notice that you do not have any code here, except for **Event BeginPlay** node and **RunRobot** nodes in the Condition box on the left, with no coding in the Run Motors box on the right

Add the Light Sensor

- In the **Components Panel** at the top left of the interface, **click** the green **Add Component** button and type “sensor” in the Search bar
- Select the **BP_Sensor_Light** by clicking it

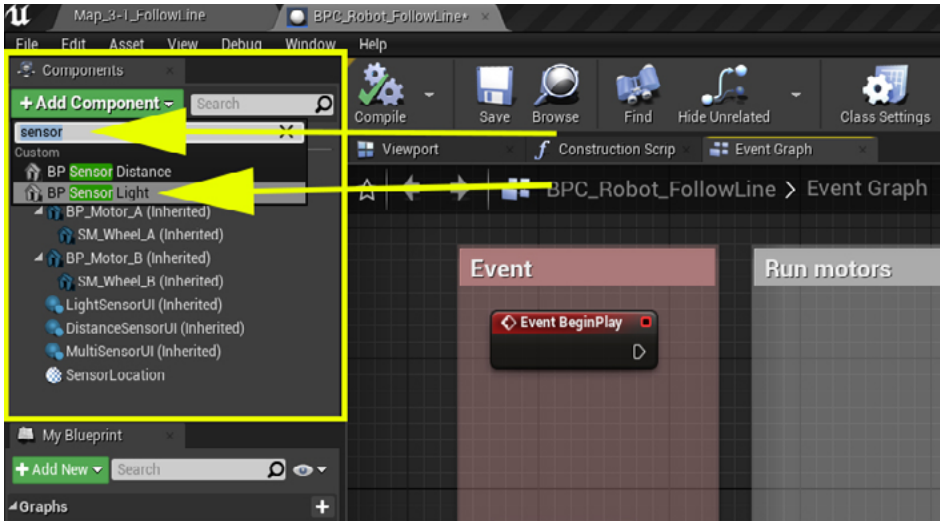


Figure 3 - 6

- This will add your new component at the bottom of the list of components
- If it does not appear directly under the **LightSensorUI** component, you may have added it under another component. You will need to click and drag the **BP_Sensor_Light** up and onto the **Powercore**, for it to be dependent on our robot’s Powercore. It will then appear at the bottom of the component list. If it is under another component, it becomes a “child” of that component, rather than being a child of our Powercore (robot)

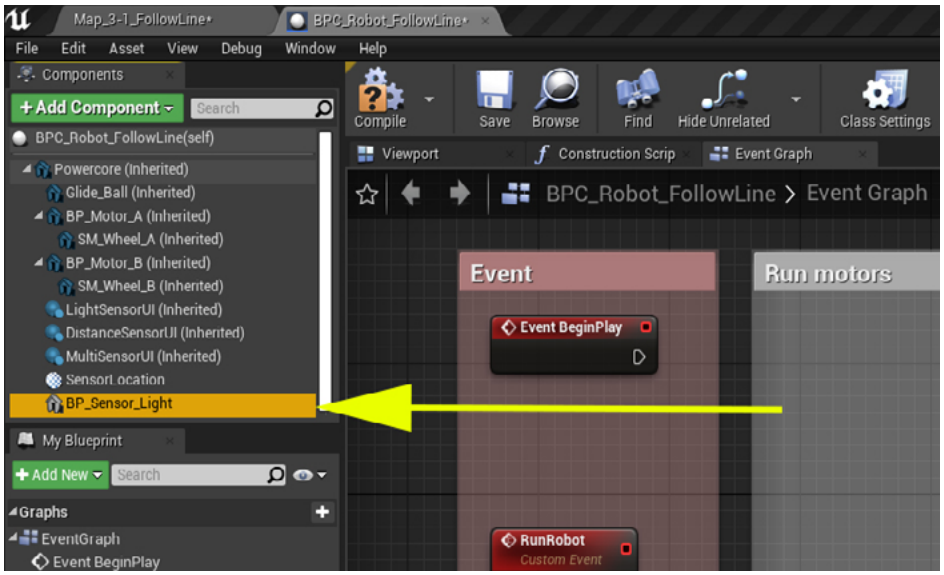


Figure 3 - 7

- To see where the sensor is located in relation to the **Powercore**, click the **Viewport** tab (see Figure 3-8 below)

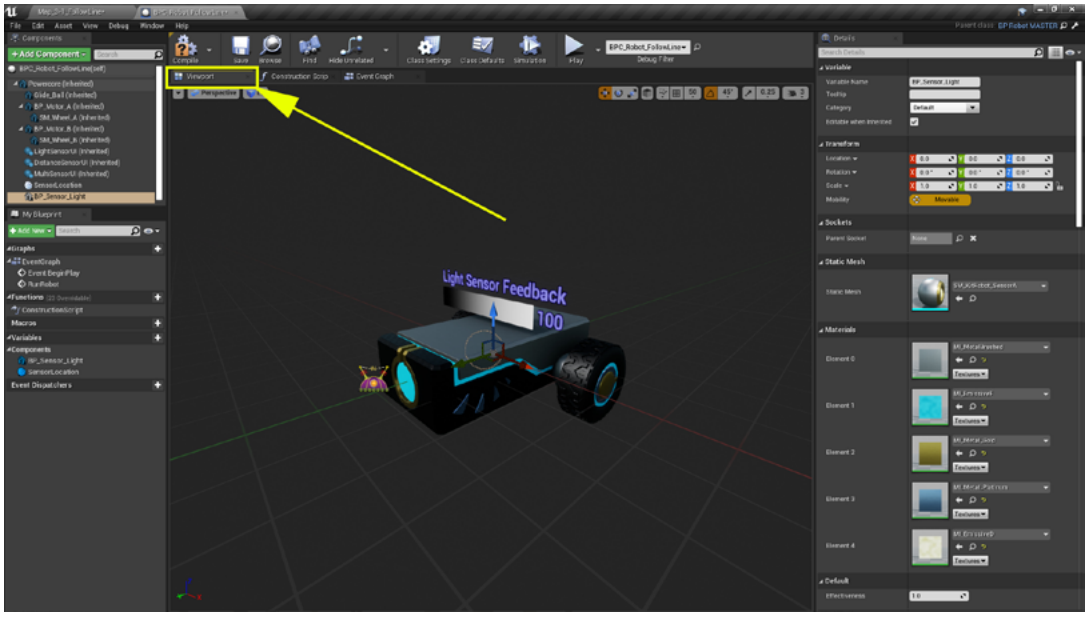


Figure 3 - 8

- Orient your robot until you can see the helper. **Pro tip**: Pressing the **F key** will zoom in on the selected object, enlarging it and centering it in your view
- The sensor has been placed automatically inside the **Powercore**. You can see it selected as a faint yellow shape within the robot. We need to move the sensor to a better position where it will function the most efficiently. To do this, use the movement tool located in the top-right of the **Viewport** window.
- Click the first tool to use the Select & Translate objects tool (Move Gizmo), or **press** the **W** key, (the gizmo shows up as a 3-headed arrow). We can click and drag the arrows of this gizmo to move the sensor up, down, left, or right until the light sensor covers the Cleverlike logo. This is our beginning position

- Our sensor is still pointing sideways, so we need to point it to look downward at the ground
- Again, in the **Viewport** window, on the right side of the toolbar, click on the second tool, or press the key **E**, to use the Select & Rotate Objects tool (Rotation Gizmo) to turn it to face the ground

- Notice that the selected object (the light sensor) settings are changing in the **Details panel** to the right, under the **Transform** section. You are changing the Location settings with the Move Gizmo (W) and the Rotation settings with the Rotate Gizmo (E). Notice that as you drag the sensor up or down, the value in the Location Z value box is changing in the **Details panel**
- Your robot with the sensor will now look like this image:

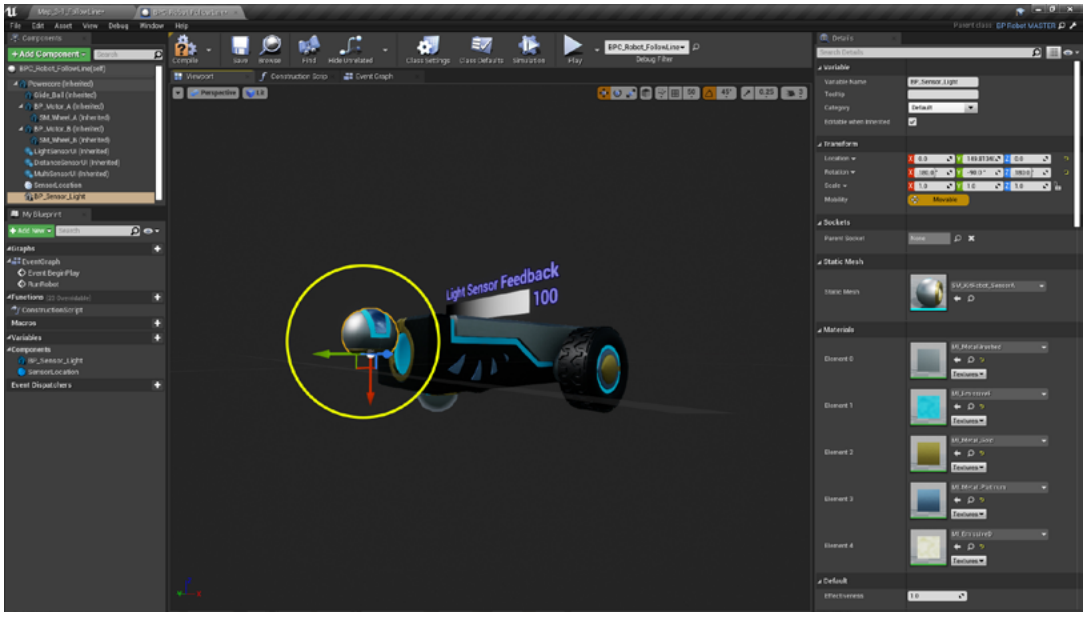


Figure 3 - 9

- With the **BP_Sensor_Light** selected, notice the **Details panel** on the right. You can type values in the location fields if that works better for you

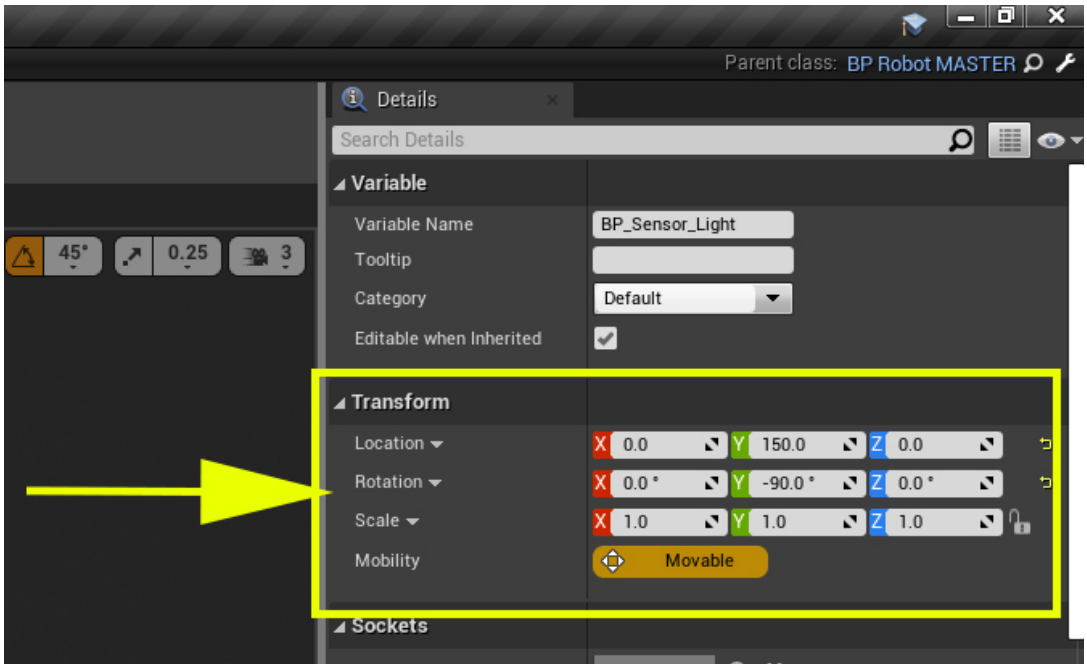
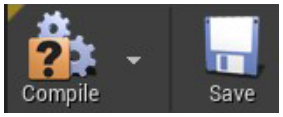


Figure 3 - 10

- Last, we need to compile our change in the robot's internal code, **click Compile** and Save in the **top left** of the toolbar
- *(After compiling and saving, the Compile button will change from the orange question mark to a green checkmark).*



Click the **Map_3-1_FollowLine** tab at the top of the screen to return to the level and **zoom** in (scroll wheel on the mouse or F key) on the robot. Your viewport should now display a sensor at the front of the robot

Figure 3 - 11

- Note the robot in the level and the direction the robot is facing. The robot must begin facing the route we want to drive forward towards and be on the left edge of the black line, or it will not locate the line properly
- Arrange your view of the robot to the direction it needs to drive forward on the path line

Figure 3 - 12

Exercise 2: Getting Input from a Sensor

The robot needs to detect the difference between the white border and the black path line. We will use the light sensor to detect the difference. We will focus on the black line, so our two conditions will be "seeing" the line and "not seeing" the line

Sensor Type

- We will use a light sensor that will measure reflected light. A light or white surface reflects more light than a dark surface, thus a light sensor will return a higher numeric value on a light surface and a lower numeric value on a dark surface

Sensor Location

- The sensor must look down at the ground while the robot is driving
- The sensor must be close to the ground to get an accurate reading
- The sensor should be located at the front of the robot, away from the wheels

Adding Light Sensor Code

Once the sensor is placed in an ideal location, we will need to add commands to the robot's code for it to work and to return the input it sees for making decisions. We will now add the light sensor code to the Blueprint Editor to direct the robot to use the sensor.

Note: Be sure that you have a notepad or a shared documenting system for your school and class to log light sensor values as you proceed through the following steps.

- In the Level **Viewport**, click on the robot
- Look at the **Details panel** (on the right), be sure it shows that you are working on the BPC_Robot_FollowLine

Open the Blueprint Editor

- In the **Details panel, click Edit Blueprint**, then **Open Blueprint Editor** (your screen should now display the **Event Graph** inside the Blueprint editor; if you are still viewing the Blueprint's viewport, click the **Event Graph** tab)
- You should now be looking at the **Event Graph** (see Figure 3-13)

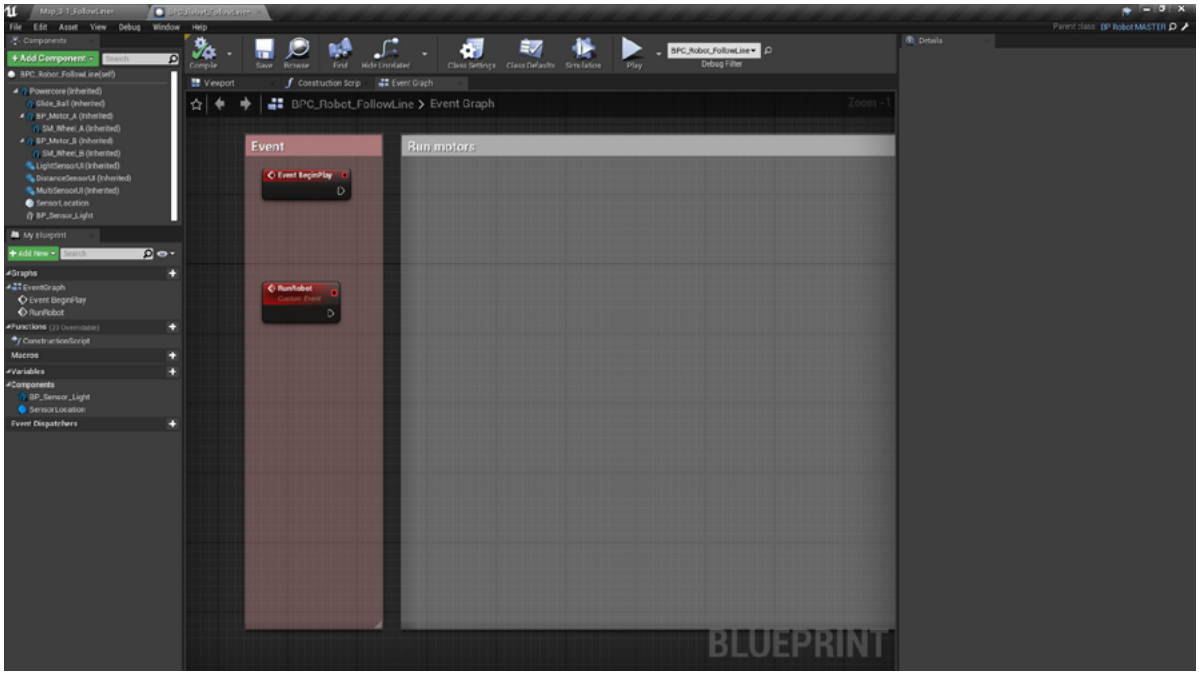


Figure 3 - 13

- You will notice that there is no code here, except for **Event BeginPlay** and **RunRobot** nodes in the **Events box** on the left, with no coding in the **Run Motors** box on the right
- In the left side **Events box**, from the **Event BeginPlay** node, click the output execution pin and drag to the right into the gray **Run Motors** box to create a (white) wire
- When you let go, a list of executable actions will pop up, and you will be able to search for the function we need
- Next, type “run robot” to search for the **Run Robot** function

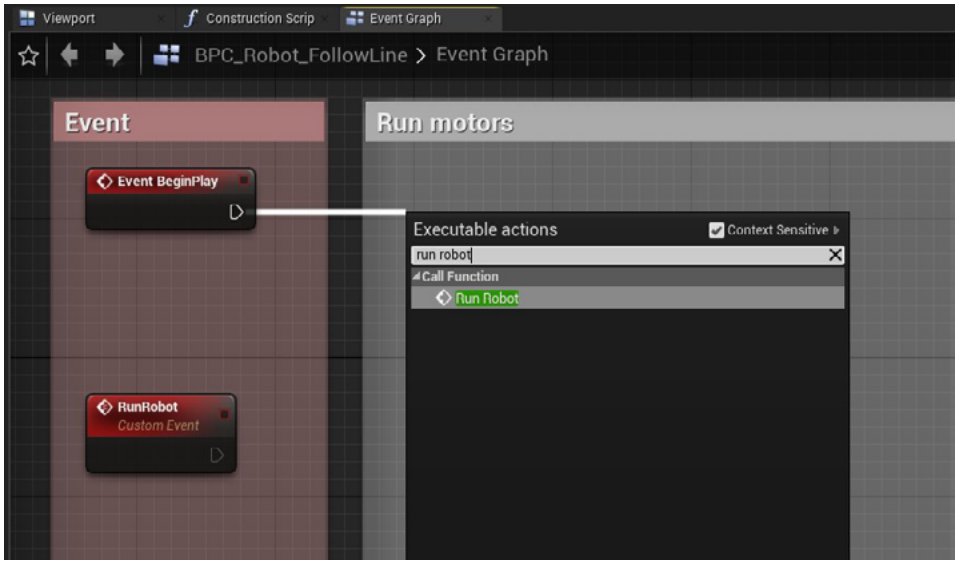


Figure 3 - 14

- When **Run Robot** is selected in gray, press **enter** on the keyboard to select the function. This will add a **Run Robot** node, and automatically create a wire (displayed as a white line) from the output execution pin to the input execution pin connecting the nodes. This will call the **Run Robot** event when you play the game

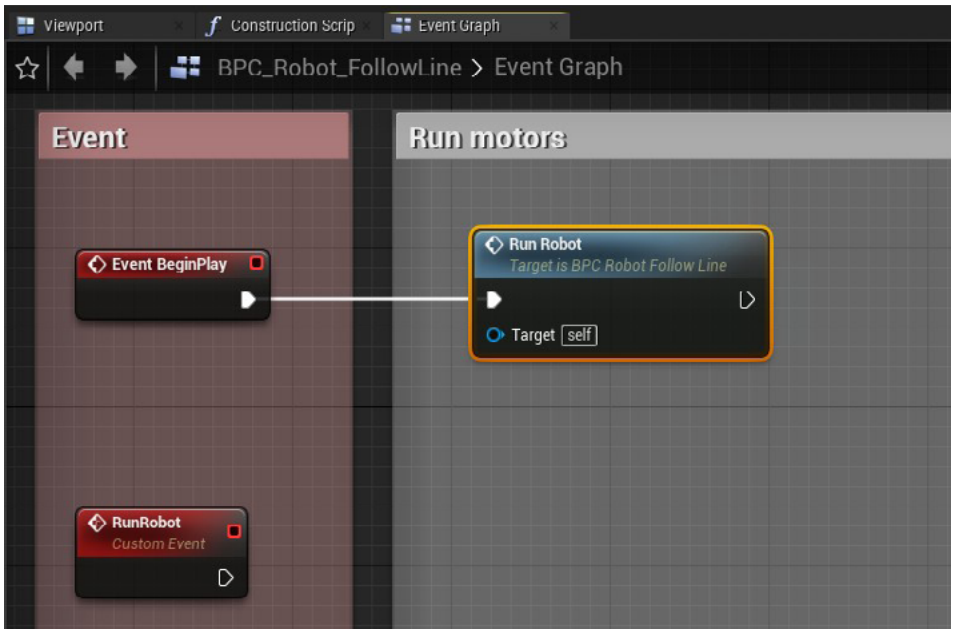


Figure 3 - 15

- *Remember to **Compile** and **Save** every time you add code!*

- *Now we will hook up the sensor so that it will run when the **Run Robot** function is called. From the **Components** panel, click on **BP_Sensor_Light** and drag it into the gray Run motors box (see Figure 3-16)*

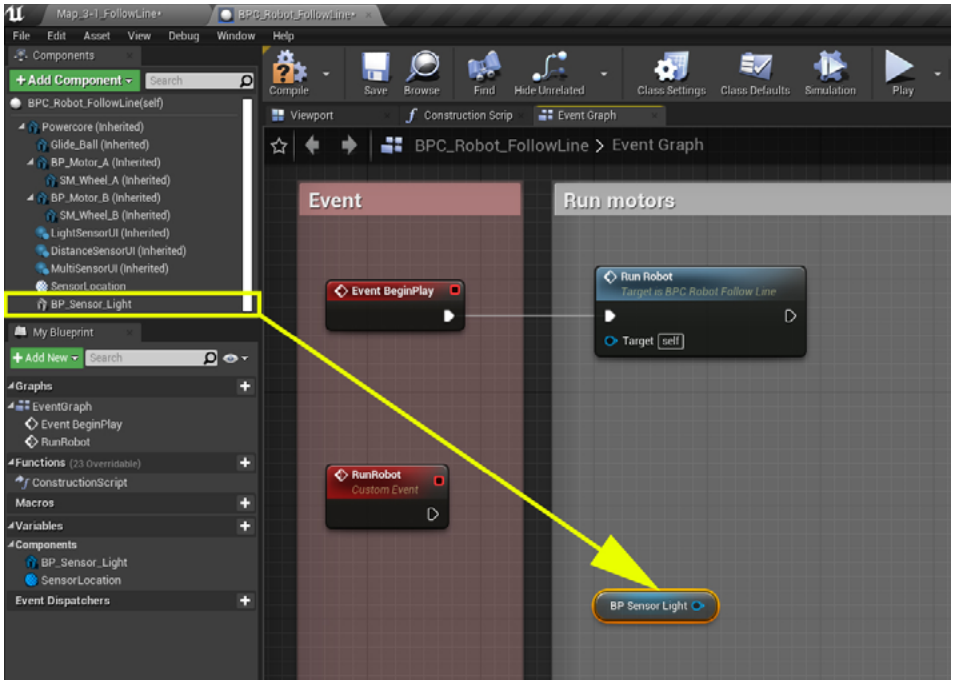


Figure 3 - 16

- *Now, we need to connect our sensor to a node to run the sensor. We can create a new node by dragging a wire from the blue **BP_Sensor_Light** output pin to the right and releasing your mouse button when hovering over a blank area of the gray background*
- *When you let go, a list of executable actions will pop up, and you can search for the function we need.*
    - We are going to save time by using a new specialized node called "Run Light Sensor with Threshold." This new node is like the **Run Light Sensor** node, though it contains the conditional statement built-in. It will run the light sensor and you will only need to change the **Threshold** value within the node, see Figure 3-18
- *In the pop-up box, type in "run light sensor with threshold" to search for it, then select the **Run Light Sensor with Threshold** node*

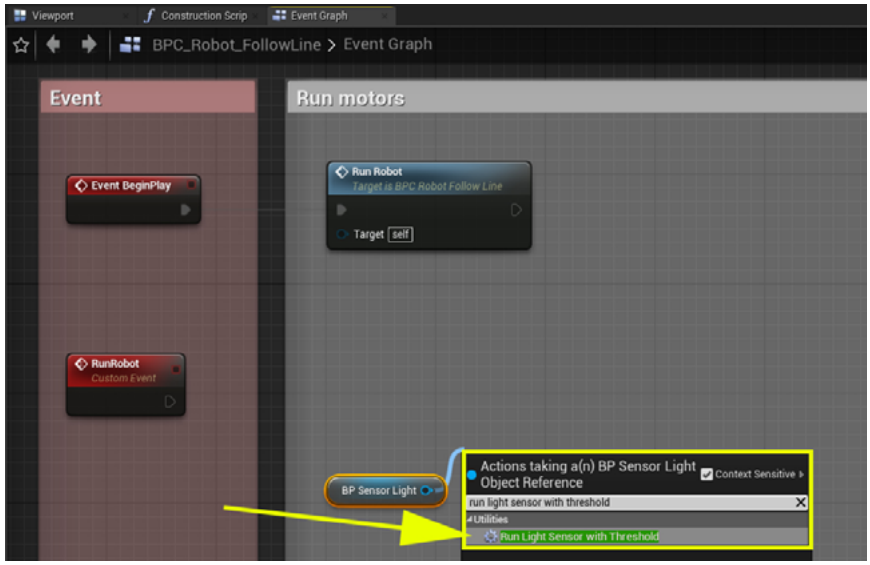


Figure 3 - 17

- *This will place a **Run Light Sensor** with Threshold node in the **Event Graph**, with a blue wire connecting to its Sensor input pin from the **BP_Sensor_Light** node*

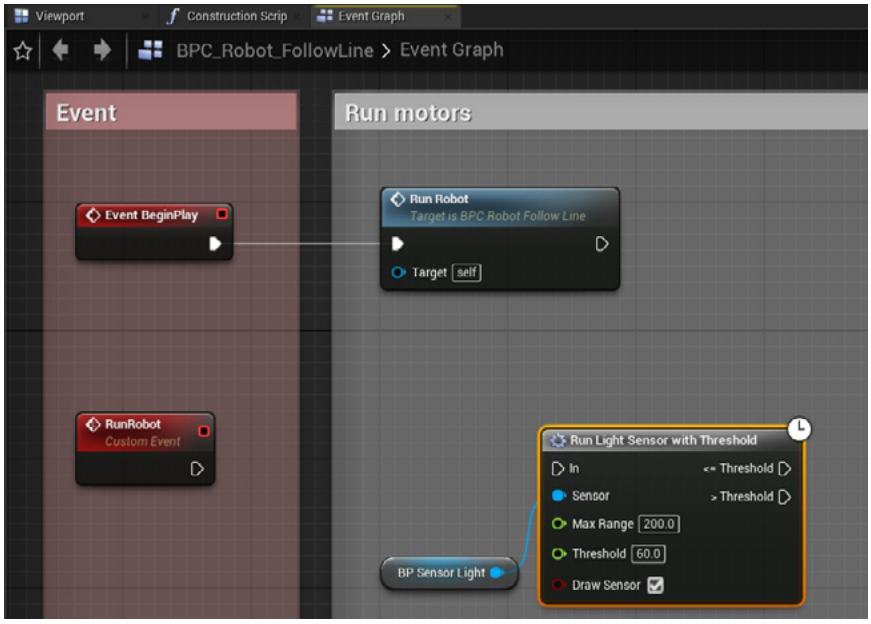


Figure 3 - 18

- *Last, we need to connect the **Run Robot** node to the **Run Light Sensor with Threshold** node, by dragging a wire from the white execution pin of the **Run Robot** node to the **Run Light Sensor with Threshold** node's input pin (also white)*

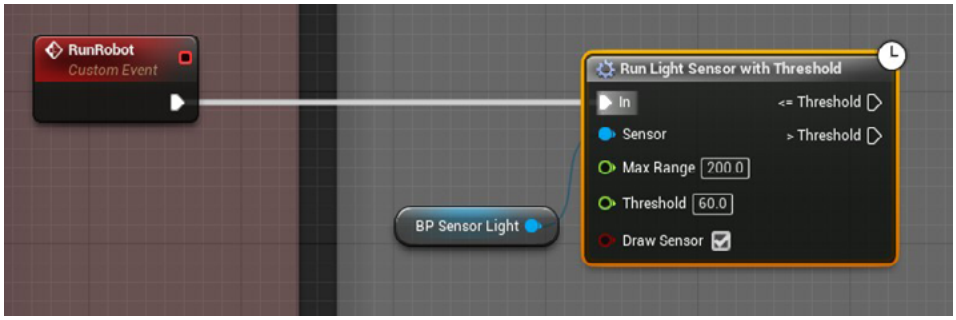


Figure 3 - 19

This process has placed the sensor node into the code and told it to turn on when we play the game. However, our current code will run these commands and immediately stop. We will not be able to see what the sensor is detecting. To observe what the sensor is detecting, we need to create a loop

Pro Tip

If you need to disconnect a wire, hold the Alt key and Left-Mouse click on the wire or on the pin.

If necessary, toggle ON the ability to observe the lines this sensor is using to "see" with:

- Click the option box next to **Draw Sensors** to toggle ON the visibility of the line traces
- It will place a checkmark in the Draw Line Traces option box

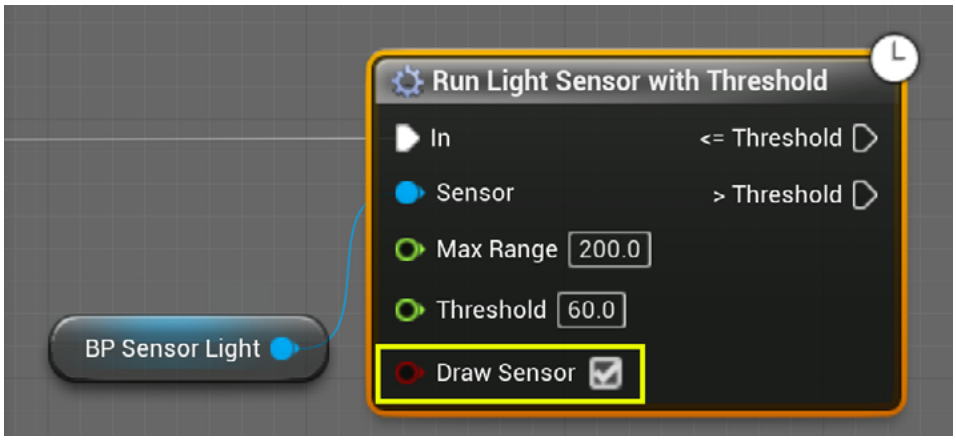


Figure 3 - 20

We are going to copy the **Run Robot** node from the Run motors comment box to place a duplicate of it in a second position in our code:

- In the gray Run motors comment box, click on the **Run Robot** node (to select it)
- Make a copy by pressing the **keys Ctrl + C**
- Paste it, by pressing **Ctrl + V** (this will place the copy below the original node)
- Then drag the new copied node to the right of the **Run Light Sensor with Threshold** node
- Next, to connect them, drag a wire from the **<= Threshold** pin to the input pin in the new **Run Robot** node
- And drag a wire from the **> Threshold pin** of the **Run Light Sensor with Threshold** to the input pin in the new **Run Robot** node to connect both thresholds to the same input on the Run Robot node
- Your code will call the **Run Robot** node after running the **Light Sensor with Threshold** to create a loop.

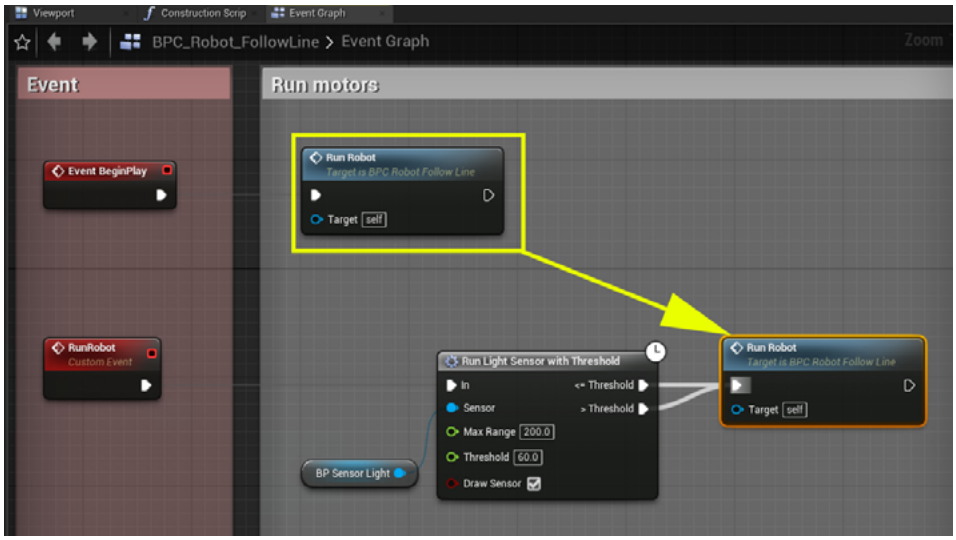


Figure 3 - 21

- Your code should now look like this:
- Now you can see the intensity values change when playing the game

- *Remember to **Compile** and **Save** every time you add code!*

Let's play the game and see how it works.

- Click the **Map_3-1_FollowLine** tab

Getting the value of the white border:

- In the **Viewport**, use the **Move Gizmo** to move the robot manually left or right so that the sensor is on the left side of the black line, and still above the path

Figure 3 - 22

- Click on **Play** or press **ALT P**
- Observe where the lines from the sensor are pointing and the number displayed in the "Light Sensor Feedback" meter above the robot

The Light Sensor Feedback meter displays as a bar above the robot. The intensity meter will display as a purple checker bar being covered with gradient shading from black (0) through gray to white (100) to cover the purple checker bar, as it sees reflected light.

1. It will show more of the purple checker bar if it sees less light and is returning a lower value
2. It will show more gradient gray to white if it sees more light and is returning a higher value

- Log your sensor feedback values and note where the sensor is looking
- Stop the playing of the project by clicking the Stop button or pressing the Esc key
- Repeat this process to log the following values:
    - The value when the sensor is looking at the black line
    - The value when the sensor is looking at the white border

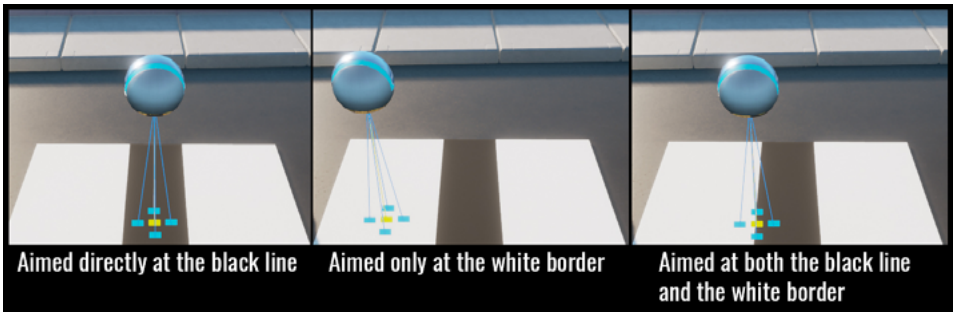


Figure 3 - 23

- The value when the sensor is seeing both the black line and the white border

- **Compile** and **Save** your Blueprint.

Students and Teachers Note: If you are having trouble getting your robot to return a large variance between the white border and the black line, you may have the sensor located too high. You can adjust the sensor's placement within the robot's Blueprint Viewport and try again.

Alternatively, you can open a different robot that has the light sensor already positioned into the best placement, and rebuild your code within that robot's **Event Graph**. To do this:

- From the **Content Browser** panel, open the **Robots** folder, then the **L3_Robots** folder
- Select the robot named **BPC_Robot_FollowLine**
- Then click and drag it from the **Content Browser** into the level
- If you get an **Actor Placement Warning**, click OK. This warning will not affect the lesson

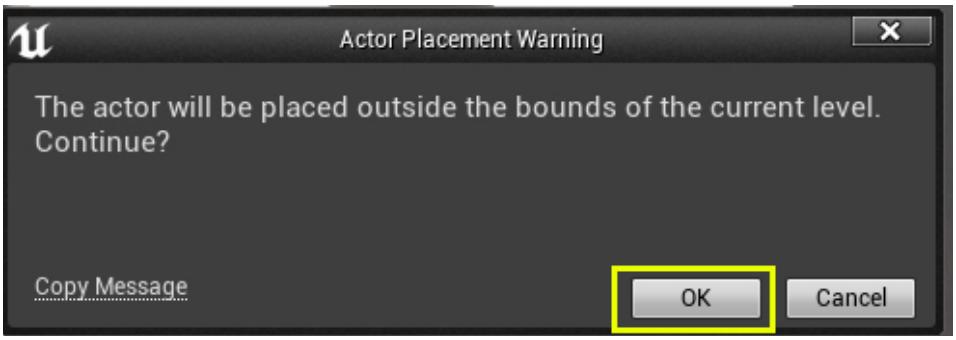


Figure 3 - 24

- Select this robot in the **Viewport**
- In the **Details panel, click Edit Blueprint**, then **Open Blueprint Editor** (your screen should now display the Event Graph inside the new and active Unreal tab of **BPC_Robot_FollowLine**
- Return to Exercise 2 at Figure 3-17 and work forward to build your code for this robot

Exercise 3: Determine the Threshold Values Needed

What is a threshold?

In our project, the value of the sensor can be between 0 and 100. The robot will need to decide between two different actions based on the sensor feedback at any given time while driving.
The threshold is a single numeric value that will tell the robot's conditional statement to select an action to perform. The condition asks whether the sensor has seen a value that is greater than or equal to (>=) the threshold number—in which case it triggers one action—or the sensor has seen a value less than (<) the threshold number, triggering a different action. Thus, the threshold is our "limit" used by the robot to decide if it is on the white border or the black path line.

We need to determine the following answers:

- How do we use the light sensor to get the feedback from what the robot "sees?"
- What value is shown if the line is detected?
- What value is shown if the line is NOT detected?

Note: Be sure that you have a notepad or a shared documenting system for your school and class to log light sensor values as you proceed through the following steps.

Now, we need to establish the light sensor threshold value needed for consistently seeing the black portion of the path. Having noted the values of the black path line versus the white border (ground), we need to establish a threshold value that divides the full range of sensor input into “seeing the white border” and “seeing the black path line.”

- Refer to your notes where you logged the White border/ground and delivery path line values
- Determine the threshold value with this formula: **((High – Low) / 2) + Low = Threshold**
    - So, if the High value (the white line border) = 90, the Low value (the black path line) = 10, then the formula will read **((90-10)/2)+10 = 50**
- Observe and log your Threshold value

We have included a Threshold calculator in the project. It can be accessed by navigating through the Window menu at the top left of the interface.

Window > Editor Utility Widgets > Threshold_Calculator

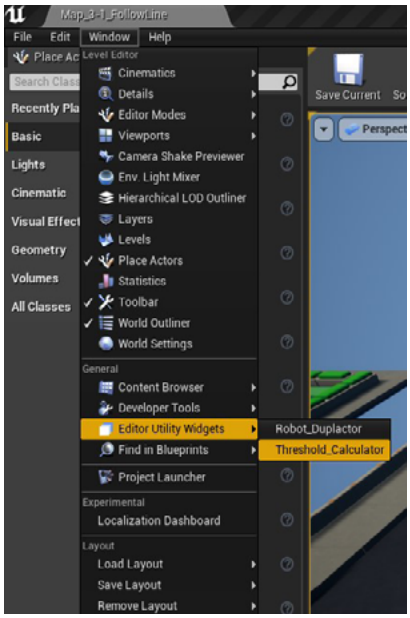


Figure 3 - 25

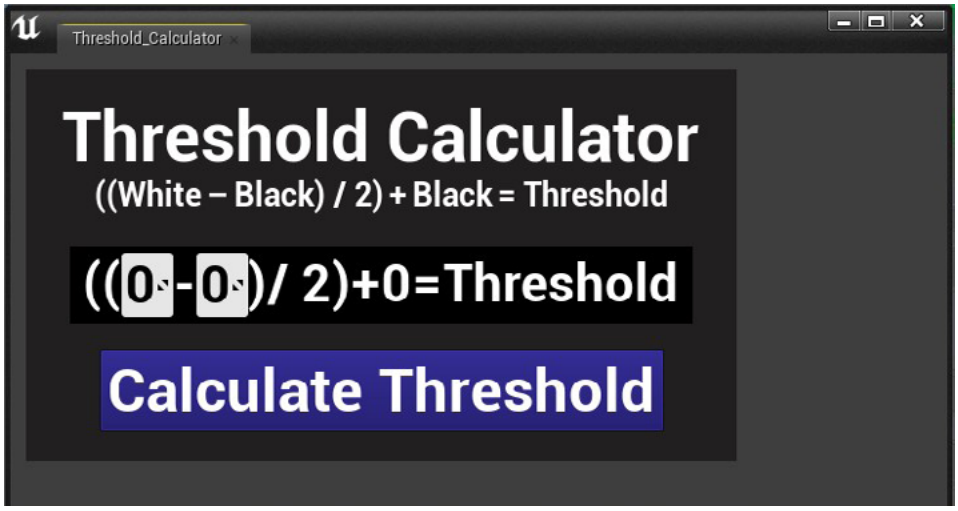


Figure 3 - 26

Exercise 4: Finding the Line

We will test the light sensor input, and have it make decisions based on its current feedback. Then we will tell it to run the motors depending on that feedback. We need to add in some motor commands to make the robot move and turn on its own.

Aligning the Robot to the Delivery Path Line

Notice the direction the robot is facing in reference to the line before we begin moving it. This will determine the choices we make when coding the movements. Position your robot towards the path, with the light sensor on the white border to the left of the black line, for this part of the exercise.

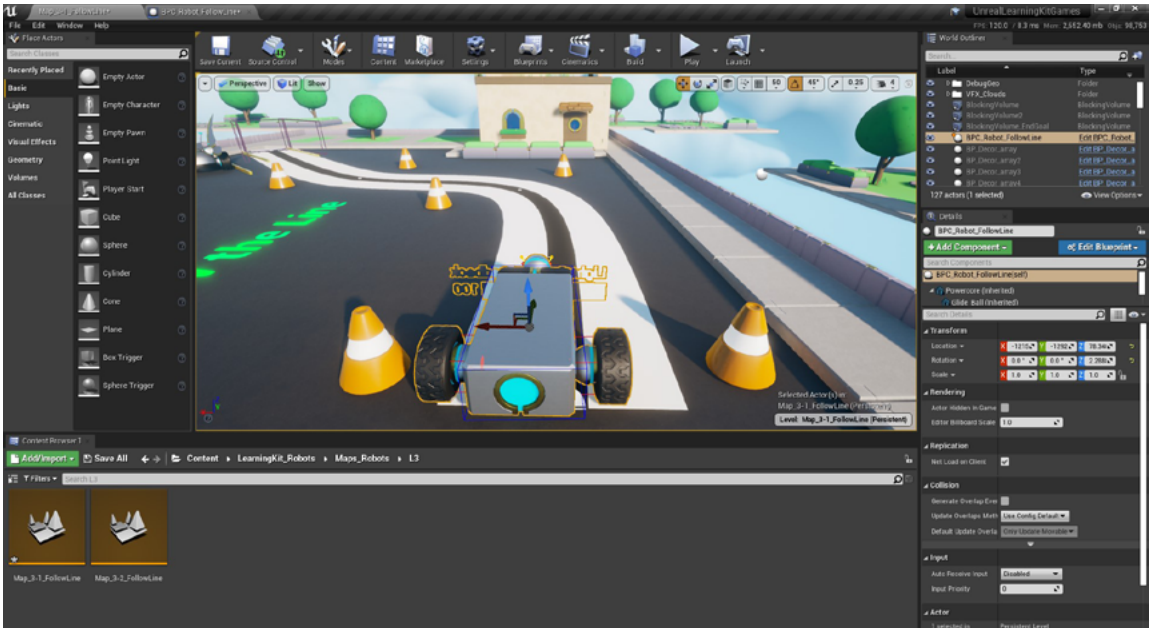


Figure 3 - 27

Responding to Sensor Input

When the robot can read the input values from the sensor, it can perform a specific action based on the feedback from the sensor. We first want it to decide that if it sees the line, then it should arc turn away from the line.

Activity: Now we will set the threshold value so that it will give feedback when it sees (or doesn't see) the black line. If the input value is higher than the threshold, it is seeing the white border.

Open the robot in the Blueprint Editor.	
If you already have the Blueprint open	**If you don't have the Blueprint open**
Switch to the tab labelled: **BPC_Robot_ FollowLine**	• Select the robot (you should see **BPC_ Robot_FollowLine** displayed as our active Actor in the right-side Details Panel). • In the **Details panel**, click **Edit Blueprint**, then **Open Blueprint Editor**

- If you are on the Viewport of the Blueprint, click the **Event Graph** tab
- Your screen should now display the **Event Graph** inside the Blueprint editor of **BPC_Robot_FollowLine**

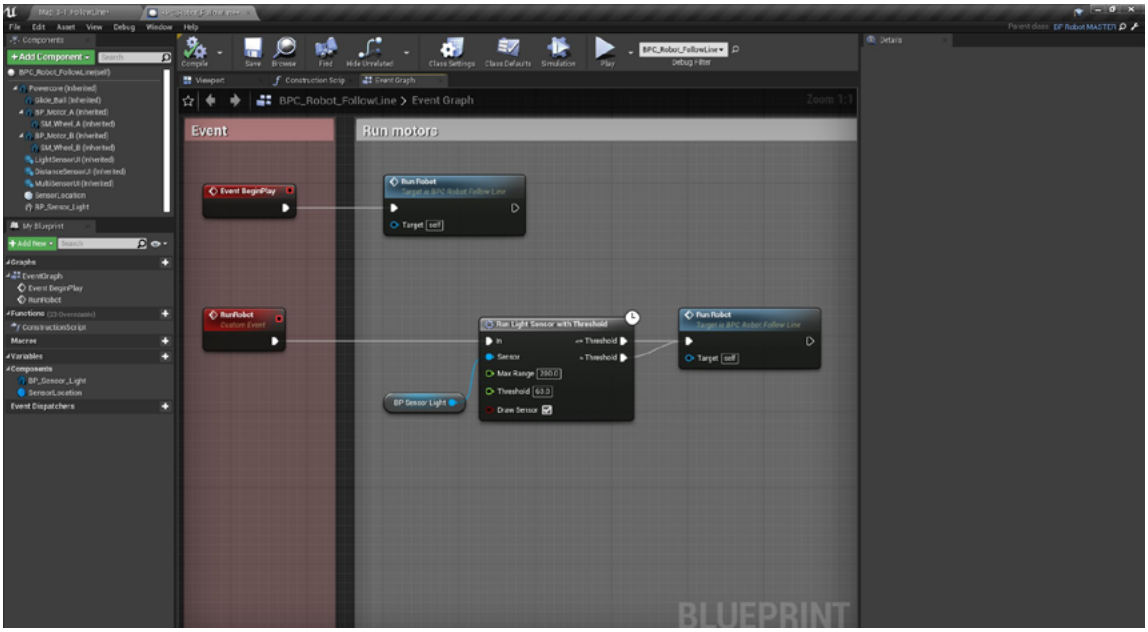


Figure 3 - 28

Edit our previous code to make room for new commands:
We need to make space available in the code to be able to place new commands between the Run Light **Sensor with Threshold** node and the **Run Robot** node at the end

- Click and drag the Run Robot node and move it further to the right (see Figure 3.29)

We need to break the wires between the **Run Light Sensor with Threshold** node and the **Run Robot** node at the end of our code. We will be adding new nodes and connecting them to different commands

- To break the wires connected to the **Run Robot** input pin, hold the **Alt** key on the keyboard and left-mouse click the wire(s) or the white execution pin of the **Run Robot** node

Your Blueprint should now look like this:

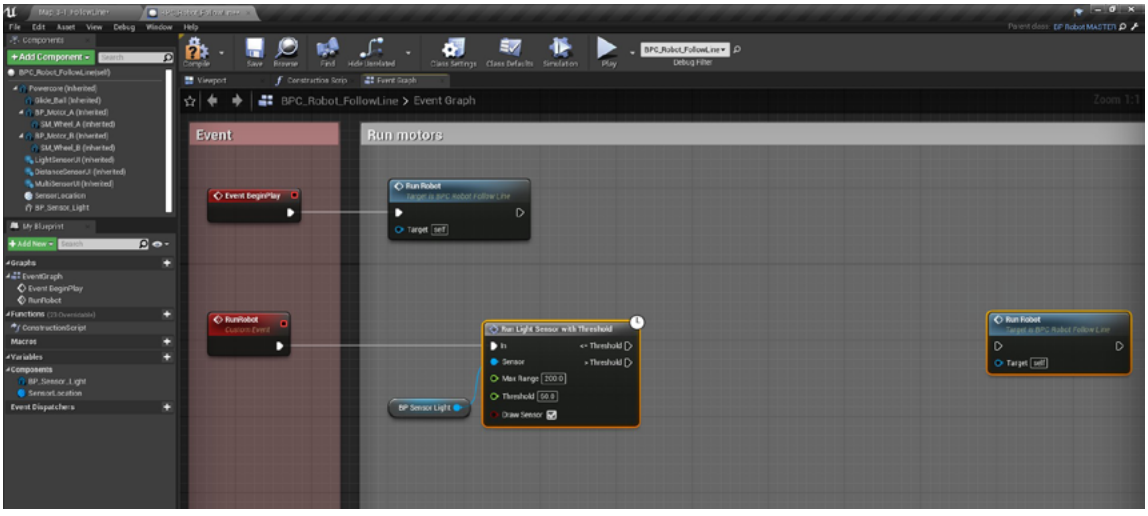


Figure 3 - 29

Motor Commands

- The **Run Motor** node controls the motor connected to it. Figure 3-30 shows the Blueprint setup for making the **BP_Motor_A** component turn. The **Speed** input controls how fast the motor turns, positive numbers make it spin in one direction, and negative numbers make the motor spin in the opposite direction. It takes values from –100 to +100

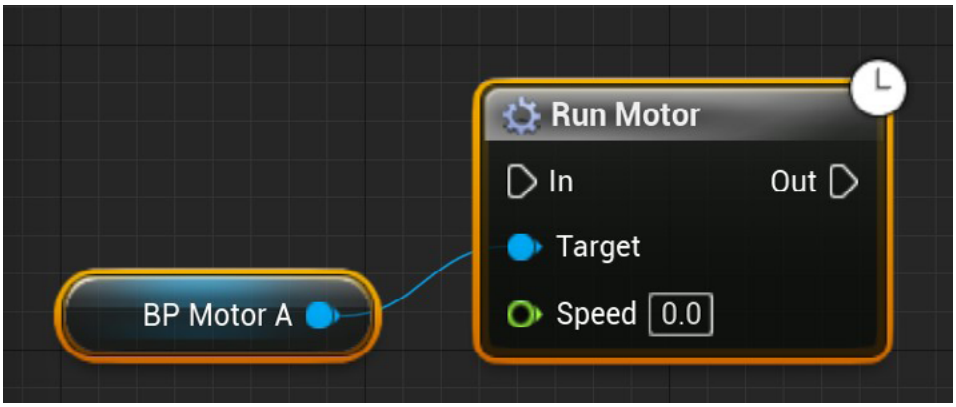


Figure 3 - 30

- The **Stop Motor** node controls the motor connected to it. Figure 3-31 shows the Blueprint setup to make **BP_Motor_A** stop if it is turning. The **Coast** input will tell the motor to coast slowly to a stop if toggled ON or stop turning immediately if toggled OFF. In Figure 3-31 the motor will stop immediately, as it is toggled OFF

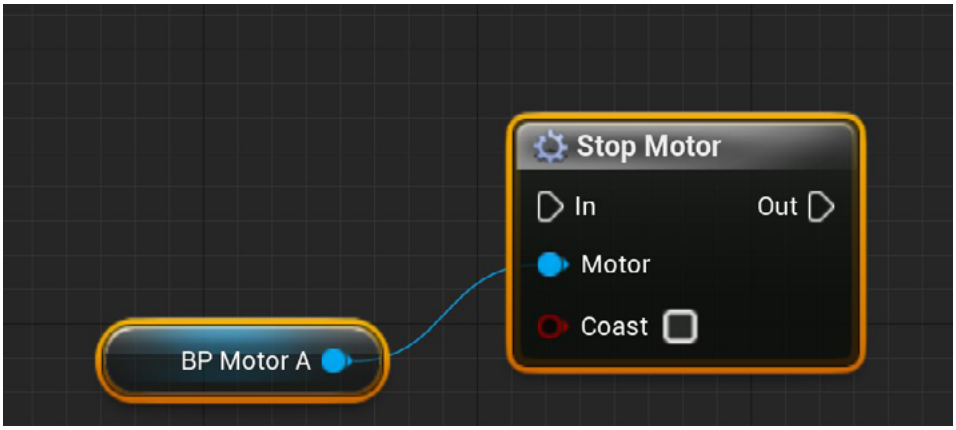


Figure 3 - 31

Action when seeing the white border

When the input value is higher than the threshold, it is seeing the white border (ground)). We want the robot to look for the black line continuously, so if it is on the white border or anywhere off the black line, we want it to turn toward the black line.

Coding the robot to turn toward the black line

Our robot is currently placed slightly left of the black line, on the white border.
Since we want the robot to turn right (towards the black line), we need the left motor (Motor A) to move forward and the right motor to stay still.

Let's first add the **BP_Motor_A** node:
From the **Components panel**, **click** the **BP_Motor_A** and **drag** it into the event graph

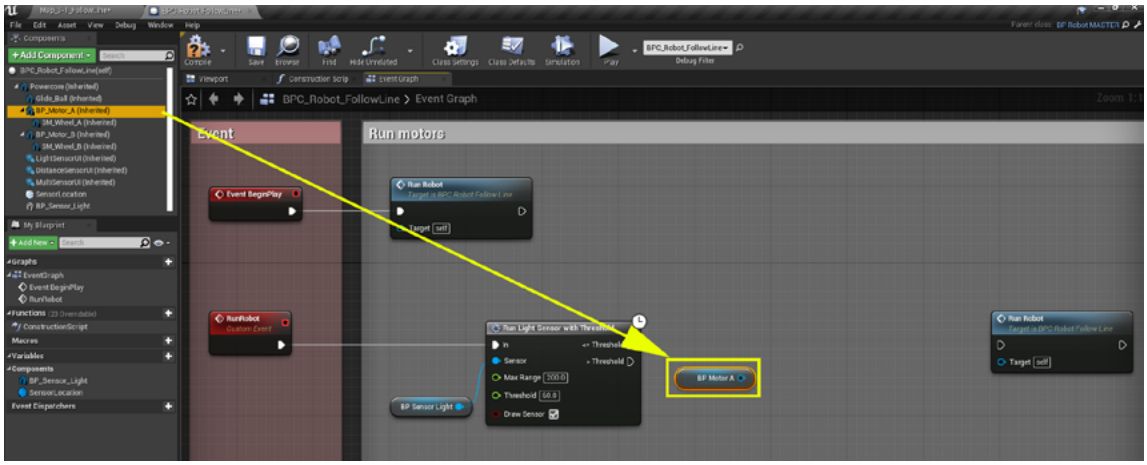


Figure 3 - 32

Next, **click and drag** from the **BP_Motor_A** blue output pin to the right.

- Search for (type) "run motor", and select the **Run Motor** node

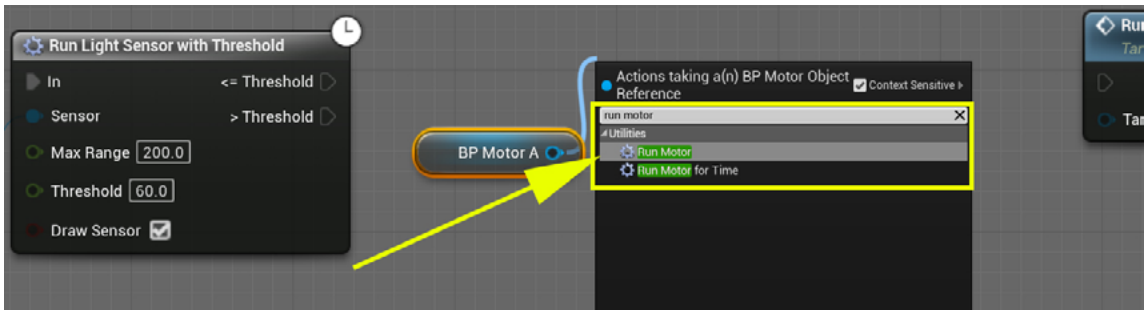


Figure 3 - 33

- Remember, we are seeing more reflected light when the robot is NOT on the black line, therefore, the light sensor input value is greater than the threshold value
- Connect (click and drag) the white > **Threshold** pin of the **Run light sensor with Threshold** node to the **Run Motor** input pin

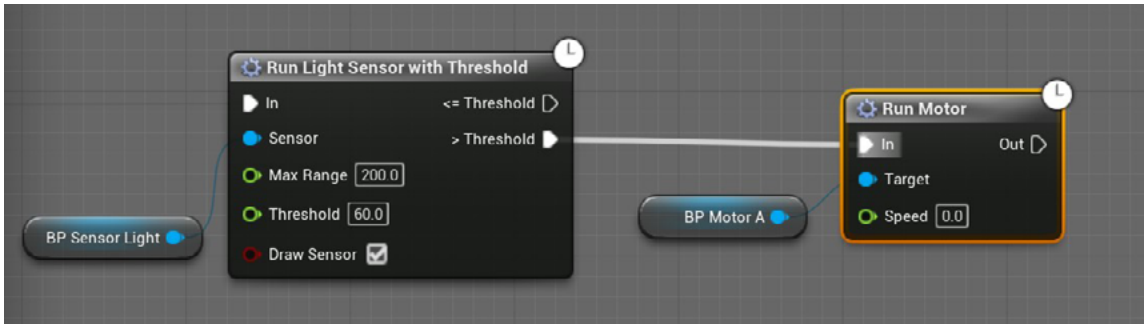


Figure 3 - 34

- Set the speed of the **Run Motor** node to a value between 0 and 100 (We recommend a speed of 30) to turn the motor in a direction that will push the robot forward. This will also turn the robot toward the black line
- From the **Components Panel**, click and drag **BP_Motor_B** to the right of the **Run Motor** node, to place it into the flow of code as shown in Figure 3-35

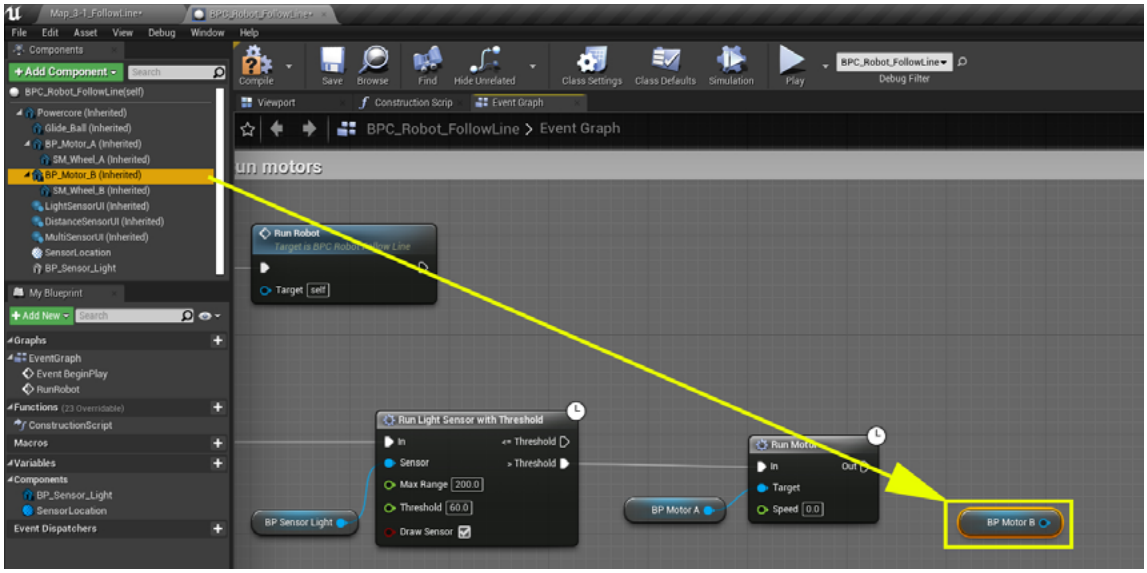


Figure 3 - 35

- **Click and drag** from the **BP_Motor_B** blue output pin to the right
- Search for (type) “stop motor”, and select the Stop Motor node

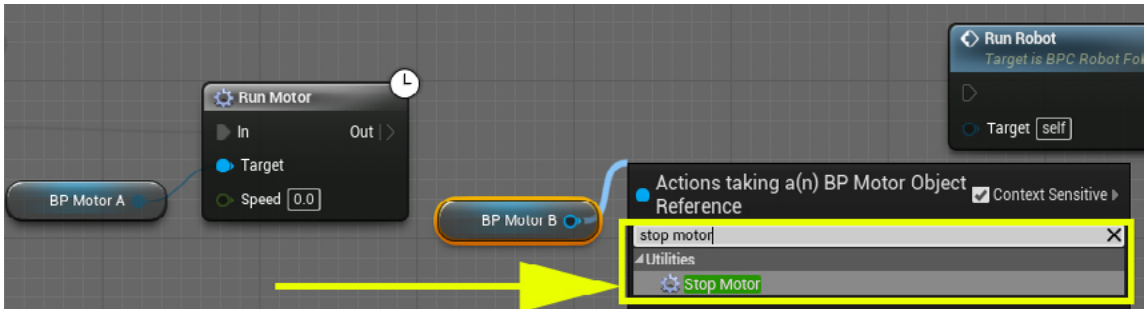


Figure 3 - 36

Connect the Run Motor to the Stop Motor.

- **Click and drag** a (white) wire from the **Run Motor** output pin to the **Stop Motor** input pin

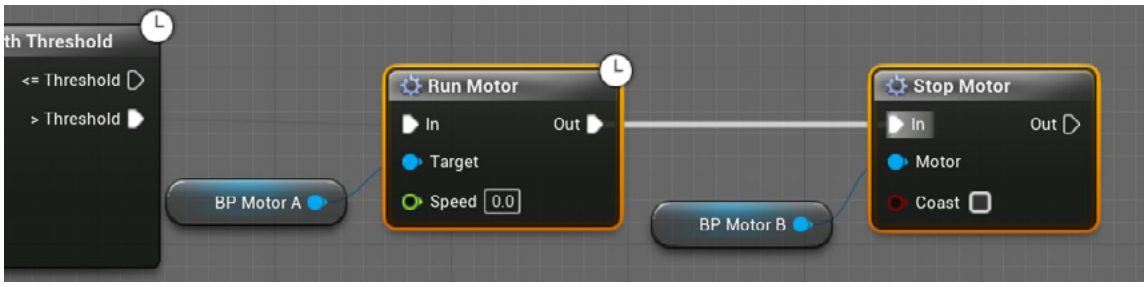


Figure 3 - 37

Connect the **Stop Motor** node to the copied **Run Robot** node at the right:

- **Click and drag** a (white) wire from the **Stop Motor** node's output pin to the **Run Robot** node's input pin

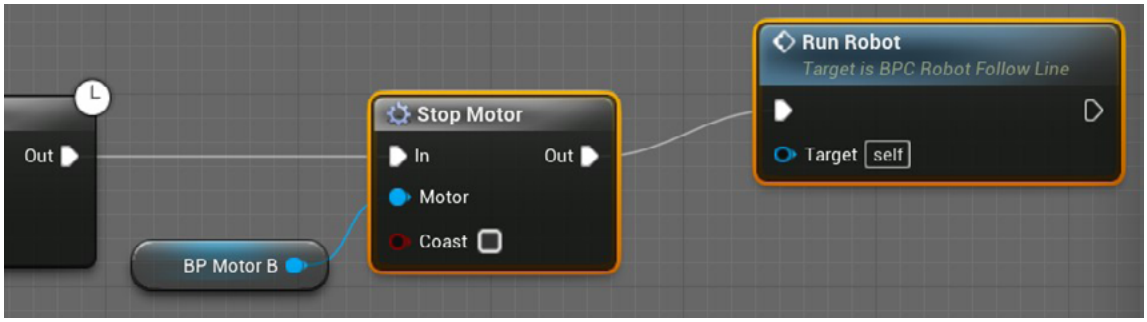


Figure 3 - 38

Create a comment box for the four new commands as a group:

- **Click and drag** around all the four nodes with the lasso or marquee tool to select all of them at once at once
- Press the **C** key to create a comment box around the selected nodes
- Type in "Turn toward line" and press Enter
- This will name your subsection of code

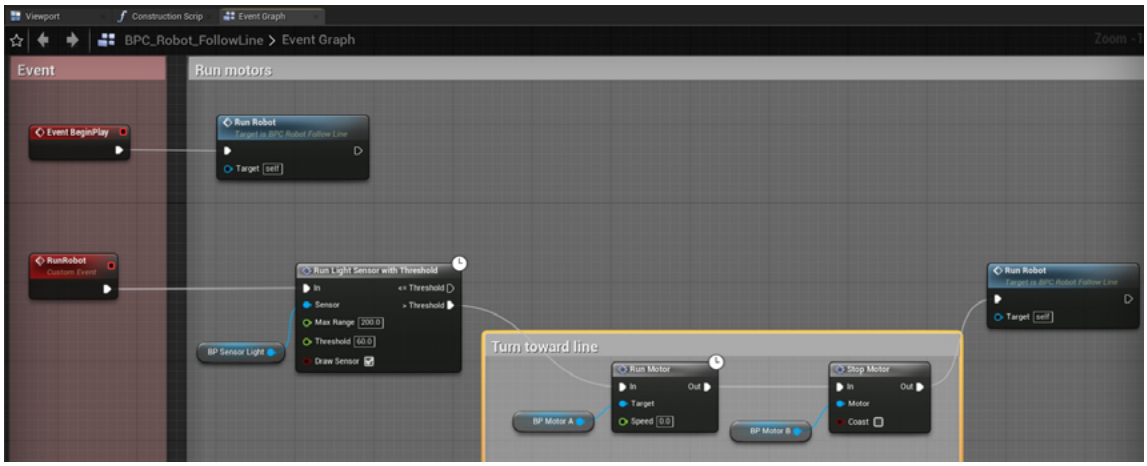
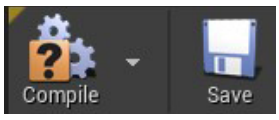


Figure 3 - 39

- **Compile** and **Save**



- Click on the **Map_3-1_FollowLine** tab to return to the map
- Position your robot to be at the beginning of the path——but left of the black line—on the white border
- Press the **1** key to zoom out to see the entire map
- Play the game

A Note on Performance

The performance of these robots is tied to the performance of the computer you are working on. You can toggle a display mode that shows how many frames per second (fps) the project is running at by pressing c **Ctrl + Shift + H**. The development of these lessons was created on a computer that can reach 120 fps. If your robot is not running as expected, it could be due to the performance of your computer. Don't get discouraged, you can close other applications on the computer to help get better performance, and you can always try again and continue to gather more data.

- Observe, test, and debug to make sure your robot turns towards the black line when it sees the white border or any area that is NOT the black line

Exercise 5: Completing the Branch

When you observe the robot's actions note that it is continually turning towards the black line, and does not yet know what to do when it sees it. We want to look now for a sensor input that is less than or equal to (<=) our threshold value when it sees the black line. We will build the other side of our branch.

Action When Seeing the Path (Black) Line

Next, we will add nodes to test if the robot sees the black path line; connect the sensor output to the new branch; add motor functions, and change the motor speeds to turn in the proper direction.

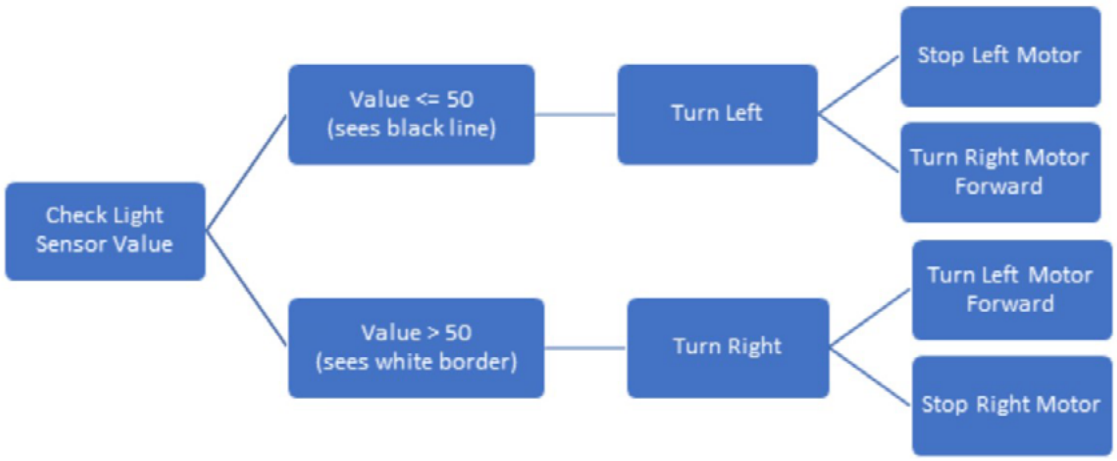


Figure 3 - 40

Review the Logic Diagram Above. If our sensor returns a value that is <= our Threshold (50), it is seeing the black line. What do you want to have happen when it sees the black line? While we may think we want it to drive forward along the line, we are going to make the robot "wiggle" along the way by moving away from the black line, then turning back towards the black line, and thus continuously moving forward using the line as a guide. To do this, when it sees the black line, we want the robot to turn left, or away from it.

We are going to quickly make the other branch with all the nodes by duplicating the current subsection of code from the first branch, and then editing it to move in the other direction when the sensor sees a value less than or equal to our threshold value.

Open the Blueprint Editor

- **Click** the robot in the **Viewport** (you should see **BPC_Robot_FollowLine** displayed as our active Actor in the right-side Details panel)
- In the **Details panel**, **click Edit Blueprint**, then **Open Blueprint Editor**
- If you are on the Viewport still, click the **Event Graph**
- Your screen should now display the **Event Graph** inside the **BPC_Robot_FollowLine** Blueprint

Duplicate the previous commented code using copy and paste:

- Select the entire **Turn toward line** comment box (click and drag around the entire turn toward line subsection. Be sure that all five elements (nodes and comment box) are now framed in orange to show that they are selected)
- ***Tip:*** *you can also use* ***shift click*** *to add to any selected node to make sure you have all of them selected*
- **Right-click** the selected nodes and select **Copy** from the pop-up dialog box (or press **Ctrl + C**)
- Move your mouse cursor to the upper blank gray area of the Blueprint
- **Paste** the duplicate sub-section by pressing **Ctrl + V**. This will paste a copy above the original code
- Click and drag the duplicated nodes as needed to line it up with the original code box (see Figure 3-41)

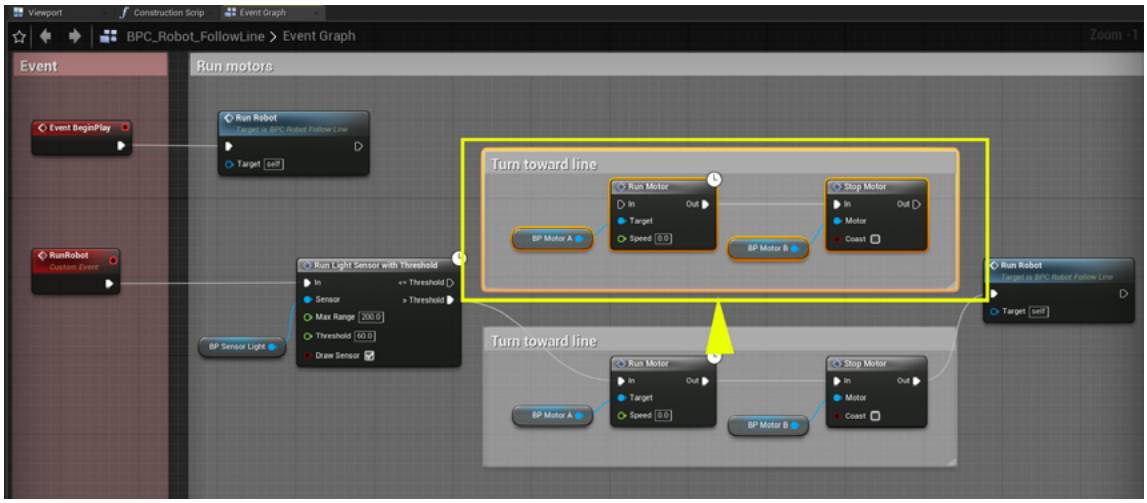


Figure 3 - 41

Rename the new comment box:

- Click anywhere in the gray Blueprint area to unselect the new section
- **Right-click** on the new upper box to select it
- Select **Rename** from the dialogue box
- Type in “Turn away from line”, press **Enter**
- This will rename your code box so that the two boxes have unique names (the upper box is named “Turn away from line” and the lower box is named “Turn toward line”)

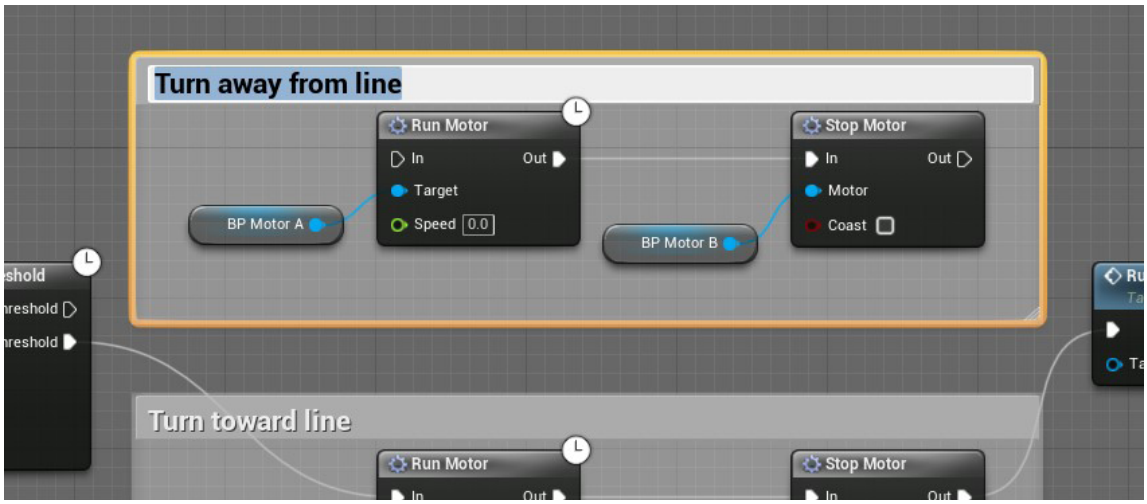


Figure 3 - 42

To quickly and easily reverse the direction that our robot will turn, to turn away from the line, we will transpose the **BP_Motor_A** and **BP_Motor_B** nodes:

- Hold the **ALT** key and click the blue wires connected to the motors in our "Turn Away From Line" box
- Click and drag the **BP_Motor_B** to the left of **Run Motor** node
- Click and drag the B**P_Motor_A** to the right of the **Run Motor** node
- Your code should look like Figure 3-43

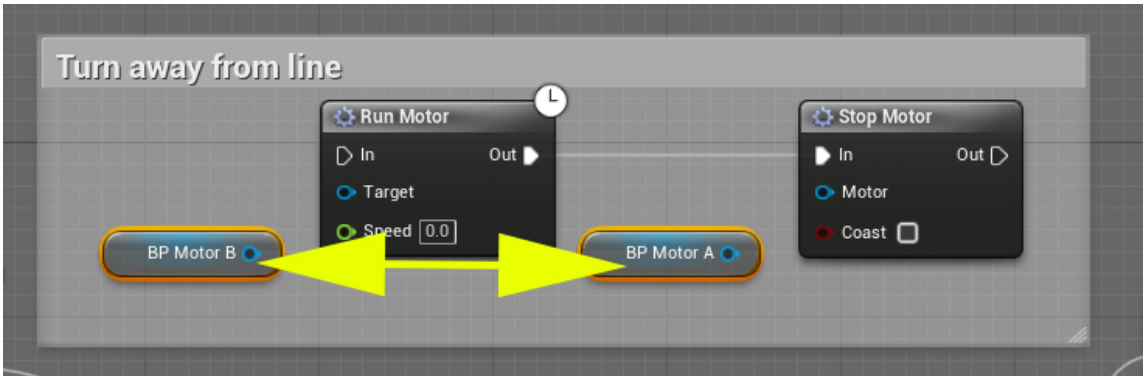


Figure 3 - 43

Hook them back up:

- Click and drag a (blue) wire from the **BP_Motor_B** pin to the **Run Motor** node's Target pin
- Click and drag a (blue) wire from the **BP_Motor_A** pin to the **Stop Motor** node's Motor pin (see Figure 3-44)

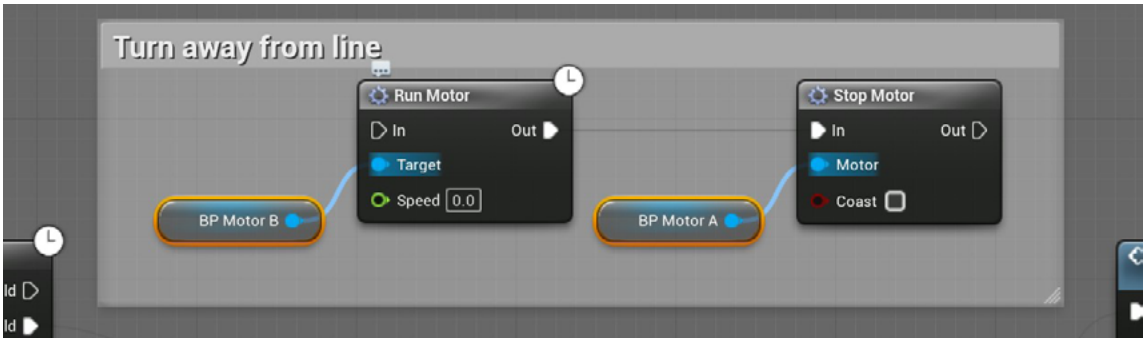


Figure 3 - 44

- Set the speed of **BP_Motor_B** to a value between –100 and 0 by clicking into the **Speed** text box and typing in a number of your choice. (We recommend -30. This should be the same speed as the other **Run Motor** node, except in the opposite direction)
- Remember that you need to have a negative value to move forward with **Motor B** since it is flipped

Hook up all the code:

- **Click and drag** a (white) wire from the **Run Light Sensor with Threshold** node's **<= Threshold** pin to the **Run Motor** node Input pin
- **Click and drag** a (white) wire from the **Stop Motor** node output pin to the **Run Robot** node input pin

Look at your code to be sure that it looks like Figure 3-45:

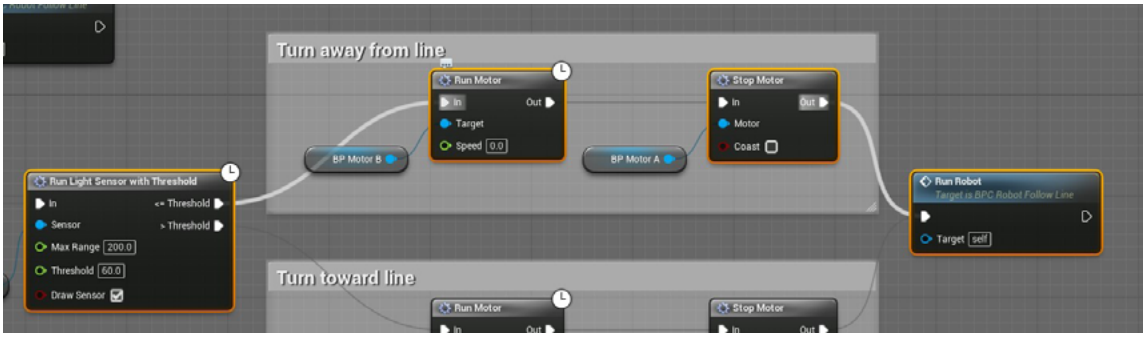


Figure 3 - 45

- **Compile** and **Save**

- Your code should look like Figure 3-46. You can experiment with the speed values and the choice of **Coast ON** or **OFF** as you test your robot

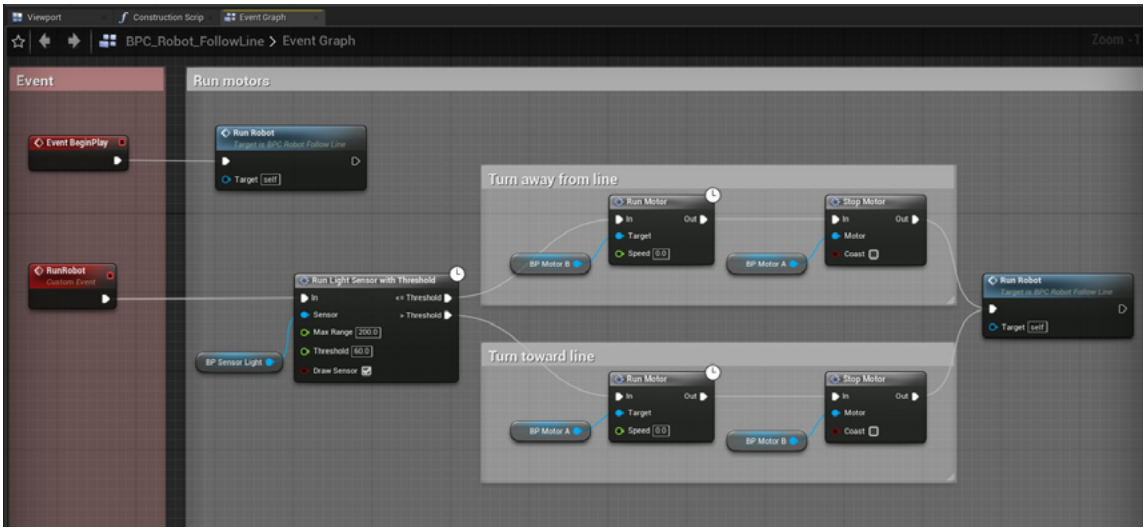


Figure 3 - 46

- *Remember to **Compile** and **Save** every time you add code!*

- Click the **Map_3-1_FollowLine** tab to return to the map.
- Position your robot to be at the beginning of the path, left of the black line, on the white border
- Press **1** key to zoom out to see the entire path
- Play the game
- Experiment with changing the speed values and test your code to see how well your robot performs the correct actions accurately

Observe the results:

Try the robot several times, noting your robot starting position each time you play it, and then note your robot's actions in relation to where it started from. Remember to try it two or more times, as actions will vary slightly.

CHALLENGE

Continue to refine your sensor location on the robot, the robot's starting position, and the code until your robot can follow the path line all the way to the end of the path.

- Troubleshoot by editing the code until your robot is successful in moving away from the line if it sees it. Use the following questions to guide your debugging:
    - What happened when the robot approached the line? Did it stay on the line?
- Make corrections until the robot turns away from the line if it is on it
    - What happened when the robot left the line? Did it look for the line again?
- Make corrections until the robot turns towards the line if it is NOT on the line
    - Did the robot move too fast and get lost along the way? Is it turning circles far away from the path?
- Did it move forward along the line?
- Find the values that give the most reliable results

Hints:

- If you are having trouble, try lowering your speeds so the robot can turn more accurately
- Experiment with the robot's starting position, left to right, and including the light sensor's height above the path

Note: The robot will stop at its destination. This activity's current robot model will stop when it hits the Blocking Volumes box, which is an invisible wall. We do not have to code this action.

EXTENSION ACTIVITIES

Multiple Light Sensors

How can we make the line-following action a smoother motion (without a lot of "wiggling" back and forth)? Consider the use of two sensors to make the movement smoother.

Where should each of the sensors be placed?

- Move the first light sensor
- Place a second light sensor
- Make sure to calculate the thresholds when adding or moving sensors

If we use a second light sensor, how will the code change for each sensor's instructions?

- Can we reuse any of our existing code?
- Could we now have three states? For instance, See line with left sensor, turn left; see line with right sensor, turn right; and see no line, go straight.

Sensor Locations for Using Two Sensors:

Figure 3 - 47

Speed

- Speeds will be up to the user to work out
- How fast can you go?
- What new problems arise when adding speed?

New Coding Technique

- Learn about using a sequence command for running two sensors simultaneously, or close enough to the same time to make them both run at the same time, rather than one at a time

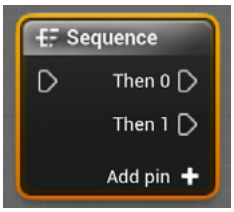


Figure 3 - 48

- Conditional statement using the threshold test of a light sensor

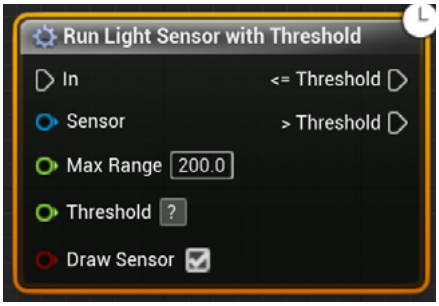


Figure 3 - 49

- Looping settings: where to start, where to end, how to run for unlimited repetitions

Review

- Understanding the need for sensors, and how they analyze input
- Understanding sensor effectiveness when placed in different positions on the robot
- Understanding the use of a threshold to convert a range of sensor values. The sensor then uses a Boolean to execute along a branch
- Understanding coding commands that help sensors determine robotic movement
- Coding connections: conditional statement using the threshold test. Using loops to repeat actions
- Notes and demonstrations of challenges, success, and understandings

RESOURCES

- Observation log – a blank one, and one for the Extension Activity

ASSESSMENT

Rubric

Concept	Distinguished	Proficient	Competent	Developing
Robot Design concepts	Can explain robot sensor components and how they work to others. Project. Project demonstrates use of sensors in lesson, and the best positions on the robot. Definitions of components are exhibited, by demonstration or in student documentation.	Demonstrates use of sensors on the robot. Full understanding of robot components is indicated to teacher.	Can make robot respond to sensors, does not provide meaningful understanding of sensors. Basic understanding of hardware components demonstrated.	No evidence of understanding robot sensors and how they function. No understanding of hardware components demonstrated.
Software concepts	Complex command combinations and project goals demonstrated. Can explain command groups needed for using sensors, sensor position, and desired resulting robot position or action. Commands or groups of commands are demonstrated by presentation or documentation.	Student understands command groups needed for using sensors, sensor position, and desired resulting robot position or action. Commands or groups of commands mentioned in student's presentation.	When prompted, student has basic understanding of commands needed for using sensors, sensor position, and desired resulting robot position or action, but may not be presented in demonstration.	No evidence of understanding commands and how they function. No inclusion or mention of commands in student's presentation.
Coding concepts	Can debug code errors. Complex code combinations and unique use demonstrated. Can explain codes from most sections. Codes are included in student's demonstration or documentation.	Demonstrates understanding of default settings, loops, and conditional commands. Student references at least one of each command type code in their presentation.	When prompted, student has basic understanding of codes, but may not be mentioned in student's presentation.	No evidence of command codes and how they function. No inclusion or mention of codes in student's presentation.
Real-world concepts	Has innovative ideas to create real-world uses of robots, coding, and movement. Demonstrates understanding and documents it.	Understands some use of coding, movement, and robot uses in real-world applications. Includes at least one example in student's presentation.	Basic understanding of existence of coding, movement, and robot in real-world applications. May be explained verbally when prompted, but may not be included in presentation.	No evidence of understanding real-world applications using coding, movement, and robots.
Challenge Activity (Line following robot)	Demonstrates the fully automated line-following robot, demonstrates coding process iterations, and provides documentation of attempts, challenges, and successes.	Coded a line-following robot, with multiple iterations of coding, observation, and testing during the building process.	Basic understanding of how a line-following robot should work. Was able to code some elements of the line--following robot.	Was not able to demonstrate a line following robot. Did not understand how a line--following robot would work.

STUDENT GUIDE

LET'S TRAIN VIRTUAL ROBOTS

LESSON 3: SELF-DRIVING CAR

UNREAL ENGINE