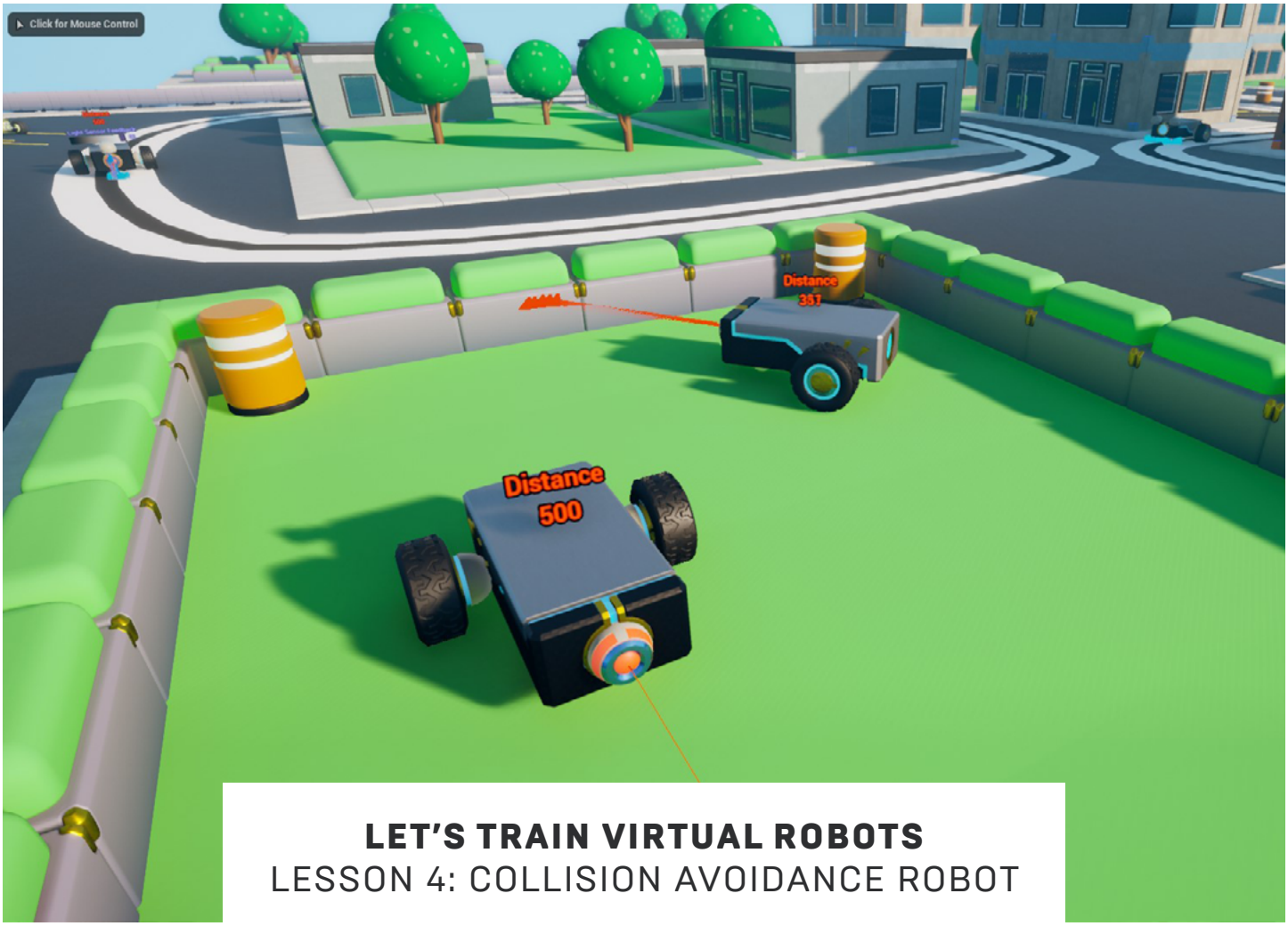
LET'S TRAIN VIRTUAL ROBOTS

LESSON 4: COLLISION AVOIDANCE ROBOT

UNREAL ENGINE

STUDENT GUIDE

TABLE OF CONTENTS



LET'S TRAIN VIRTUAL ROBOTS
LESSON 4: COLLISION AVOIDANCE ROBOT

OBJECTIVE

This activity will challenge students to modify a working autonomous vehicle to consider the importance of safety. We will add a sensor capable of detecting the presence of objects in front of the vehicle. When the vehicle gets too close to an object, it will stop.

This activity takes your skills to the next level by adding another sensor to the vehicle and introducing another conditional statement.

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 FOUR: COLLISION AVOIDANCE ROBOT

Introduction

Smart cars and drones (whether underwater or airborne) use sensors to target objects or avoid collisions. How do these sensors detect objects nearby? Even your iPhone uses a LiDAR sensor to enhance its video focus.[1]

Understanding how distance sensors, like LiDAR, work will help you understand how a robot can detect the distance of an object. Coding the robot to read the distance value will allow us to use a conditional statement to change the robot behavior when an object is too close.

In many real-world robotics applications, a LiDAR sensor is used. A LiDAR sensor (Light Detection And Ranging) is a laser beam sensing method used to examine a surface.[2,3] When used with code, the sensor data can be used to create a 3D model of an object. It can also be used to detect the size and motion of objects. It uses infrared or ultraviolet light to see the area and objects around it. Creating code that uses the input from sensors allows you to create specific solutions to problems.

Lesson Overview

In this activity students will learn how to use a distance sensor to detect objects in the path of our robot. We will program our robot to avoid running into obstacles. The robot will ultimately be programmed to follow a line like we did in Hour of Code LessonThree. This activity will upgrade our robot to stop if an object blocks its path. Capabilities like this could save lives while advancing technology!

All the functions of a LiDAR sensor are beyond the scope of our activities, so we have created a simplified version that measures distance. The distance sensor in the robotics learning activity is similar to a laser distance-measuring device. It will return the distance of the first object in its direct line of sight.

---

[1] LiDAR explained: What this laser tech can do for your new iPhone. Perry, A., Mashable.com 09/08/2020, https://mashable.com/article/iphone-12-what-is-lidar-explained

[2] What is LiDAR? Velodyne LiDAR, accessed 07/14/2021 from https://velodyneLiDAR.com/what-is-LiDAR/

[3] What is LiDAR? NOAA, 2021, National Ocean Service website, 02/26/2021. https://oceanservice.noaa.gov/facts/LiDAR.html

For the collision avoidance task, when the distance sensor detects an object in the path of the robot, it will stop until the object is cleared. When the robot no longer sees an object in its path, it will resume moving forward.

For the robot to receive the information, the flow of code must contain a loop where the sensor continues to test the result and make adjustments to avoid objects and continue moving forward.

The distance sensor will see an object by returning a distance value to the robot; when that distance is a low enough number, the robot will be commanded to take the appropriate action—that is, to stop moving forward, back up, or turn.

This lesson will also explore the benefits of using the light sensor(s) to follow a line, and the distance sensor to avoid obstacles.

Follow the delivery line to your destination, but don't run into anything!

CODING CONNECTION

For the robot to receive real-time information about the environment, the code must contain a loop. Each pass through the loop will get the latest values from the sensor and send the correct commands to the motors based on the current state. This process will happen several times per second, so the robot will be able to react immediately to the environment while it's moving. The robot will continually run until the student stops playing the program.

STARTING THE LESSON

Defining a Purpose

Technology is constantly upgrading to the next best thing. It's time to upgrade our self-driving robot! Instead of solely focusing on the line, we will add the ability for the robot to avoid colliding with objects in its path. To do this, we will need to add a new sensor and modify the code.

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
- Obstacles (barrels) will move in front of the robot at random intervals
- Obstacles (barrels) will move away from the path after a brief delay

Rules

- 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
- The robot will stop driving if it sees an obstacle in front of it
- The robot will resume driving when the obstacle is removed from its path

Requirement

- Uses a distance sensor to “see” obstacles along the path in front of it
- Uses a light sensor to “see” the white border or the black line
- Stops when seeing the obstacle and resume movement when the obstacle leaves the path.
- Follows the line we coded in Lesson Three:
    - 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: Learning about Distance Sensors

The robotics learning activities includes a distance sensor that returns the distance of the nearest object in the direct line of sight from the sensor. The sensor has a maximum effective range, so it can only detect objects within a specific distance. A distance sensor is typically used to tell a robot how to see an object in its path and continue moving without hitting obstacles.

Activity: Play the robot in Map_4-1_Distance to see a demonstration of the various functions of the distance sensor and an example of how it could be used in our Robot Town.

- In the **Content Browser**, navigate to the **Content -> LearningKit_Robots -> Maps_Robots** folder
- **Click** on the **L4** folder
- Open **Map_4-1_Distance**

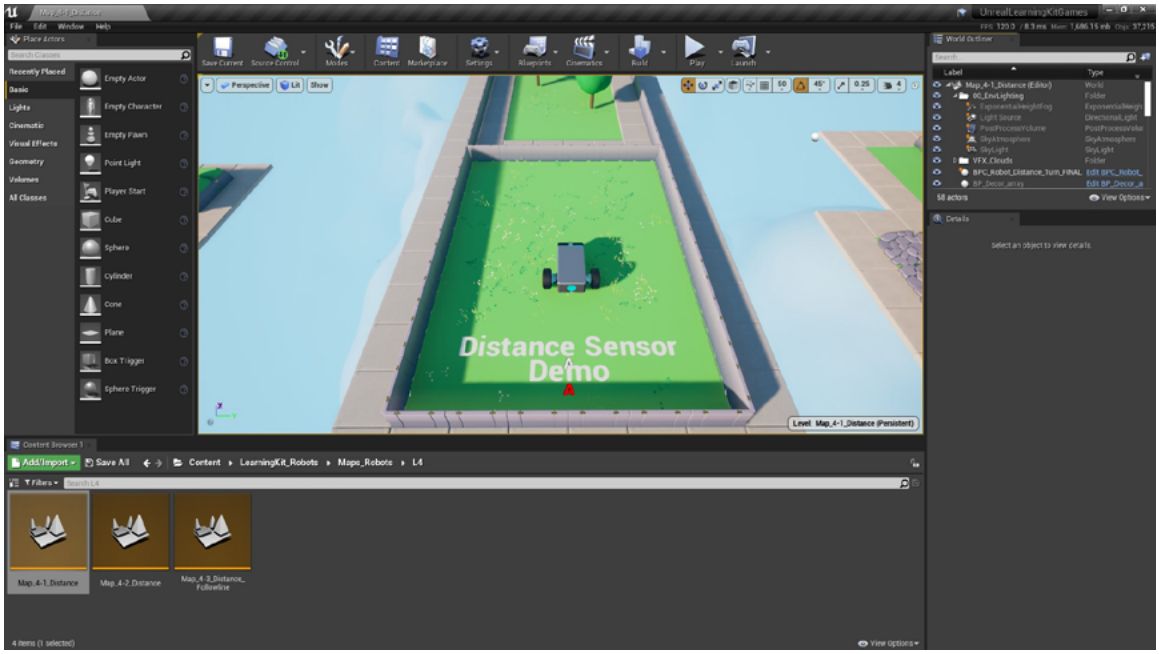


Figure 4 - 1

- **Click on Play** to run the example
- **Observe** what the robot is doing in this example
- **Consider:** What should our robot do in this situation?

Exercise 2: Opening Our Robot and Adding a Distance Sensor

This activity will start with a forward-moving robot that does not yet have a distance sensor. We will be adding a distance sensor and coding it to make sure it can avoid obstacles.

Activity: Open Our Beginning Robot

- In the **Content Browser**, navigate to the **Content -> LearningKit_Robots -> Maps_Robots** folder
- **Click** on the **L4** folder. Open **Map_4-2_Distance**

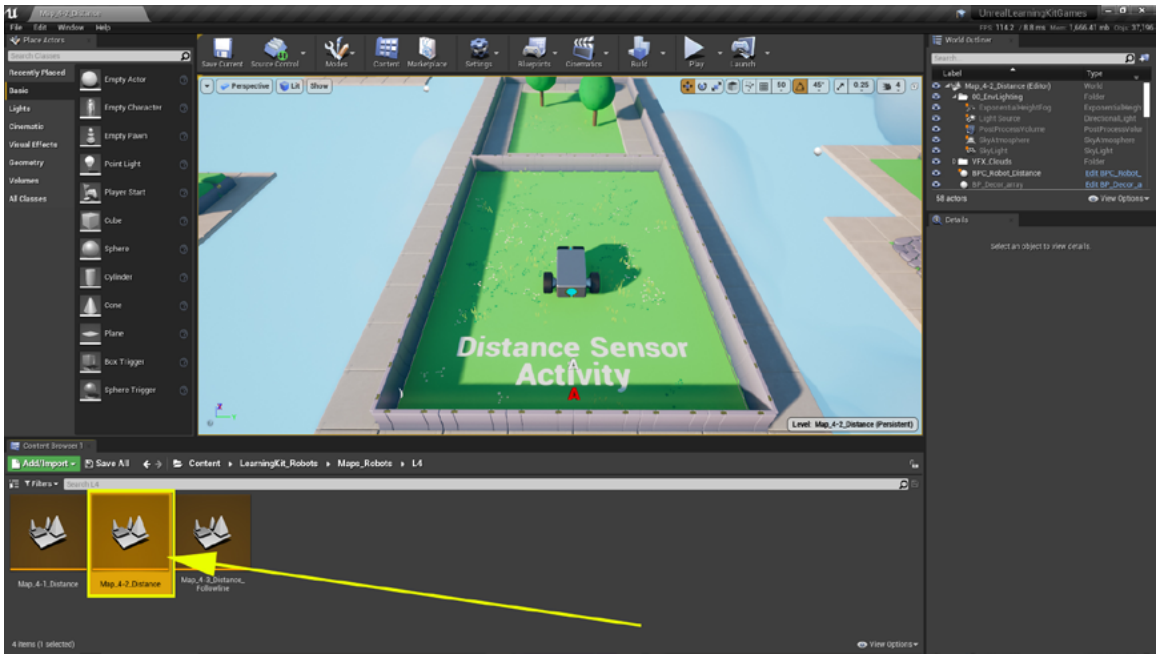


Figure 4 - 2

Sensor Position

Where does the robot need to look for obstacles? We are assuming that our robot is moving forward, thus it needs to look in front of itself. The Distance sensor is illustrated below on the robot. Look at the robot in your open project file, and consider the following details:

- **Where** should your distance sensor be located on the robot?
- **What direction** should the sensor be facing?
- **How far** does the sensor need to "see" in order to be effective?

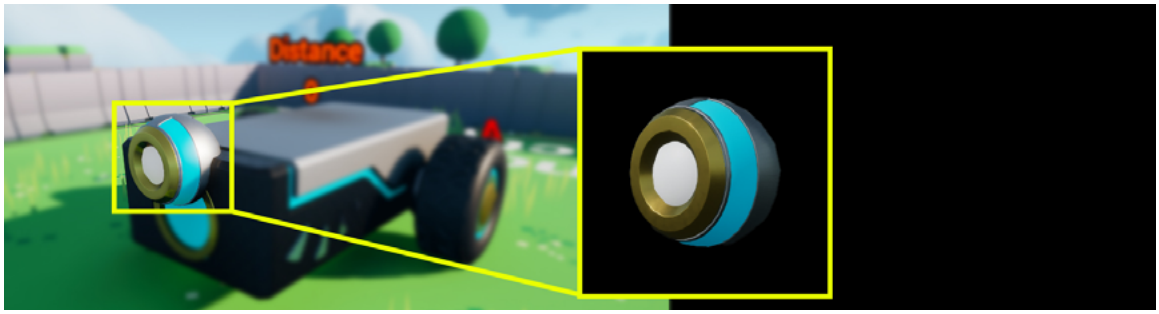


Figure 4 - 3

Add the Distance Sensor

- **Click the robot** in the **Viewport** to select it
- **Focus your view** of your robot to be able to view and place your sensor into the best position based on the prior robot construction and sensor

Focus Keys Tips:

- *The F key will focus your view up close on the robot*
- *Use the 0 key to view the entire robot from above*
- *Use the 1 key to view the working area*

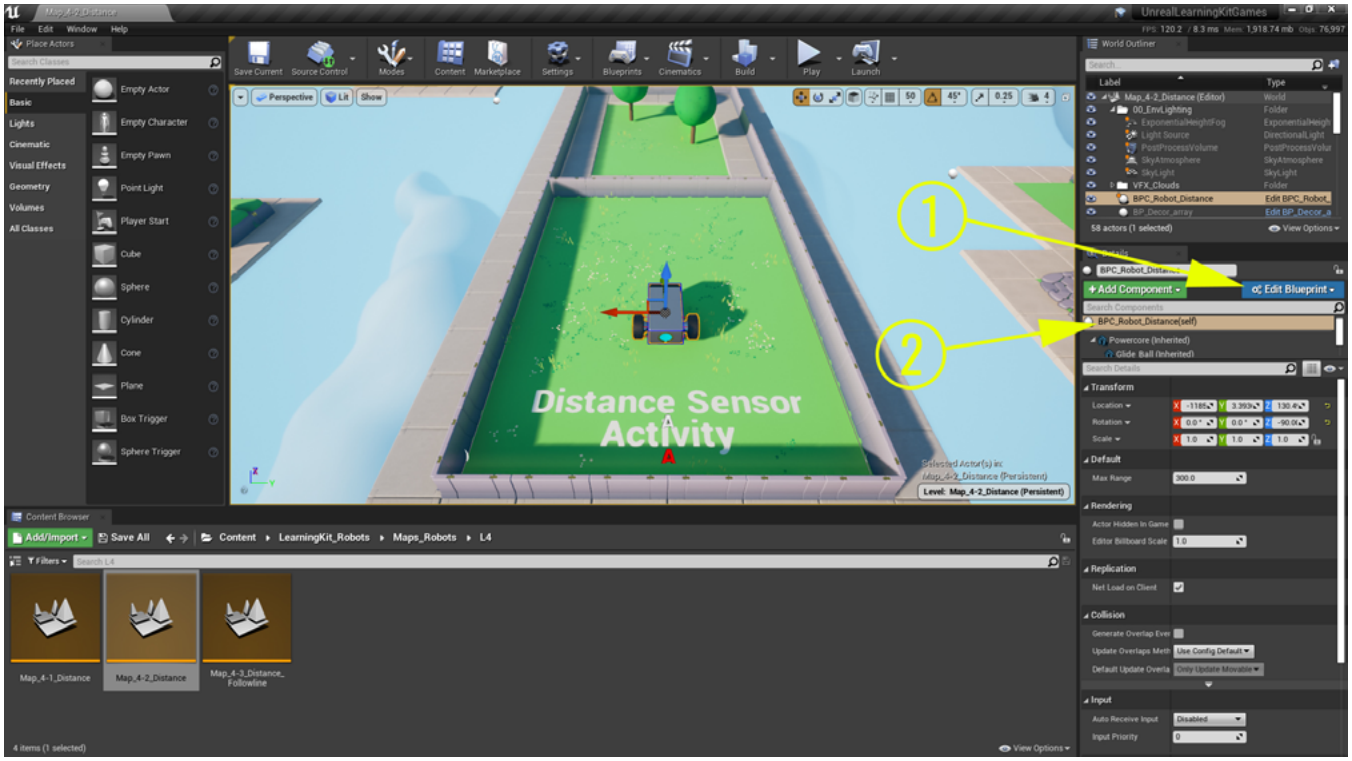


Figure 4 - 4

- In the **Details panel**, click **Edit**, then **Open Blueprint Editor**
- From the **Components panel**, select **Add Component** (add a new component to this actor)
- In the search box, type in “sensor” and select **BP Sensor Distance**

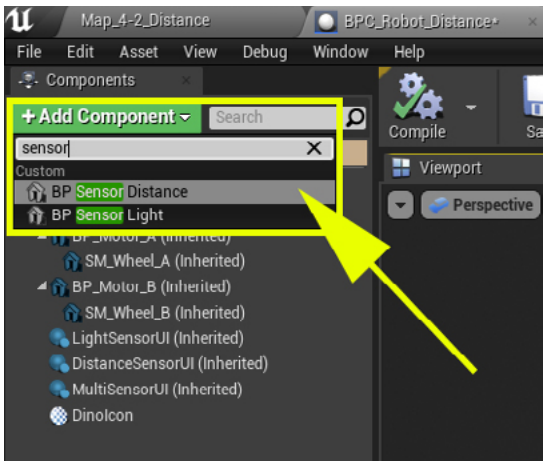


Figure 4 - 5

- **Click the Viewport** to view the robot design
- **Click and drag** the sensor into position on your robot
- **Rotate** the sensor (use the **Rotate Gizmo** on the viewport toolbar) so it faces the direction of the robot will be traveling. The white circle in the middle of the blue ring is the front of the sensor

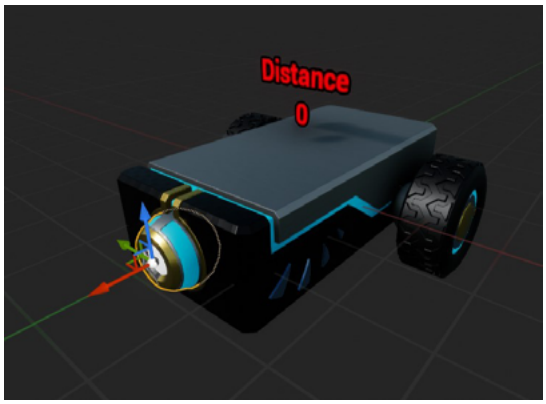


Figure 4 - 6

- Remember to **Compile** and **Save** after you have your sensor in the desired location.

Exercise 3 – Activate the Distance Sensor

Code the Distance Sensor to Run

- Click on the Event Graph to view your Blueprint code

The Move Robot Forward code has already been provided for your convenience. We taught you how to code the robot driving movement and comment in prior activities. In Lesson Four, we will have you focus on adding and coding the Distance sensor. This is what your code will look like now.

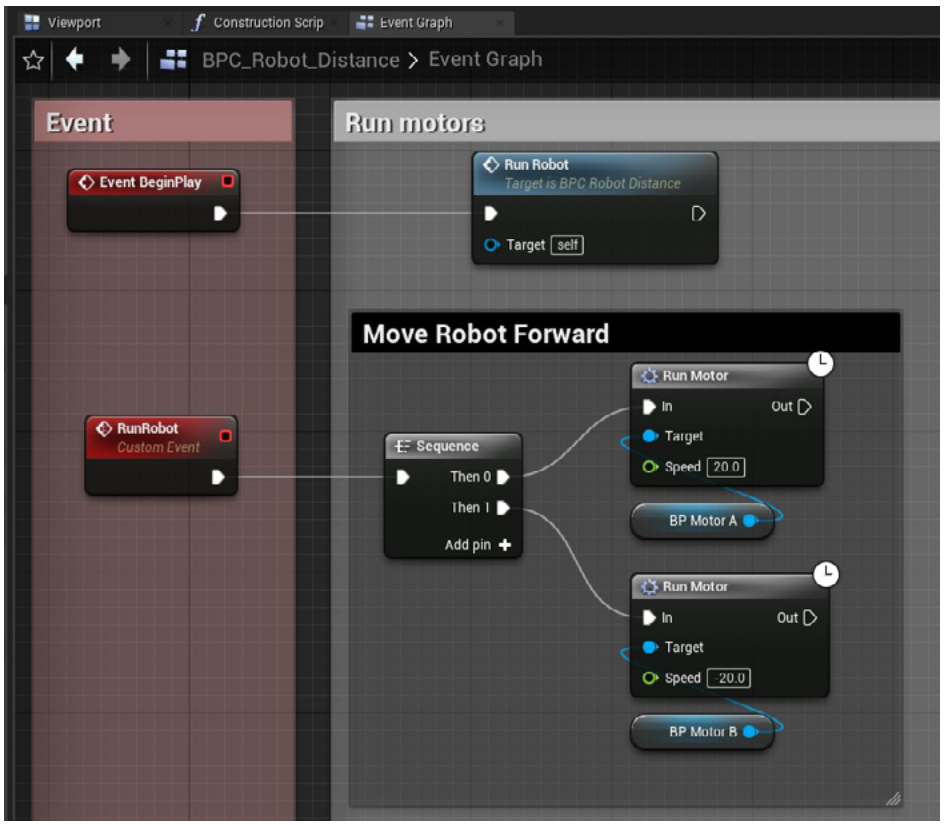


Figure 4 -7

We need to add the Distance Sensor node to the code:

- We need to make some room in our code. On the **Move Robot Forward** comment box, **click and drag** it to move to the **right**
- To disconnect the current wire, hold the **ALT** key and **left-click** on the wire between the red **Run Robot** event and the **Sequence** node
- We are making room to add the logic for the distance sensor

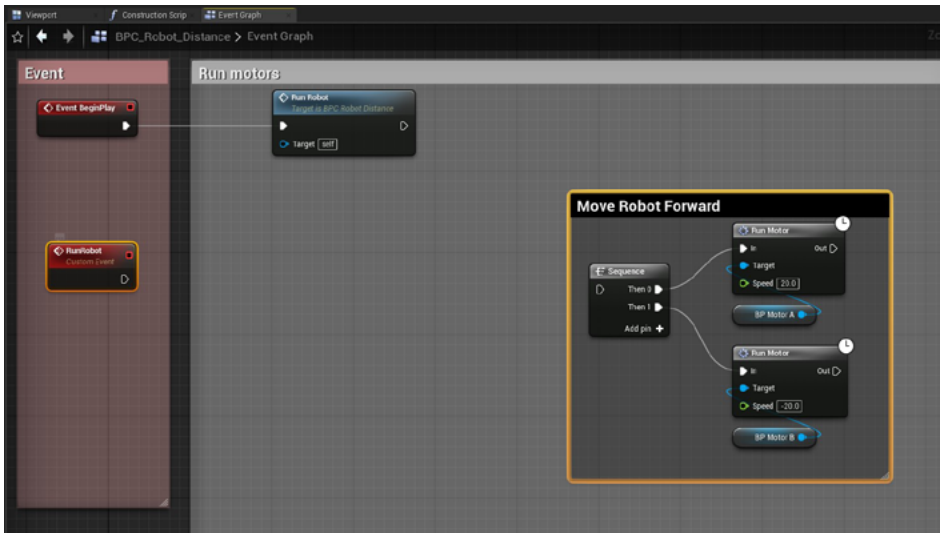


Figure 4 -8

- From the **Components panel, click and drag** the **BP_Sensor_Distance** node into the **Event Graph**
- **Drag** it into position as illustrated in Figure 4-9

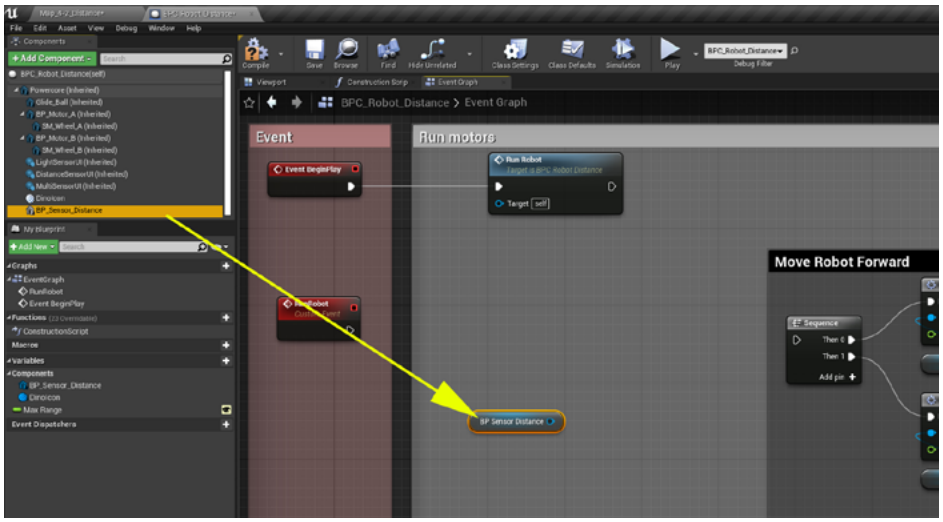


Figure 4 - 9

Now we will add the "**run distance sensor**" node.

- **Drag** a blue **wire** from the **BP_Sensor_Distance** node **output pin** to the **right**
- From the pop-up menu, **type** into the search box "**run distance sensor with threshold**"
- **Select it** or **press Enter** to place the node into your graph

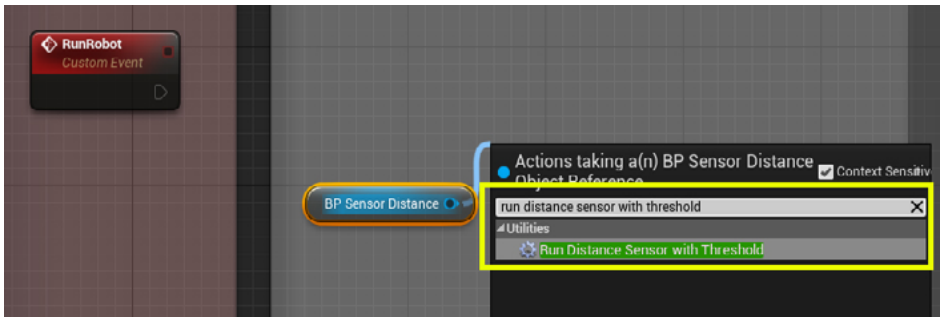


Figure 4 - 10

- The **BP_Sensor_Distance** should now be wired to the **Sensor** input on the **Run Distance Sensor with threshold** node. Your code will now look like Figure 4-11

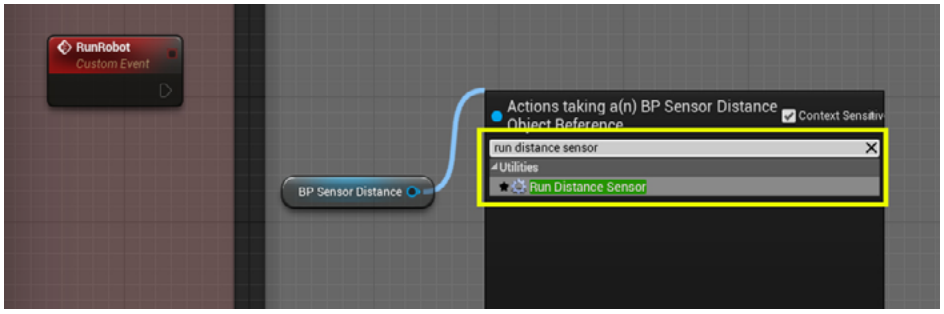


Figure 4 - 11

Now we need the **RunRobot** Event box to connect to the **Run Distance Sensor** node in the Run motors box:

- **Click and drag a (white) wire** from the **RunRobot** Event to the **Run Distance Sensor** nodes **In** pin
- The nodes should be connected as shown in Figure 4-12

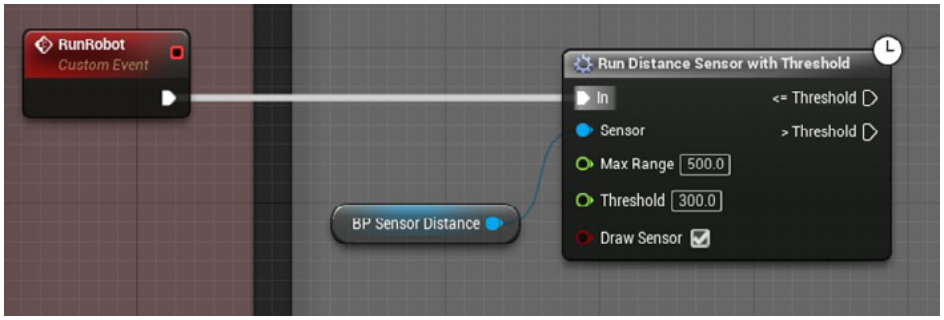


Figure 4 - 12

Let's save some time placing another Run Robot node by duplicating the above node.
In the Run motors box, you can either:

- Click on **Run Robot** node, then **press CTRL+C**
  OR
- **Right-click** on the **Run Robot node**, then in the shortcut menu **click** on **Copy**

To paste the node:

- **Click** in the **gray** graph area, then press **CTRL+V**

Place it to the right of the Run Distance Sensor node:

- **Click and drag** the **Run Motor** node to the right of the **Run Distance Sensor** node
- Your code should look like Figure 4-13

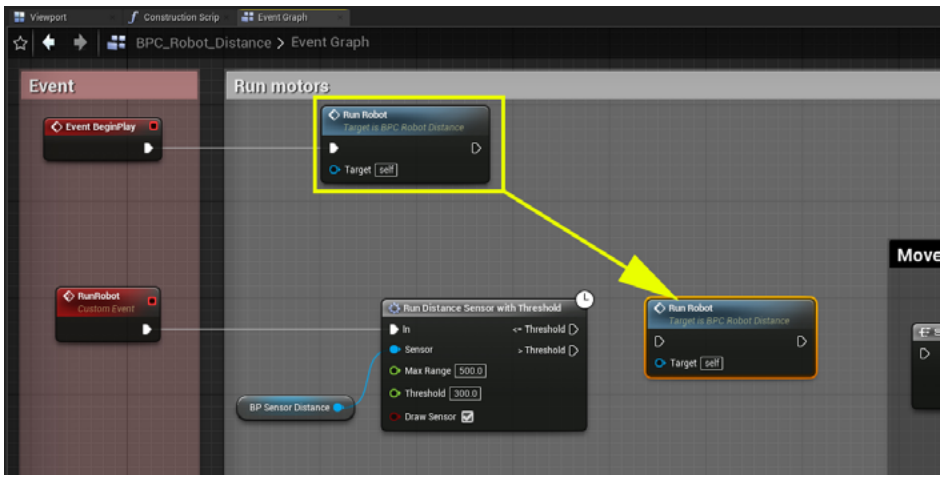


Figure 4 - 13

Now we need to connect our decision branches to the **Run Robot** node:

- **Click and drag** the **<=Threshold (True pin)** to the **Run Robot** node **input**
- **Click and drag** the **> Threshold (False pin)** to the **Run Robot** node **input**
- Your code should look like Figure 4-14

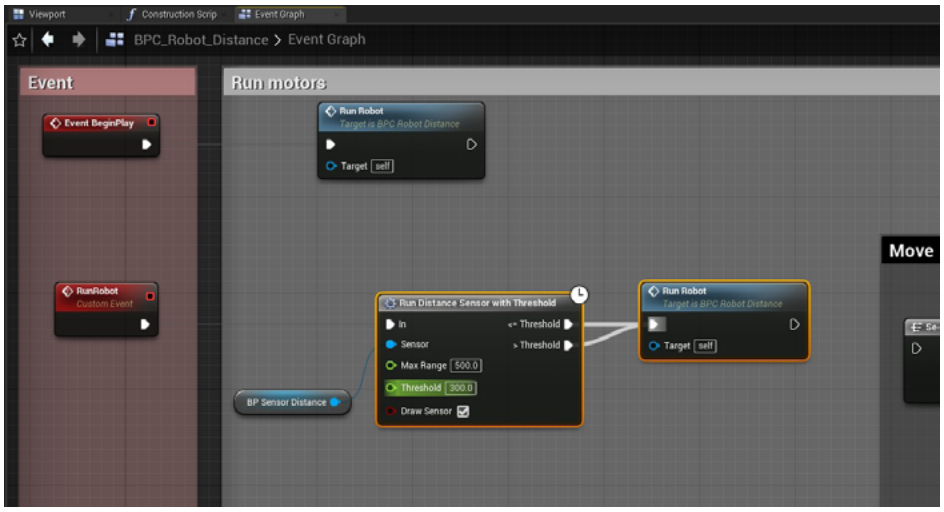


Figure 4 - 14

Let's test this code before we move on.

- **Compile** and **Save**
- **Pro tip: Drag** the Event graph to place it in a separate window, to be able to watch the robot gameplay while watching the flow of your code progress
- Click on the **Map_4-2_Distance** tab
- Use the Move gizmo to place the robot so that it is facing a wall, about one robot length away
- **Play** the game
- **Observe** the **distance readout** above the robot
- **Repeat** as needed and observe the differences in distance readout values

The information on the screen can be interpreted as numbered below in Figure 4-15.

1. The **line** is the max range. It changes to a green color after it detects collision
2. The **red dot** is the first collision observed by sensor

The **distance value** above the robot is the current distance measured from the sensor to the red dot along with the line

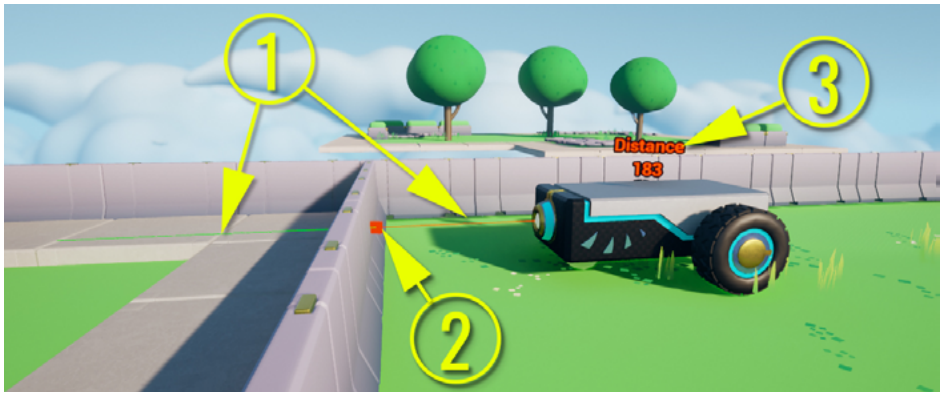


Figure 4 - 15

Action: Now we will run the code telling the robot to drive forward while looking for an obstacle (the wall) in front of it. This will be done in the **BPC_Robot_Distance** Event Graph

- First, disconnect the **Run Distance Sensor** node from the **Run Robot** node (click on each wire while holding the **ALT** key to break them)

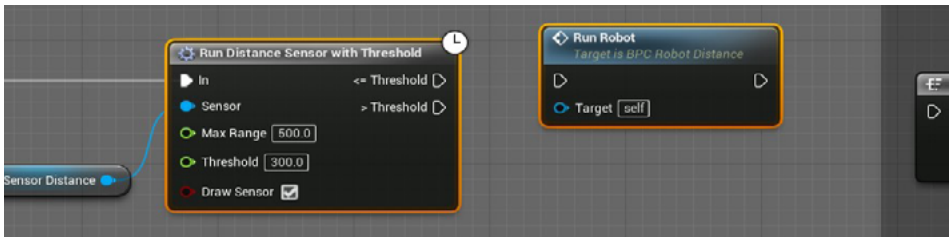


Figure 4 - 16

Next, switch the order of the Move Robot Forward comment box and the Run Robot node so they are arranged as shown below. (**Click and drag** each box **left** or **right** until rearranged to look like Figure 4-17)

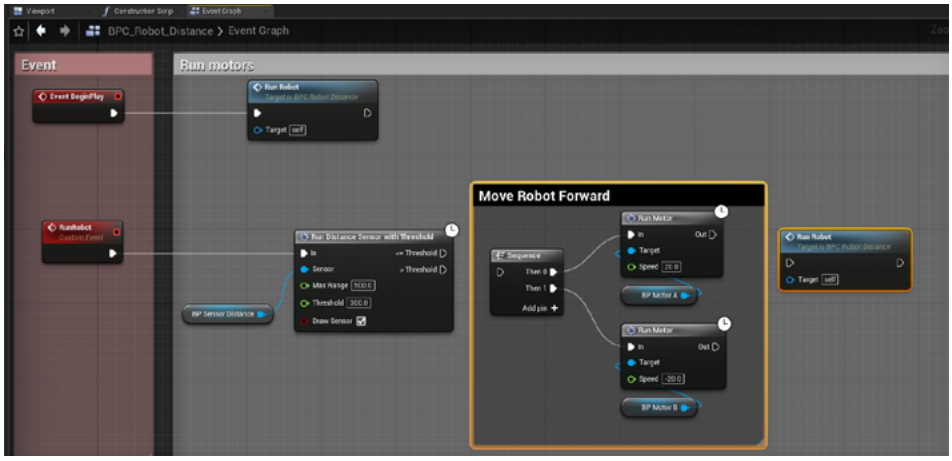


Figure 4 - 17

- **Drag a white wire** to connect the > **Threshold** pin to the **Sequence** node **input** inside the Move Robot Forward box (Figure 4-18)

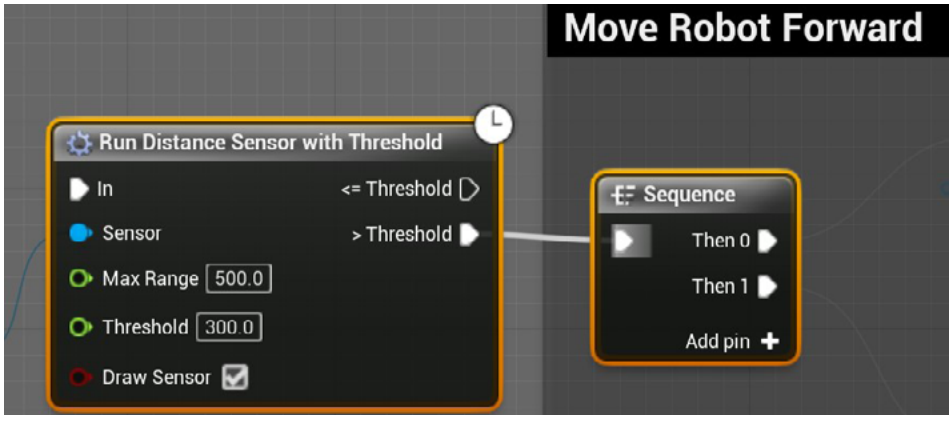


Figure 4 - 18

- **Drag white wires** to connect the **Run Motor Output pins** to the **Run Robot Node input pin**

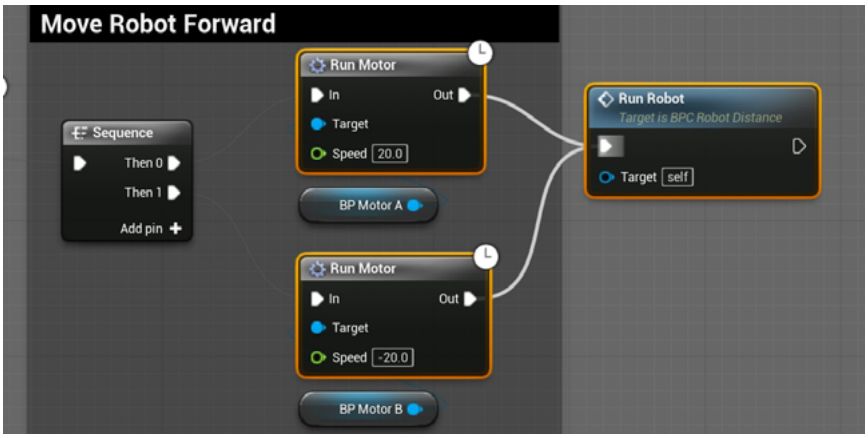


Figure 4 - 19

- Your code should now look like Figure 4-20

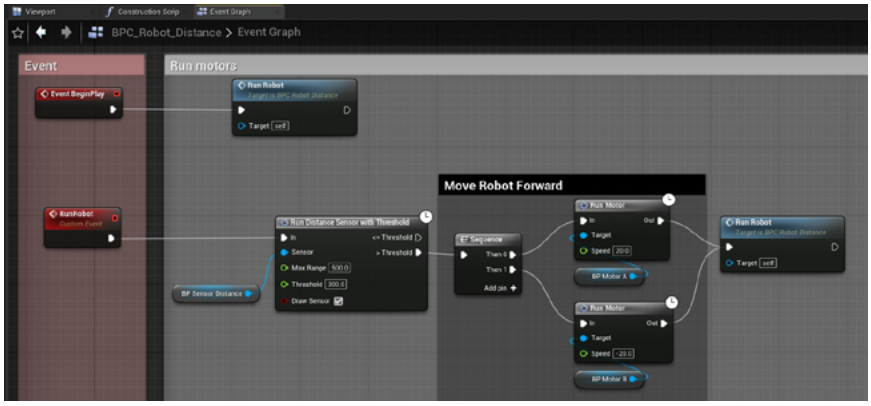


Figure 4 - 20

- **Compile** and **Save**
- **Click** the **Map_4-2_Distance** tab window
- **Play** the game
- **Observe**, reposition your robot and repeat the process to see how the robot performs when it starts too close to the wall and farther away from the wall
    - If the robot was too close to the wall, did it move when you started the program?
    - Did the robot's distance sensor see the wall as it approached it?
    - What numeric value did you see above the robot as it got close to the wall?
    - What should the robot do when it sees an obstacle?

What is the Sequence Node, and Why Are We Using It?

The **sequence** node will execute a series of instructions from its top output pin to its bottom output pin. Simply stated it executes in a sequence starting from "**Then 0**" through the rest of the "**Then #**" pins.

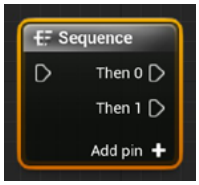


Figure 4 - 21

For this lesson we are using the **Sequence** node so the motors start turning the wheels at the same time. If we don't use this node and wired the code as seen in Figure 4-21, **Motor A** would begin to turn before **Motor B**. This would cause the robot to turn slightly to the **Motor B** side, at the moment the game starts.

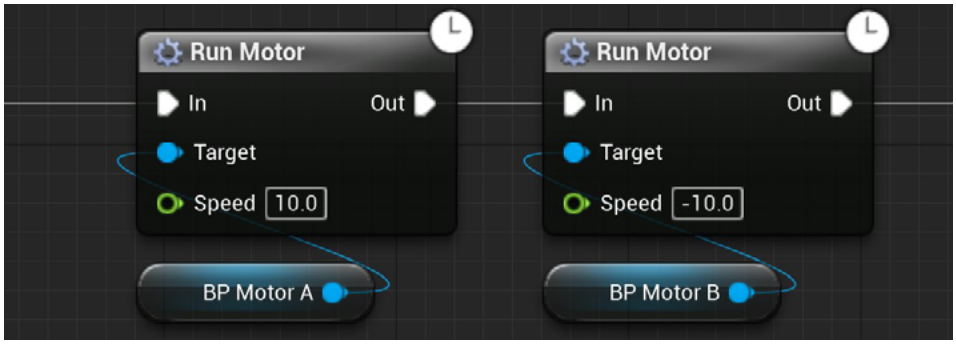


Figure 4 -22

Exercise 4 - Avoiding Running into Obstacles – STOP Driving if it Sees a Wall

Detecting a Possible Collision

As we get feedback from the sensor, we need to determine what distance should be considered "too close." If an object is anywhere within this close distance, the robot should stop.

When the Distance sensor sees an object, obstacle, or wall, it registers a specific numeric value. Let's learn more about the **Max Range** and **Threshold** settings within the **Run Distance Sensor** node.

To allow for slight variations, we want to establish a range of values that the sensor may see for each object or wall we are looking to avoid, thus determining the low threshold value and the high threshold value. When the value is within this set range, we can program it to react to commands for those objects. Our Run Distance Sensor node has these settings:

- **Max Range:** The maximum distance the sensor is **looking ahead of itself**. The default value is 500 centimeters
- **Threshold:** The distance at which the sensor will **detect collision** and will fire the **Threshold Reached** pin. The default value is 300 centimeters
  When the sensor detects an object, it will mark it with a red square along the laser (see Figure 4-15)
- **Draw Sensor:** Toggle ON/OFF for the **visibility** of the red and green sensor laser (see Figure 4-15)

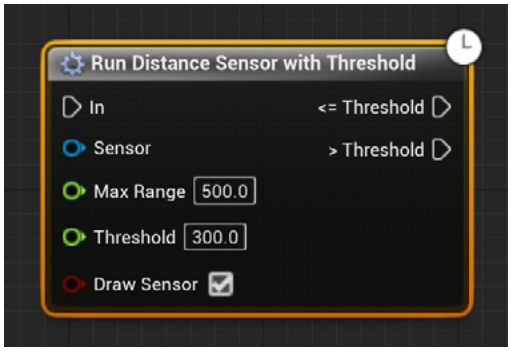


Figure 4 -23

The logic is telling the sensor to look ahead of itself up to 500 centimeters and if it sees an object in its path at 300 centimeters then fire the **Threshold Reached** pin. If the sensor does not see an object, then fire the **Threshold NOT Reached** pin.

Considerations: Did the current settings see the obstacles in time? Should we ask the sensor to be looking further ahead? How close to the wall do we want to get before stopping?

Logic decisions:

1. **While the sensor "sees" NO objects in its path— Threshold NOT Reached:**
    - Which direction should the robot move?
    - How long should the robot move?
    - What speed should the robot be moving?
    - How long should the sensor test for the condition?

2. **Action after sensor sees an object/obstacle (walls, barrels, etc.)— Threshold Reached:**
    - Should the robot stop?
    - How does the robot return to its forward driving?

3. **Sequence / Loops:**
    - Loop beginning: Where does each loop begin?
    - Loop Ending: Where does each loop end?
    - Duration: How long should the robot continue repeating the loops?

Now let's put that knowledge to good use and make our robot stop when it senses an object.
First, let's make some room for the new code.

- In the **BPC_Robot_Distance Event Graph**, select nodes **Move Robot Forward** comment box and the **Run Robot** node by clicking on each while holding **SHIFT**
- **Position your cursor** on a 4-headed movement arrow on either box
- **Click and drag** to move the **Move Robot Forward** comment box and the **Run Robot** node to the right

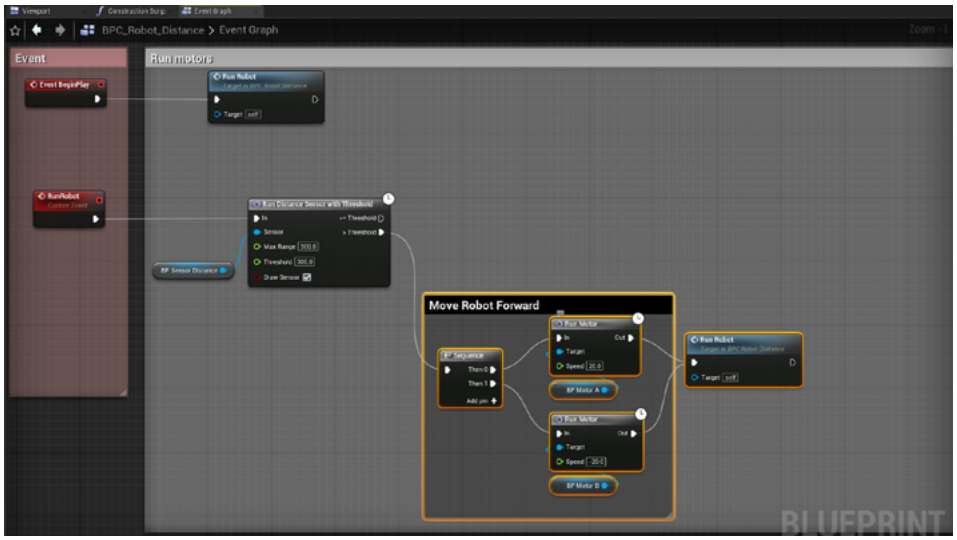


Figure 4 - 24

- **Select** and **Copy** the **BP_Motor_A** and **BP_Motor_B** reference nodes (you can click on one and shift-click on the other node to multi-select items at the same time)
- Use **CTRL+C** to **Copy**, or **right-click** one of the selected nodes and choose **Copy**
- Click in the gray Blueprint area
- **Right-click** and **Paste** the **BP_Motor_A** and **BP_Motor_B** reference nodes above the **Move Robot Forward** comment box. (see Figure 4-25)

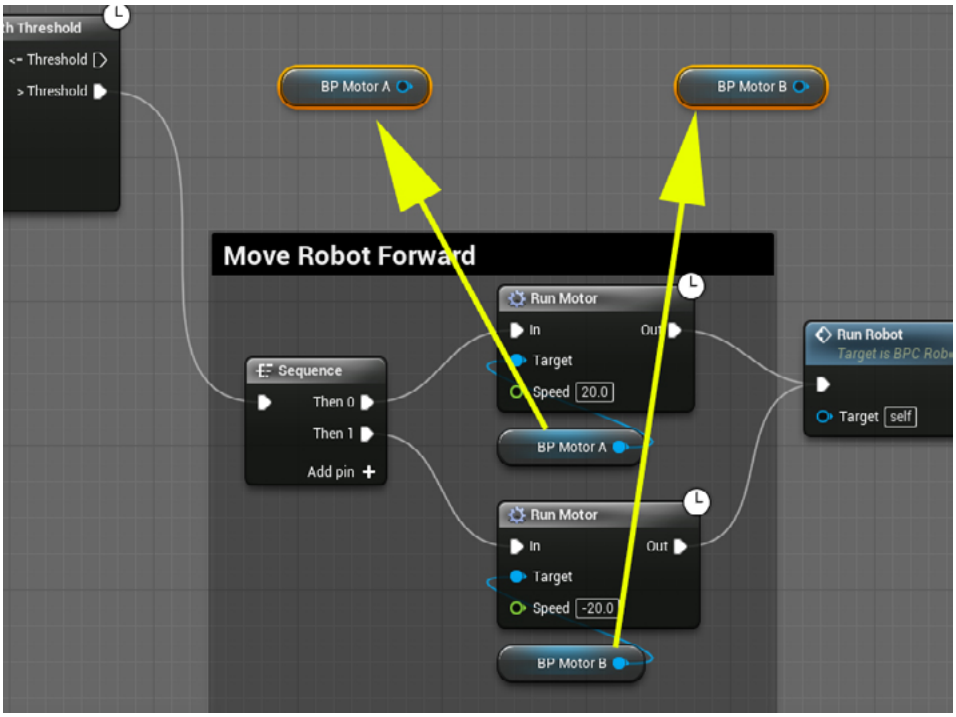


Figure 4-25

- Click and drag from the **BP_Motor_A** pin and search for “stop motor”
- Select the **Stop Motor** node

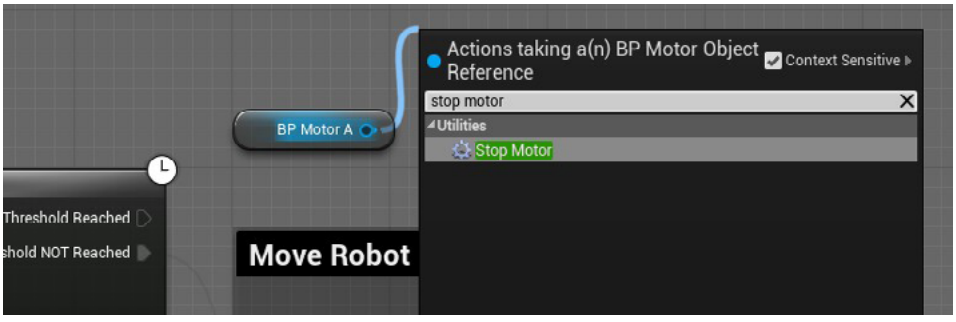


Figure 4-26

- Repeat the above step for **BP_Motor_B** node
- Your nodes should look like Figure 4-27

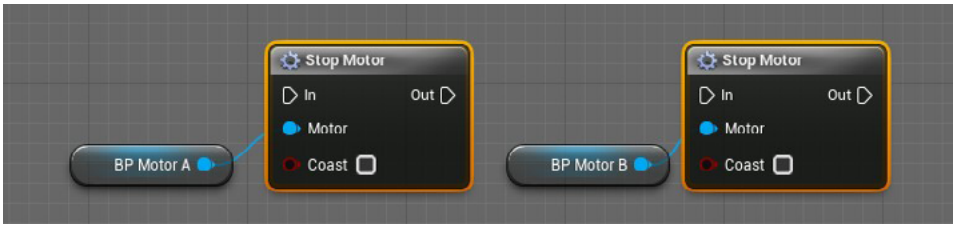


Figure 4-27

To complete this code, we will need to tell both motors to stop when the threshold is reached.

- Connect (wire) the **Threshold <=** pin to the first Stop Motor node input pin
- Then **connect** (wire) the **Output** pin of the first **Stop Motor** node to the **Input** of the second **Stop Motor** node
- The code should look like Figure 4-28

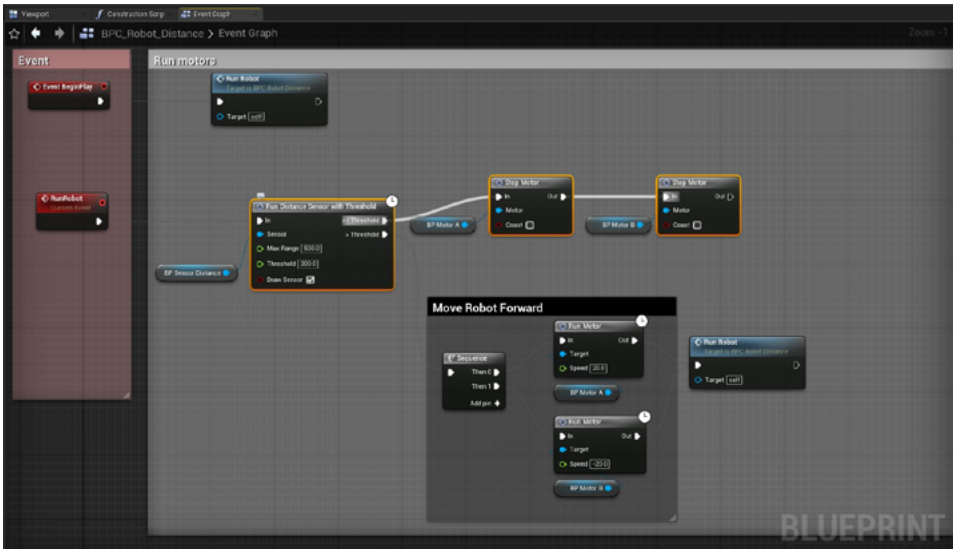


Figure 4-28

- **Compile** and **Save** your code
- Click on the Map_4-2_Distance tab window
- **Play** the game
- **Observe** and **record** in your notes what the robot is doing

Congratulations! Your robot should now be driving forward and stopping before running into any of our walls. You are ready to advance your capabilities and upgrade your line-following robot from Lesson Three into a line-following, collision-avoiding robot!

Challenge - Combine the Driving and Collision Avoidance

We should now have the functionality to follow a line AND avoid a collision. These functions must be combined into a single robot that will follow a line and briefly stop if an object is blocking the way. It will be easier to implement if you understand the solution prior to coding. (This is true for most coding.)

Objective

Add a distance sensor to a line-following robot so it successfully follows the line, stopping for any obstacles that get in the way. When the obstacle moves out of the path, the robot continues moving forward along the path.

Starting the Challenge

- Open **Map_4-3_Distance_Followline**

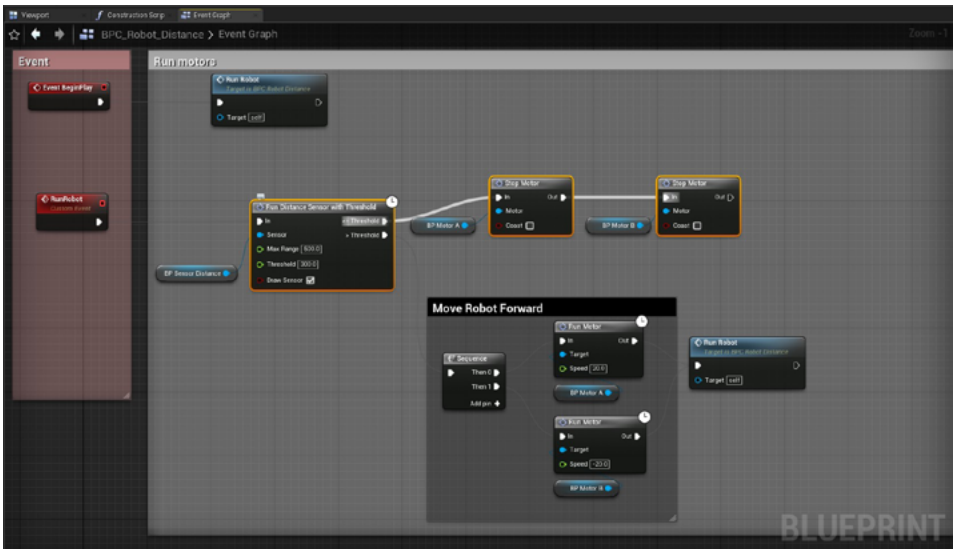


Figure 4-29

- Select the robot and edit the **BPC_Robot_DistanceFollowLine** Blueprint, and you will see that the **Event Graph** contains the programming to follow the line. Modify this code to stop the robot when an object is too close

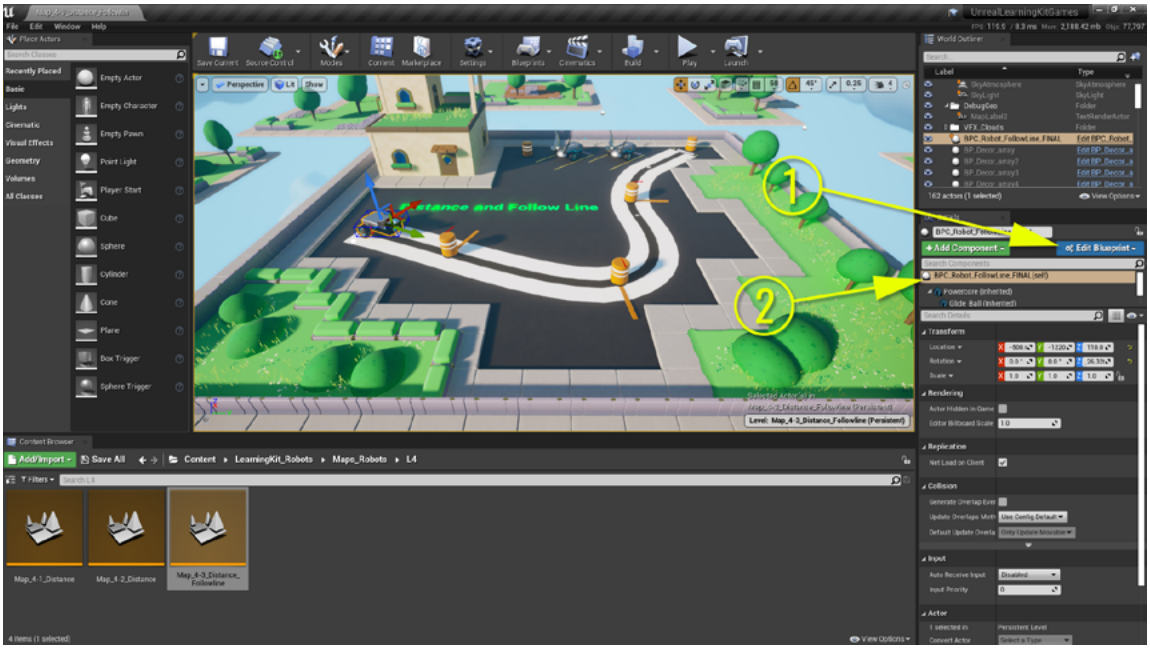


Figure 4-30

- Return to the earlier steps, if necessary, to place a **BP_Sensor_Distance** on this robot
- Place the distance sensor in a good position to detect an object in front of it
- Stop the robot if an object is too close and resume driving when no object is present. Refer to the steps above to refresh your memory of the steps to complete
- Don't forget to **Compile** and **Save** often as you test each iteration
- Keep a notebook nearby so you can **make notes** according to what configurations you test
- **Play** and **Test** to be sure your robot can follow the path line

Testing and Debugging

Continue testing and playing the robot and editing the code until the desired action is achieved and is consistent. The end goal is to have the robot reach the parking space without running into any barrels along its path.

Activity Review

- Learn to use a distance sensor.
- Use feedback from the distance sensor to avoid collisions
- Determine how to use sensor feedback to decide when to stop the motor
- Combine line following and collision avoidance into a single robot
- Solve more complex logic by making a plan
- Demonstrate evidence of challenges, success, and understandings

EXTENSION ACTIVITIES

If you have completed this activity successfully and would like to attempt additional challenges, consider the following:

- Try making the robot move faster. Adding speed to a vehicle introduces many interesting new challenges. What new problems do you discover by simply increasing the robot's speed?
- Change the path! To edit the base level you will need to open the level with the name ending in "_Env." Click on the path and click on the white dots along the path to activate the Move gizmo. (This path is known as a spline. It can be moved into many interesting shapes.)

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.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, but 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 (Object Avoidance Robot)	Demonstrates object avoidance robot in action with clean, efficient code. Shows documentation of attempts, trials, and successful robot runs.	Uses conditional statements, loops, and distance sensor threshold settings correctly to avoid objects in the robot's path.	Can explain how a LiDAR sensor works, and explains the conditional statements with threshold settings necessary. May not have succeeded in a complete robot run without collisions.	Did not code an object avoidance robot. Does not express knowledge of how to code a robot to move with a distance sensor.

STUDENT GUIDE

LET'S TRAIN VIRTUAL ROBOTS

LESSON 4: COLLISION AVOIDANCE ROBOT

UNREAL ENGINE