6 program flow

Program flow is about controlling the order in which your programming blocks are run. Typically, the blocks run from left to right, but you can control the flow of the program by making blocks wait, repeat, or choose an action based on some condition. The three main blocks used for program flow are the Wait, Switch, and Loop blocks.

You've already seen how the Wait block works. In this chapter, I'll cover the Switch and Loop blocks in depth. The Loop Interrupt block is also used to control Loop blocks, so I'll explain that as well.

NOTE The Switch and Loop blocks also have some features that are used exclusively with data wires, which will be covered in Chapters 9 and 10.

the switch block

The *Switch block* (shown in Figure 6-1) lets your program make a decision about which blocks to run. This type of structure is called a *conditional* because the program flow changes based on whether a condition is met. The Switch block checks the condition in order to choose from two or more groups of blocks; these are called *cases*. This gives your program the ability to make a decision and react to data from the robot's sensors. For example, the *RedOrBlue* program on page 74 uses the reading from the Color Sensor to decide which Sound block to run.

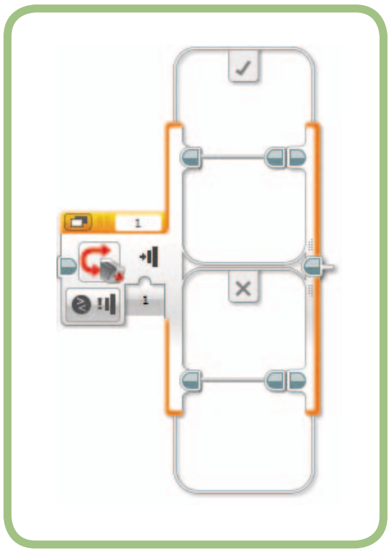


Figure 6-1: The Switch block

setting the condition

To set the condition on a Switch block, first select a sensor and mode from the Mode Selector list. Then enter the Threshold value and any additional parameters, such as the sensor port.

Think of setting the mode as asking a question. The block shown in Figure 6-1 asks the question, "Is the Touch Sensor pressed?" This is a yes-or-no question, so the Switch block can have only two cases: the true case and the false case.

Other questions allow for more than two answers. For example, the question "What color does the Color Sensor detect?" has eight possible answers because the Color Sensor can detect eight values (seven colors and the No Color value). For this question, the Switch block can have up to eight cases.

RESIZING A BLOCK

The Switch and Loop blocks automatically resize when you drag blocks into them. You may want to adjust the size of a Switch or Loop block yourself to shrink a block after removing some of the blocks inside, to see more tabs of a Switch block using Tabbed View, or to make room for comments.

When you click a Loop block or a Switch block case, a set of resizing handles appears, as shown in Figure 6-2. Click and drag a handle to change the size of the block.

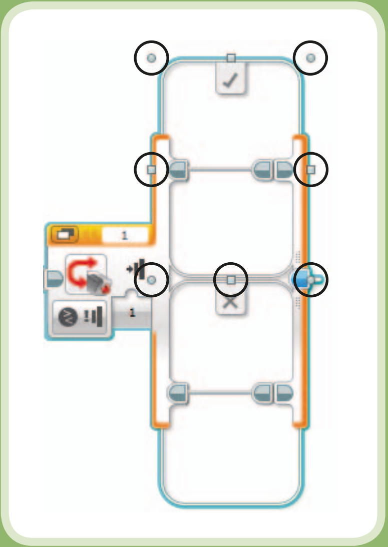


Figure 6-2: Resizing handles

the LineFollower program

The *LineFollower* program is a simple line follower, where the robot uses a Switch block to decide the direction to go in. The TriBot uses the Color Sensor in Reflected Light Intensity mode to follow the edge of a line by adjusting the steering based on the Light Sensor reading.

For this program, you need to remove the Touch Sensor bumper from the front of the TriBot and replace it with the Color Sensor. The Color Sensor should be mounted so that it points downward, as shown in Figure 6-3.

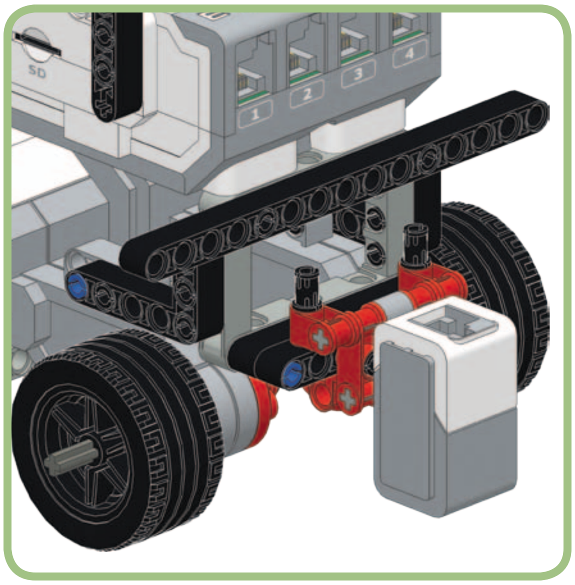


Figure 6-3: Color Sensor position for following a line

To test this program, you need a dark line to follow. You can use black electrical tape or a black marker to mark an oval on a white poster board (see Figure 6-4). Your line should be at least an inch (3 cm) wide and needs to be much darker than the background.

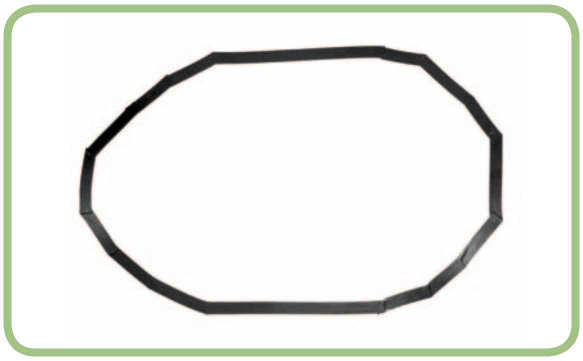


Figure 6-4: A test line made from electrical tape and poster board

NOTE This program works best with smooth turns; sharp corners may cause problems. The final version of the program, presented in Chapter 19, is better at navigating corners.

the basic program

The program makes the robot follow the line by constantly adjusting the direction in which the robot steers in order to keep the Color Sensor over the edge of the line (see Figure 6-5). With the Color Sensor in Reflected Light Intensity mode, the sensor reading tells you how much light is reflected from a small circular area under the sensor. When the sensor is over the white background, it will give a high reading because a large quantity of light is reflected. When the sensor is completely over the dark line, it will give a low reading because a small amount of light is reflected. Between these two extremes, the reading varies based on how much of the line is under the sensor. As the robot moves forward, it will use the Color Sensor reading to tell where it is relative to the edge of the line. If the robot is too far over the line, it will steer to the left, moving it back towards the edge. If the robot moves away from the line, it will steer to the right to get back to the edge.

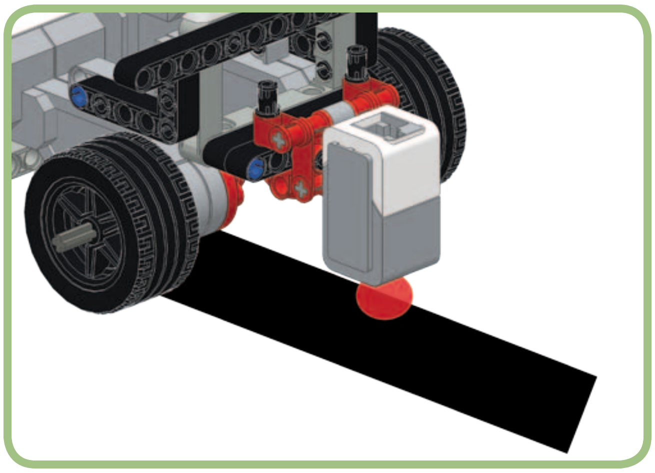


Figure 6-5: The Color Sensor over the edge of the line

The program uses a Switch block to choose between two Move Steering blocks, one of which steers to the left and the other to the right. Figure 6-6 shows the completed program. The entire program is within a Loop block, which keeps the program running until you stop it. The Switch block reads the Color Sensor and decides which Move Steering block to run. The Move Steering block on the upper section (the true case) steers the TriBot to the left, and the one on the lower section (the false case) steers it to the right.

Now let's create the program:

1. Create a new project named *Chapter6*.
2. Create a new program named *LineFollower*.
3. Add the Loop block.
4. Drag a Switch block into the Loop block (the Loop block will grow to fit the Switch block).
5. Add a Move Steering block to each case of the Switch block.
6. Select the Switch block, and set the mode to **Color Sensor – Compare – Reflected Light Intensity**.

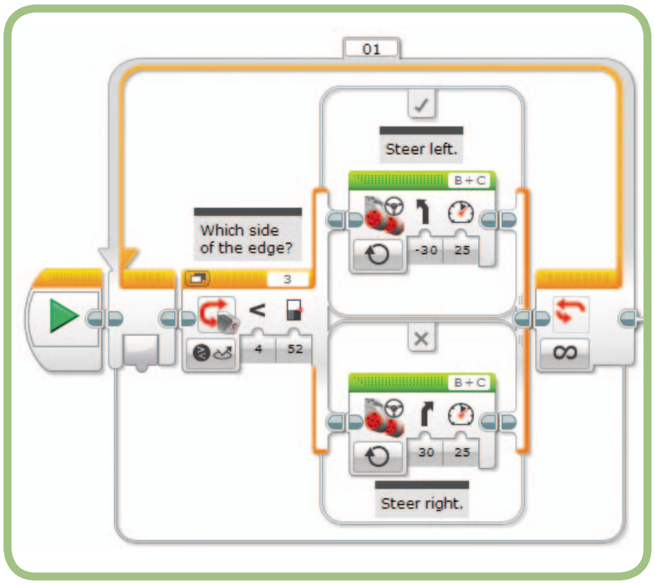


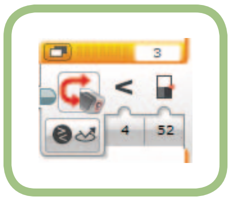
Figure 6-6: The LineFollower *program*

Now you just need to configure the settings for each block. Let's discuss how to decide on a value for each setting.

selecting the color sensor threshold

How should we determine the Threshold value to use for the Switch block? To make the TriBot follow the edge of the line, you'll need to find the value that the Color Sensor reads when it's on the edge of the line. Figure 6-7 shows the Color Sensor readings I get when I place the TriBot at five positions, moving from off the line to over the line. A simple but reliable approach to get a reasonable Threshold value is to take the average of the high (completely off the line) and low (completely over the line) values. Using the values from Figure 6-6 results in a Threshold value of (92 + 13) / 2, which I'll round down to 52 and use that as the Switch block's Threshold value. Note that this is close to the value I got when I placed the TriBot over edge of the line. Your value may be a little different depending on the sensor, the test pad, and the lighting in the room.

7. Set the Switch block's Threshold parameter. The block should now look like this:



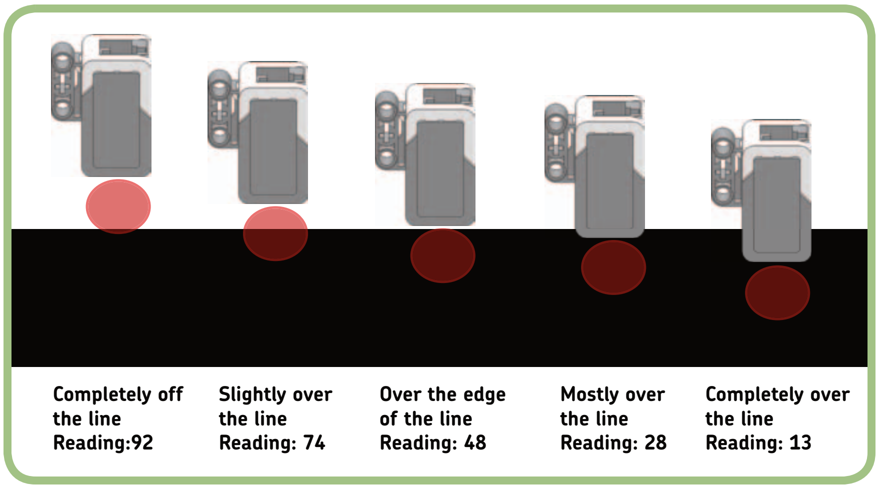
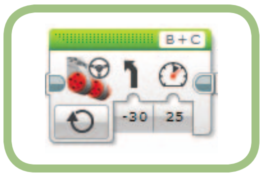

Completely off the line Reading:92
Slightly over the line Reading: 74
Over the edge of the line Reading: 48
Mostly over the line Reading: 28
Completely over the line Reading: 13

Figure 6-7: Color Sensor readings at different positions

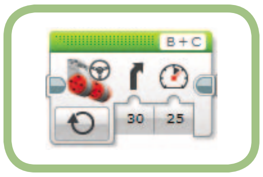
configuring the move blocks

The two Move Steering blocks have similar settings, except that they steer in opposite directions. The speed of the motors and the steering setting significantly affect how well the TriBot follows the line. If the Steering value is too low, the TriBot won't turn quickly enough to follow a curve in the line. If the Steering value is too high, the TriBot will have a lot of side-to-side motion as it constantly goes from one extreme to the other. Moving too fast makes it harder for the robot to react to changes in the line's direction. Start with a Steering parameter of positive and negative 30 (30 and -30) and a Power parameter of 25:

8. Select the Move Steering block on the upper Sequence Beam.
9. Set the mode to **On**.
10. Set the Power parameter to **25** and the Steering parameter to **-30**. The block should look like this:



11. Select the Move Steering block on the lower Sequence Beam, and set the mode to **On**.
12. Set the Power parameter to **25** and the Steering parameter to **30**. The block should look like this:



testing the program

Now download and run the program to see how well it works. You might need to adjust how fast the TriBot moves and how sharply it turns. If you do, make sure that you make the same changes to both Move Steering blocks. When it's working, see how much you can increase the speed and still have the robot follow the line reliably.

more than two choices

The first version of the *LineFollower* program makes the TriBot wiggle left and right while following a straight line, because the TriBot is constantly adjusting the steering. You can make the motion smoother with three Move blocks: one to steer left, one to go straight, and one to steer right.

In Color Sensor - Compare - Reflected Light Intensity mode, the Switch block can only choose between two sets of blocks based on the reading from the Color Sensor. To choose between three options, you'll need to use two Switch blocks. The first Switch block decides if the robot should turn to the left, and the second decides whether to go straight or turn to the right. This structure is used often in EV3 programming (and programming in general) to make sophisticated decisions.

Start by making the following changes to the *LineFollower* program:

13. Place a Switch block in the lower section of the existing Switch block, to the right of the Move Steering block.
14. Set the new Switch block's mode to **Color Sensor - Compare - Reflected Light Intensity**.
15. Drag the existing Move Steering block (the one that turns to the right) onto the lower section of the new Switch block.

16. Place a new Move Steering block on the upper section of the new Switch block.
17. Set the mode to **On** and the Power parameter to **25.**

The Switch block should now look like Figure 6-8.

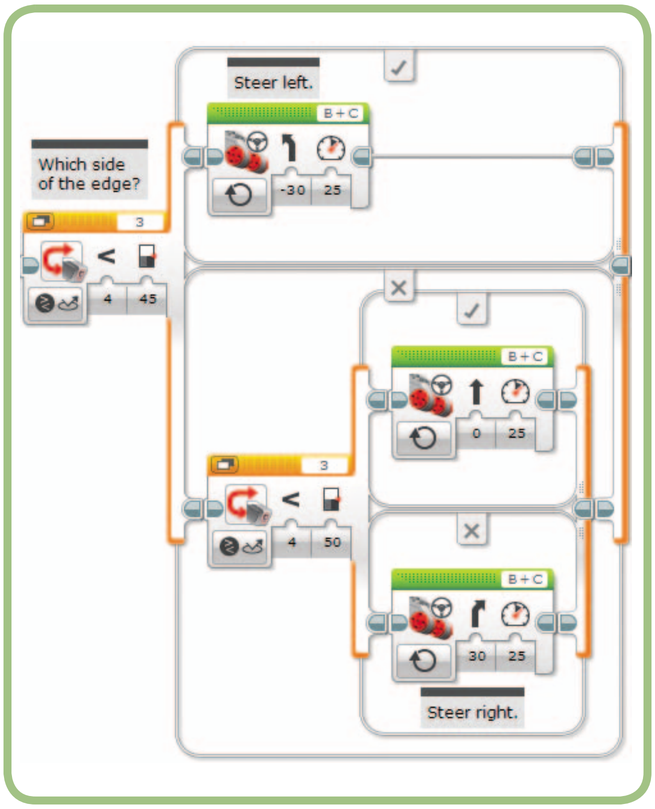


Figure 6-8: Setting the Threshold values for the new blocks

You need to set the Threshold values for the two Switch blocks so that the TriBot will go straight when it's on the edge of the line and turn when it moves away from the edge (by going either too far over the line, or too far away from the line).

Originally, I had a single Threshold value set at 52, which is in the middle of the two extremes, 13 and 92. To get two Threshold values, we can take the values halfway between this middle value and the two extremes. This gives me 32 and 72. Looking back at Figure 6-6, these values make sense because they are close to the readings I got with the sensor mostly on the line and mostly off the line. I'll set the trigger values so that the robot drives straight when the reading is between 32 and 72, and turn left or right when the reading is outside that range. Table 6-1 shows how the program should behave based on the Color Sensor reading. Be sure to calculate your own values, as they will vary depending on the lighting and the materials you used to mark the line.

table 6-1: color sensor ranges and program behavior

Color Sensor Reading	Program Behavior
0–31	Turn left
32–72	Go straight
73–100	Turn right

Follow these steps to complete the program:

18. Select the outer Switch block, and set the Threshold value to the lower limit of the range that should make the robot drive straight (which is 32, using my values).
19. Select the inner Switch block, and set the Threshold value to the lower limit of the range that should make the robot turn right (which is 73, using my values).

The program should look like Figure 6-9.

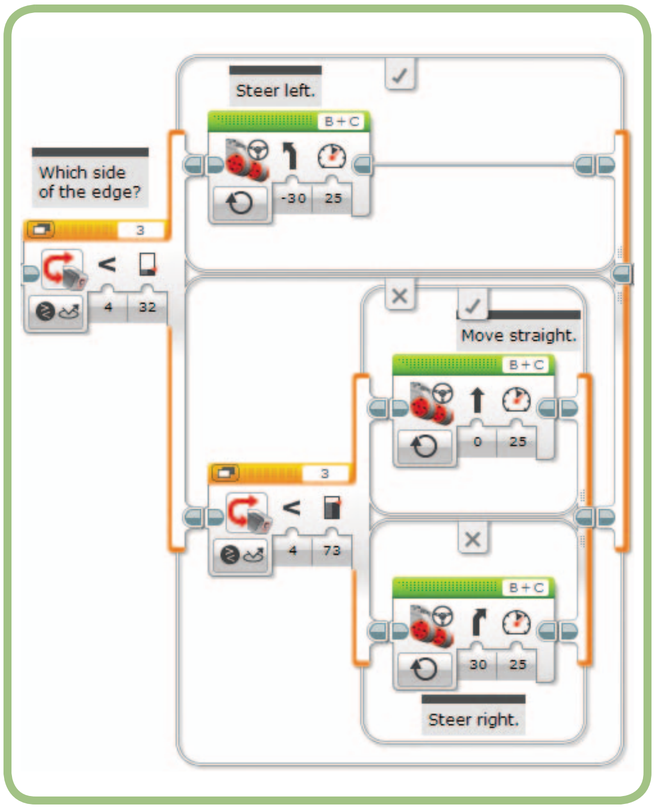


Figure 6-9: Setting the Threshold values

NOTE **In this program, each Switch block reads the Color Sensor value to make its decision. This means that the sensor is read twice and the two values may not be exactly the same. But this shouldn't present a problem because the sensor value won't change much in the time it takes the EV3 to run the two Switch blocks.**

testing the program

Now when you run the program, you should notice that the robot's motion is much smoother as it follows a straight line. Experiment with different Threshold values and Power and Steering parameters to see how fast you can make the TriBot move without veering off the line and getting lost.

using tabbed view

The *LineFollower* program uses *nested* Switch blocks, where one Switch block is placed within the other. Nested Switch blocks tend to take up a lot of room on the screen, which can make it difficult to work with the rest of the program. To shrink the size of the Switch blocks, switch to Tabbed View (click the **Flat/Tabbed Selector** button). This way, the program takes up much less space on the screen, as you can see in Figure 6-10.

NOTE **When using a Switch block in Tabbed View, I recommend putting all comments above the outermost Switch block. Because you only see one tab at a time, comments inside the Switch block might get hidden. Figure 6-10 shows a comment for the code from our line-following program.**

CHALLENGE 6-1

The first version of the *LineFollower* program chooses one of two actions each time it loops: turn left or turn right. The second version made the robot drive more smoothly by adding a third option: going straight. Improve the program again by adding two more cases so that the five options are: turn sharply left, turn gently left, go straight, turn gently right, and turn sharply right. You can do this by replacing the Move Steering blocks that turn left and right with Switch blocks that decide to turn sharply or gently based on the reading from the Color Sensor.

the RedOrBlue program

In this section, we'll create the *RedOrBlue* program, which will identify red and blue objects. We'll use the *IsItBlue* program from Chapter 5 as a starting point (Figure 6-11) because it already identifies blue objects. We'll begin by changing the program to recognize red objects, and then we'll update it to do something reasonable if the object is neither red nor blue.

The *IsItBlue* program is saved in the *Chapter5* project, so the first step is to copy the program to the *Chapter6* project and rename it.

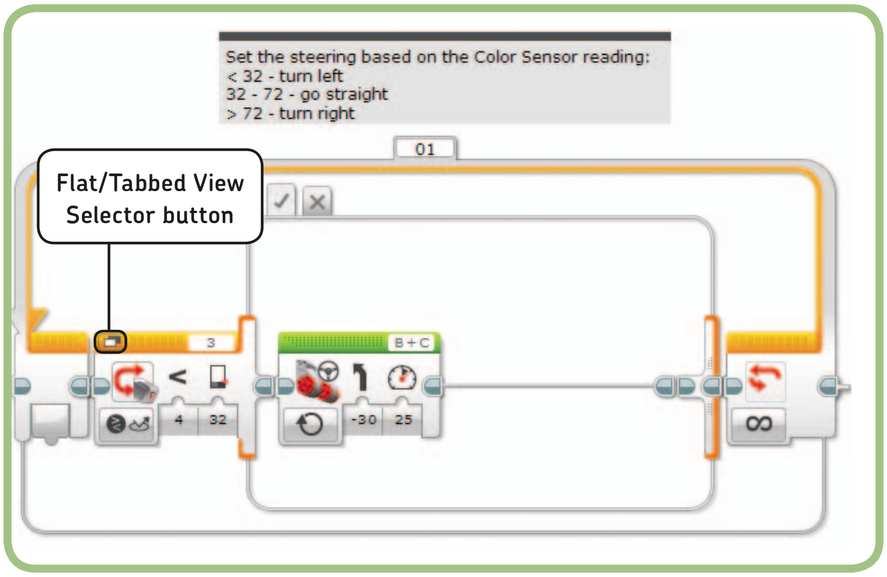

Set the steering based on the Color Sensor reading:
< 32 - turn left
32 - 72 - go straight
> 72 - turn right

Figure 6-10: The Switch blocks as seen in Tabbed View

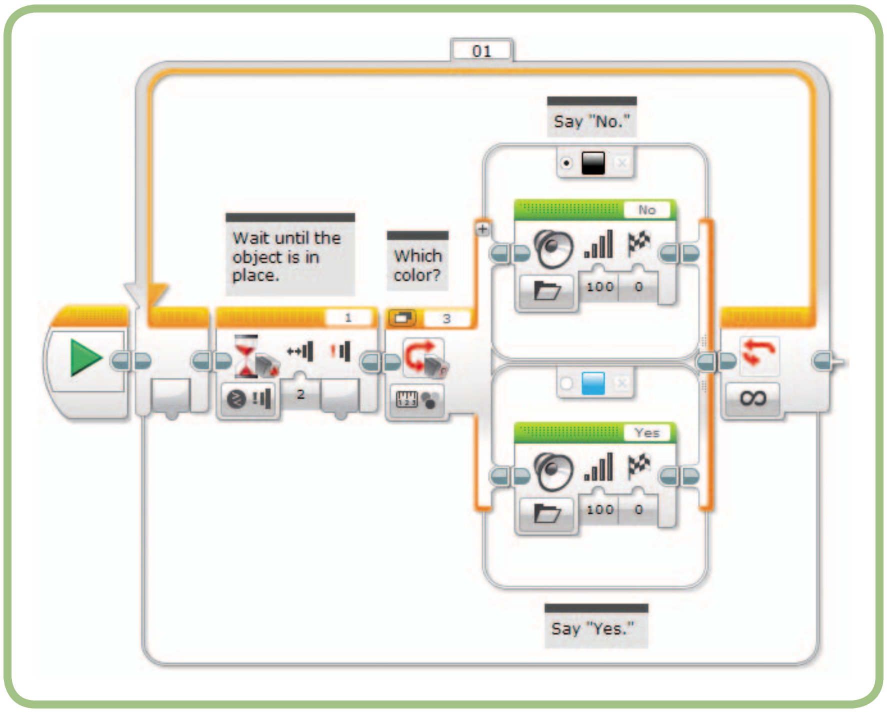


Figure 6-11: The starting point for the RedOrBlue program

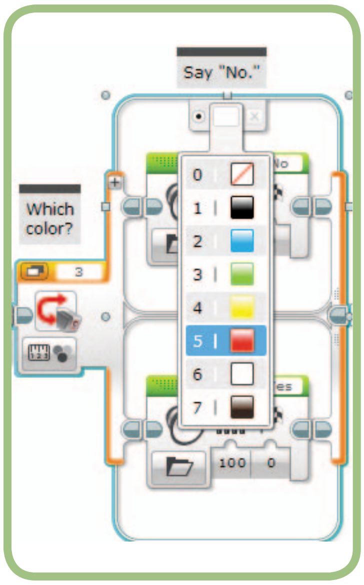


Figure 6-12: Selecting Red for the upper case

1. Open the *Chapter6* project if it's not already open.
2. Open the *Chapter5* project.
3. Open the Project Properties page for the *Chapter5* project by clicking the small wrench icon at the left side of the program tabs.
4. Select *RedOrBlue* in the list of programs.
5. Click the **Copy** button at the bottom of the window.
6. Select the *Chapter6* project.
7. Open the Project Properties page for the *Chapter6* project.
8. Click the **Paste** button. The *IsItBlue* program will be added to the project.
9. Open the *IsItBlue* program and rename it to *RedOrBlue*.
10. Close the *Chapter5* project.

identifying red objects

The lower case of the Switch block already matches blue objects, so we'll use the upper case to match red objects.

1. Click the black box at the top of the Switch block and choose **Red**, as shown in Figure 6-12.

The yes-or-no response is reasonable for the *IsItBlue* program but isn't quite right for a program that identifies more colors. Change the Sound blocks so that the program says "Red" for red objects and "Blue" for blue objects.

2. Select the Sound block on the upper case and change the sound file to *Red*.
3. Select the Sound block on the lower case and change the sound file to *Blue*.

The program should now look like Figure 6-13. When you run the program, it should correctly identify red or blue objects.

adding a new case

So far, this program only really identifies blue objects, because it will declare any nonblue object to be red (because red is marked as the default case). In this section, you'll modify the program to correctly identify red objects and say "Uh-oh" when it can't determine an object's color. The Switch block currently has two cases: one for red objects and one for blue objects. The *Add Case button*, highlighted in Figure 6-14, will add a new case to the Switch block. If you add a case and then later want to remove it, click the button with the small *x* on the right side of the tab at the top of the case.

NOTE The Add Case button only appears on the Switch block in modes where there are more than two possible cases.

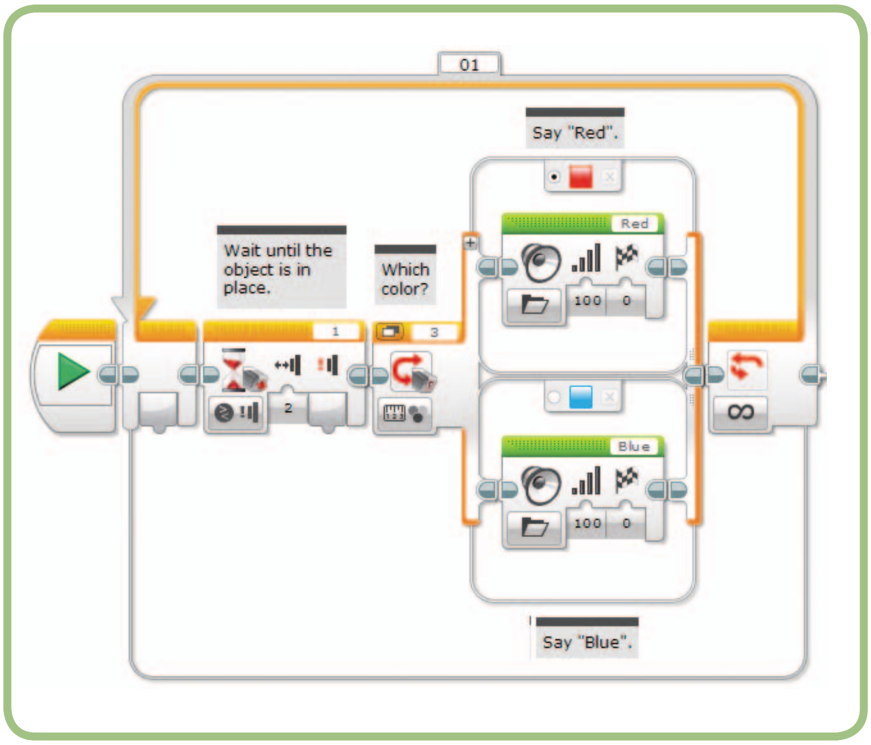


Figure 6-13: Identifying red or blue objects

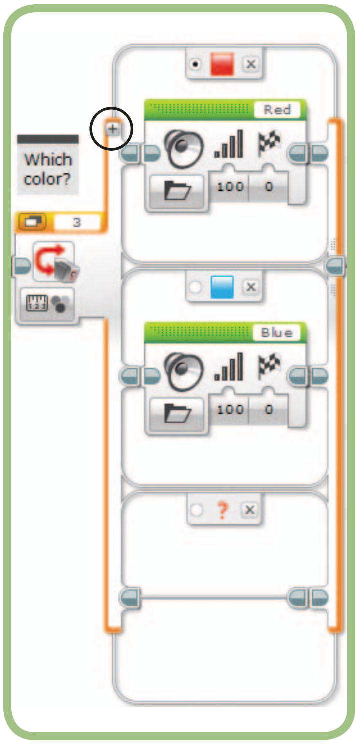


Figure 6-14: Adding a new case

Now, back to our program:

4. Click the **Add Case** button. The Switch block should look like Figure 6-14.
5. Click the red question mark on the tab at the top of the new case and a menu appears. Select the **No Color** option.
6. Add a Sound block to the No Color case.
7. Click the **Sound File** box and select the *Uh-oh* file. You'll find it under the Expressions group. The program should now look like Figure 6-15.

the default case

Right now, the red case is the default case in our program. This means that when none of the cases are met, the program will run the blocks for the red case, and the program will say "Red" when the Color Sensor reads anything other than red, blue, or No Color (for example, yellow or green). It makes more sense to set the No Color case as the default. This way, the program will say "Uh-oh" whenever it doesn't detect red or blue.

8. Click the **Default Case** button on the No Color case.

Figure 6-16 shows the final configuration of the Switch block.

CHALLENGE 6-2

Extend the *RedOrBlue* program to recognize all seven colors that the Color Sensor can detect.

the loop block

The *Loop block* lets you repeat a group of blocks over and over. The blocks inside a Loop block are called the *loop body*. You set a condition that controls how often the loop body is repeated and determines when the program moves on to the block that appears after the Loop block.

The modes for the Loop block, shown in Figure 6-17, include the same sensor modes as the Switch block, plus four additional modes:

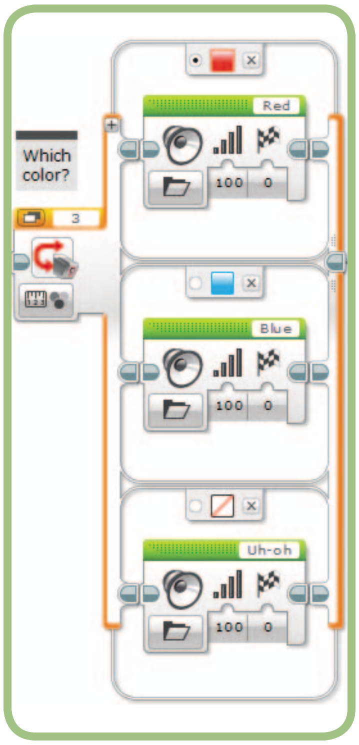


Figure 6-15: The No Color case

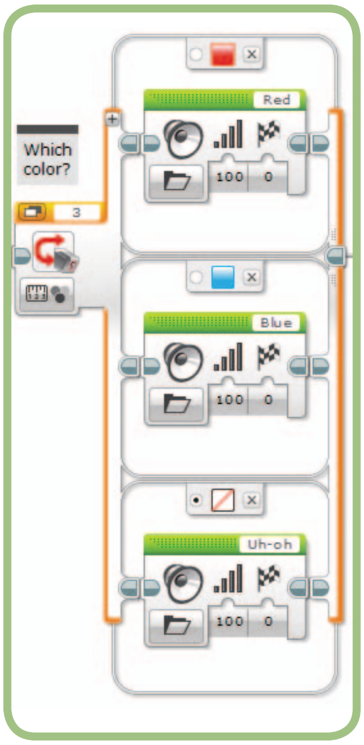


Figure 6-16: Setting the default to No Color

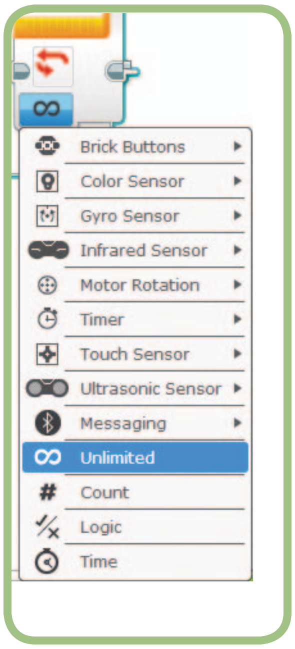


Figure 6-17: Selecting the Loop block mode

Unlimited The Loop keeps repeating until the program ends or the loop is exited using the Loop Interrupt block (discussed below).

Count The Loop repeats for the specified number of times.

Logic A value passed into the Loop using a data wire determines whether the Loop should exit. (Chapter 10 discusses using a Loop block with a data wire.)

Time The Loop continues for the specified number of seconds.

When a Loop block is in a sensor mode, the program checks the sensor measurement after the loop body runs. This means that the loop body is always run at least once. The program then decides whether to continue or leave the loop based on that sensor value. It doesn't matter what value the sensor reads while the loop body is running; only the sensor value measured at the end of the loop affects the program. If you're not careful, this can cause your program to fail. Because the Color Sensor is only checked at the end of the loop, it's very likely that the robot will move past the line without detecting it while the Sound block or Wait block is running. As an example, Figure 6-18 shows a flawed version of the *LineFinder* program that will often fail to find the line.

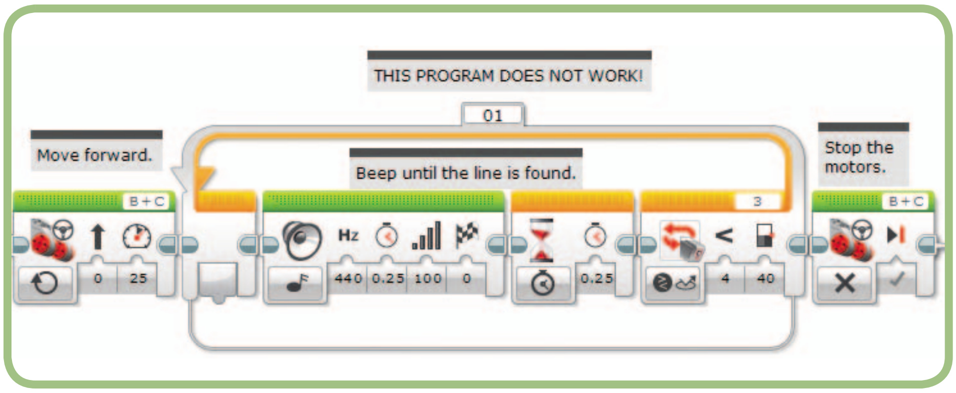


Figure 6-18: A broken LineFinder *program*

the loop interrupt block

The *Loop Interrupt block*, shown in Figure 6-19, gives you another way to exit a loop. The block only has one parameter: the name of the loop to exit. Each Loop block has a tab at the top of the block that holds the name of the loop, shown in Figure 6-20. A loop is named *01* by default; you can click the name to change it.

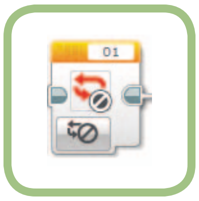


Figure 6-19: The Loop Interrupt block

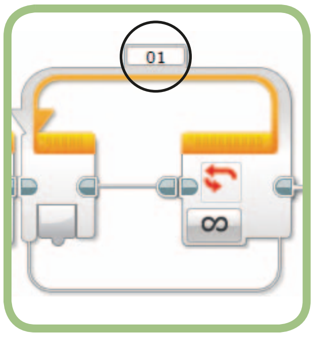


Figure 6-20: The Loop block name

The Loop Interrupt block shows only the first three letters of the name of the Loop block it will interrupt. As a result, it's usually clearer to use numbers or short abbreviations for the loop names; otherwise, it may not be clear which loop a Loop Interrupt block refers to.

the BumperBot3 program

Now you'll make some changes to the *BumperBot2* program to learn how the Loop Interrupt block works. To use this program, return the TriBot to its original configuration, with the Touch Sensor bumper mounted on the front and the Light Color Sensor on the side, as shown in Figure 6-21.

The changes you'll make will exit the loop and stop the program when you turn off the lights in the room. Figure 6-22 shows the part of the original program that we'll change. So far, the program makes the robot move forward and wait for the Touch Sensor to be pressed. When the sensor is pressed, the robot backs up, and the program loops back to the beginning.

Now you'll change the program to make the robot check the Color Sensor while it's waiting. If the Color Sensor reading is very low, indicating that the lights have been turned off, then the Loop Interrupt block exits the loop and the program ends because there are no more blocks after the Loop block.

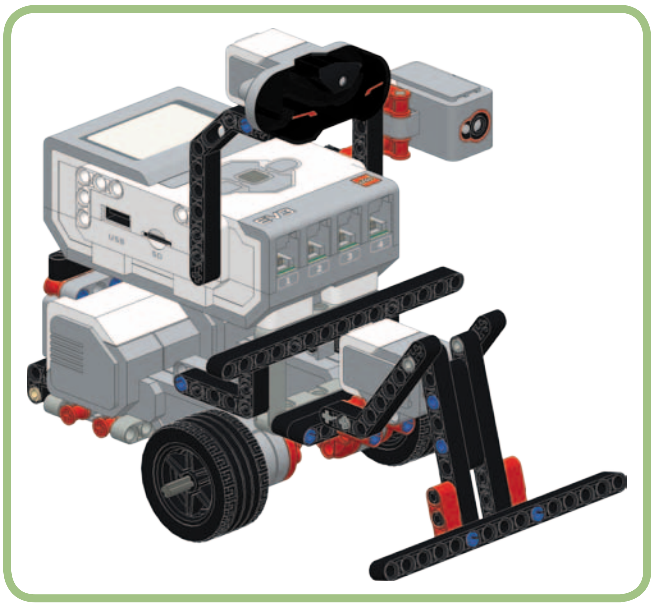
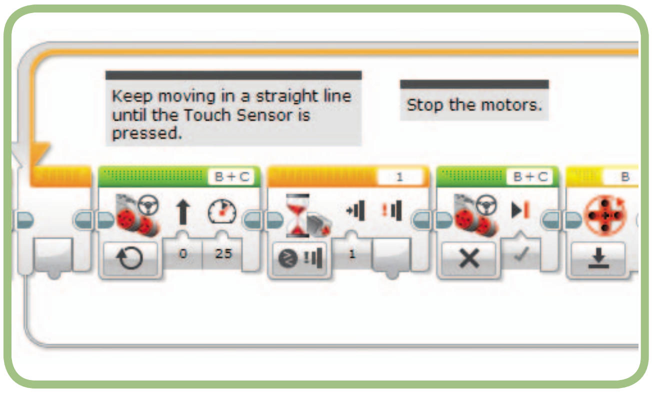


Figure 6-21: TriBot configuration for the BumperBot3 *program*



Keep moving in a straight line until the Touch Sensor is pressed.

Stop the motors.

Figure 6-22: The BumperBot2 *program*

Instead of using the Wait block to detect when the bumper is pressed, we'll use a Loop block, as shown in Figure 6-23, that repeats until the Touch Sensor is pressed. Inside the body of the loop, we'll use a Switch block in Ambient Light Intensity mode to check the ambient light level of the room. If the Color Sensor doesn't detect enough light, the program will stop the motors, say "Goodbye," and then exit the loop.

Follow these steps to create the new *BumperBot3* program shown in Figure 6-23:

1. Copy the *BumperBot2* program from the Chapter5 project to the Chapter6 project.
2. Change the name of the program to *BumperBot3*.
3. Change the name of the main Loop block to *02*.

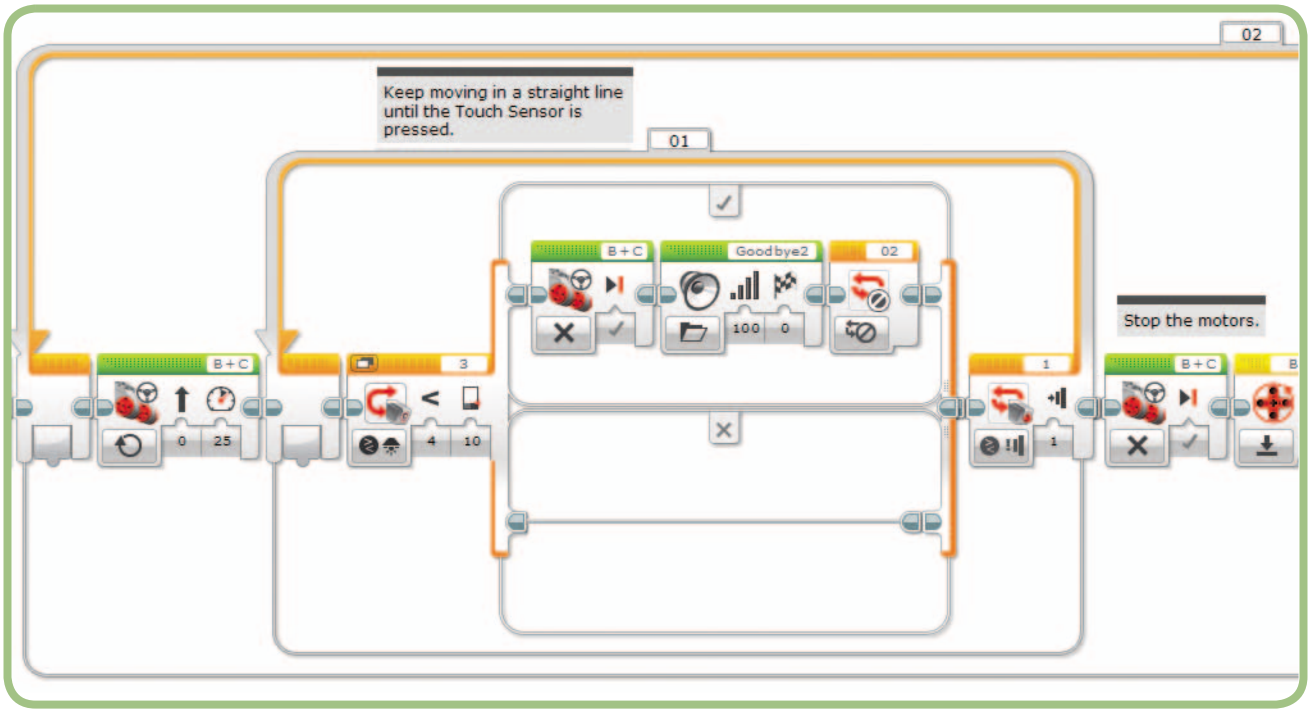

02
Keep moving in a straight line until the Touch Sensor is pressed.
01
B+C
Goodbye2
100
0
02
3
4
10
1
B+C
0
25
Stop the motors.
1
B+C

Figure 6-23: The BumperBot3 *program*

4. Delete the Wait block.
5. Add a Loop block where the Wait block was.
6. Set the Loop block's mode to **Touch Sensor – Compare – State**.

   This part of the program should now look like Figure 6-24.

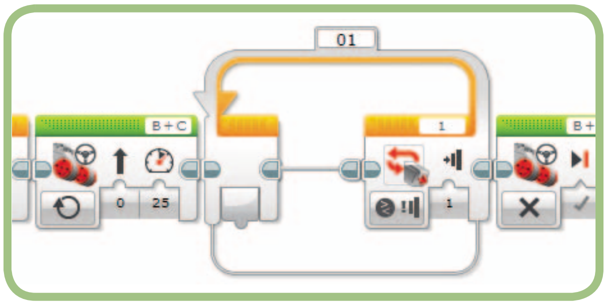


Figure 6-24: Replacing the Wait block with a Loop block

At this point, the empty Loop block does the same thing as the Wait block it replaced: It simply waits for the Touch Sensor to be pressed. But within the Loop block, we can now add some blocks that will run while the program is waiting.

The next step adds a Switch block inside the Loop block to check the Color Sensor:

7. Drag a Switch block into the Loop block.
8. Set the mode of the Switch block to **Color Sensor – Compare – Ambient Light Intensity**.
9. Set the Threshold value to **10**. You can adjust this value after some testing if it's too high or too low. The Loop block should look like Figure 6-25.

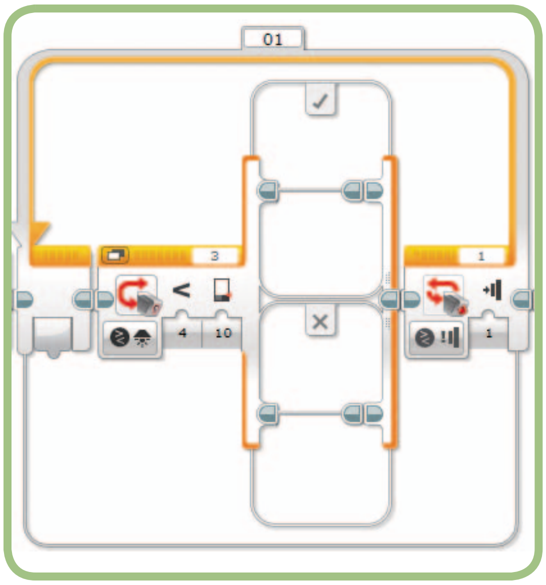


Figure 6-25: Adding the Switch block

When the Switch block runs, it checks the Color Sensor, and if the light intensity is below 10, the blocks on the upper case will run. Now add the following blocks:

10. Drag a Move Steering block to the upper case of the Switch block, and set the mode to **Off**.
11. Add a Sound block after the Move block. Select *Goodbye* from the list of sound files (it's under **LEGO Sound Files ▸ Communication**).
12. Add a Loop Interrupt block after the Sound block. Set the Loop Name to *02*.

Figure 6-26 shows the Loop block with these changes.

Now when you run the program, it should say "Goodbye" and stop if the lights in the room go out. To test the program, you can hold the TriBot in your hand, and after the wheels start moving, cover the Color Sensor with your other hand. When you cover the Color Sensor, the light level drops and the program should say "Goodbye" and exit.

further exploration

Now that you've learned the nuances of the Switch and Loop blocks, here are some activities to practice using them:

1. Write a program that makes the robot follow you—staying close to you but not too close, say between one and two feet. Use the Infrared or Ultrasonic Sensor and a Switch block to make the robot move forward or backward if you are too close or too far away. The robot should not move if you are in the desired range.
2. Use the Infrared Remote to add a pause and resume function to the *BumperBot3* program. Add a new Switch block that checks to see if a button on the remote is pressed. If it's pressed, the program should stop the motors, wait for another button on the remote to be pressed, and then start the motors again. The first button acts as a pause button, and the second button acts as a resume button.

conclusion

In this chapter, you learned to use the Switch block to make decisions, allowing you to choose between two or more groups of blocks. Either by nesting Switch blocks or by adding extra cases to a tabbed Switch block, you can extend the number of choices.

The other major flow control block is the Loop block, which allows you to repeat a group of blocks until a certain condition is reached. Much of the power of EV3 programs comes from the flexibility that's available when configuring blocks. The condition usually compares a sensor reaching a set Threshold value, but you can also configure the Loop block to repeat for a certain number of repetitions or a set amount of time. The Loop Interrupt block provides another way to exit a Loop block, which gives you even more flexibility to control how your program works.

At this point, you've seen how to use the EV3 motors and sensors and learned about the programming blocks you need to build sophisticated programs. In the next chapter, you'll use this knowledge to program the TriBot to find its way out of a maze.

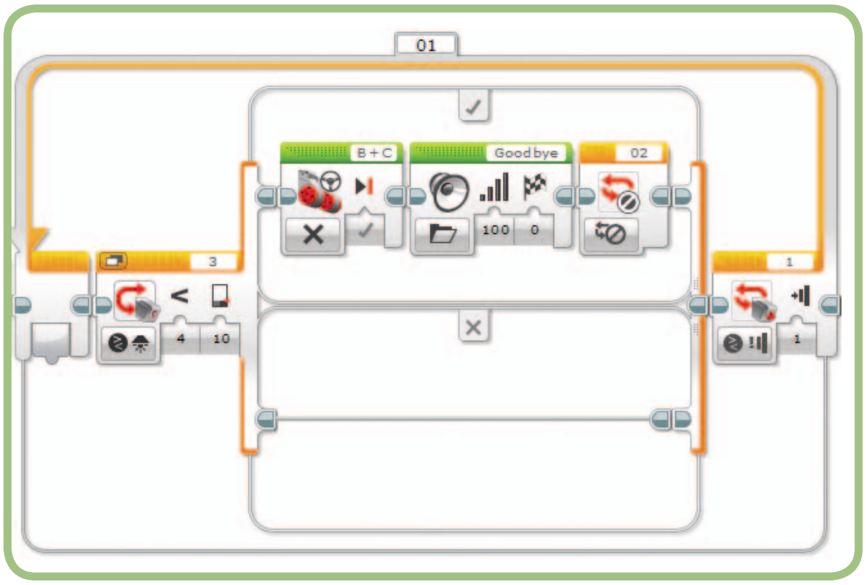


Figure 6-26: Stopping the motors and saying "Goodbye"