Lego Mindstorms Robot Workshop

University of Washington
Department of Computer Science & Engineering

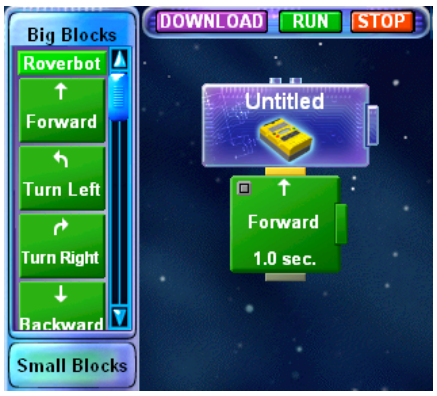
Now that you've heard a little bit about how to program the Lego robots, it's time to give it a try! Just as with human movement, any complicated robot movement can be broken down into a series of simpler movements. We'll start by getting familiar with the basic building blocks of robot movement.

A: BASIC MOVEMENTS

The robot's basic building blocks of motion are FORWARD, BACKWARD, TURN, and SPIN movements. Any movement, no matter how complicated, is built simply by stringing these basic movements together into a sequence. Let's try them!

All of the robot command blocks are found in the drawers on the left side of the screen. The basic movement blocks are green, and are found in the "Big Blocks" drawer. Click on the "Big Blocks" button to open the drawer.

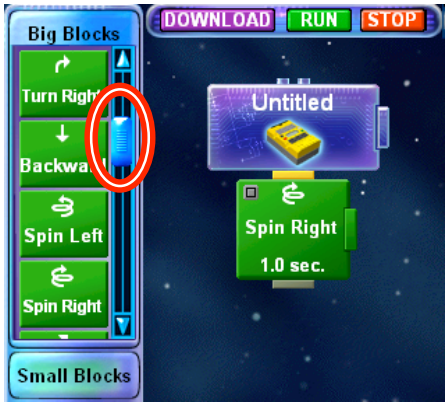
1. **Forward Movement**
   Write a program to make the robot move forward for one second. The program will look like the picture on the right. Once you've got your program set up, download it to the robot and give it a try!



2. **Backward Movement**
   Write a program to move the robot backwards. Try it out.

3. **Turning Movement**
   To turn the robot, you can either use the TURN blocks or the SPIN blocks. To make a complete turn, use the SPIN blocks. To find the SPIN blocks, scroll the window down by holding onto the blue bar (see picture on right) while dragging it down.

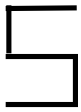
   Now write a program to make the robot turn right. Try it. Once it works, add another block to the program so that the robot turns right and then left.



B: MOVEMENT SEQUENCES

Using the basic movements you've just learned, you can make the robot move anywhere. Try the more complicated programs below, keeping in mind that any movement can be broken down into the pieces you already know!

4. **S-Shape**
   Write a program to make the robot follow the S-shape pictured on the right.



5. **Square**
   Write a program to make the robot move in a square. If you want, you can modify your program from #4 instead of starting over from scratch.

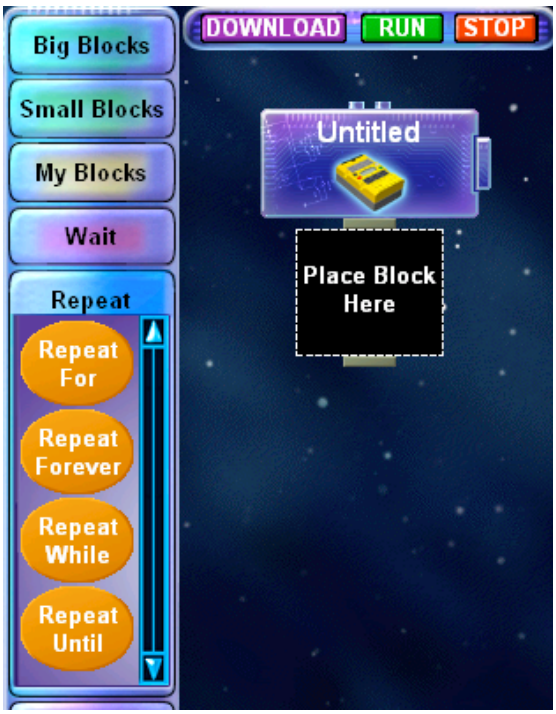
6. **Two Squares**
   Modify your program from #5 so that the robot goes along its square path twice instead of just once.

7. **One Hundred Squares**
   Now make the robot go in one hundred squares instead of just two. What kind of command might make it easier to write such a program? What kind of command would you need to make the robot turn in squares forever? Let's look at the next section before we actually try to make the robot make one hundred squares…

C: LOOPS

This section will introduce you to loop commands. Loop commands are commands that don't directly cause robot motion. Instead, they let you repeat the direct commands you've already written, so that you don't have to copy things over and over and over again.

The loop commands are orange, and are found in the "Repeat" drawer on the left side of the screen. Today we are only going to use two kinds of loops:



REPEAT FOREVER: You can probably guess what this repeat command does! That's right—whatever blocks you put inside this loop command will get repeated forever. As soon as the robot finishes executing the last command in the block, it will start over with the first one again and again and again and again…

REPEAT WHILE: This is just like REPEAT FOREVER, except that it is possible to stop this loop by changing some kind of condition. For example, if I told you to REPEAT *clap your hands* WHILE *the sun is out*, you would clap and clap and clap, but you would stop clapping when the sun went down. We will talk later about what kinds of things we can put in the WHILE part of this command to make the loop stop.

8. **Infinite Squares**
   Add a loop command to your square program so that the robot will go in squares forever. Try it out on the robot!

D: SENSORS

So far the robot has taken our commands and executed them. Every time we run a particular program, the robot does the same thing. But real robots—like the robots on Mars, for example—interact with their environment. They are able to sense things going on around them using *sensors*. When they detect specific things, they can change their behavior accordingly. For example, if a robot on Mars senses a big rock in front of it, it won't just try to move forward through the rock. It will go around it.

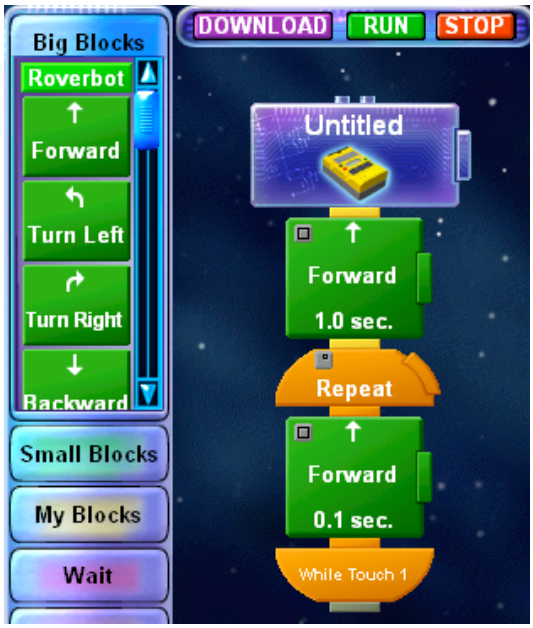
The Lego robots have a sensor just like the Mars robots do, to tell them when they are going to run into something. If you lift up the yellow bumper on the front of the Lego robot, you will see a little yellow bump. When the Lego robot hits something with its bumper, the bumper presses

down the little bump, and the robot detects that it has hit something! If the robot isn't hitting anything, the bumper hangs free and the bump is not pressed in.

The nice part about robot sensors is that they let us write programs that behave differently depending on the situation the robot is in. For example, we could write a program that makes the robot move forward if the bumper is not pressed, but move backwards if the bumper is pressed. This type of program lets the robot interact with its surroundings by changing its behavior based on sensor values.

9. Bulldozer Robot

Now that you understand the bumper sensor and loop commands, look at the program pictured at the right. What do you think it will do? What if you put a book in front of the robot?

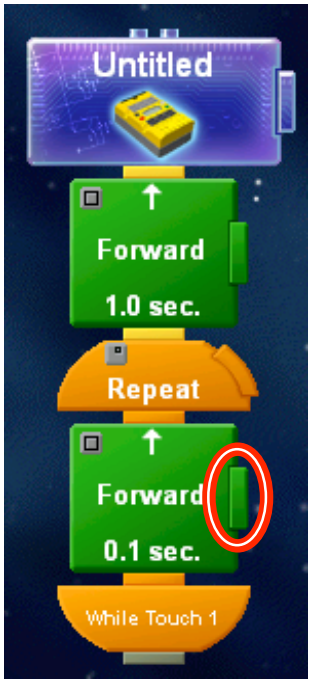


If you need to, look back in the previous section to find out how REPEAT WHILE loops work. The "While Touch 1" in the REPEAT WHILE loop in the picture means that the loop will keep going while the bumper is touched. Can you figure out what will make the loop stop?

Ask an instructor to come over and explain to the instructor what you think the robot will do when you run this program.

10. Bulldozer Robot In Action

Build the Bulldozer Robot program in the picture in #9.

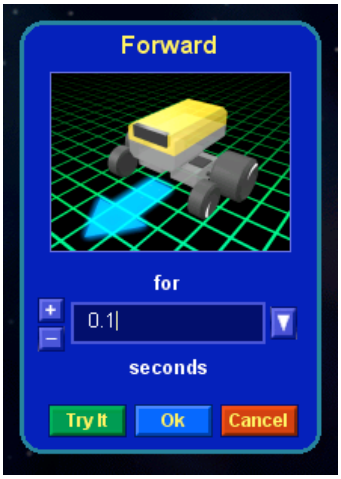


CONSTRUCTION HINT
The FORWARD block in the program picture makes the robot move forward for only 0.1 seconds instead of the usual 1.0 seconds. That's why the block says "0.1 sec." at the bottom. To make this happen in your own program, click the green box on the right side of the FORWARD block.

A box will pop up (see next page), and in the box you will be able to change the amount of time that the robot spends moving forward. Change this time to 0.1.

Test out the program on the robot. Does it do what you expected it to? If so, great job! If not, can you figure out why it's behaving the way that it is?

Once you understand the bulldozer robot's behavior, can you figure out a way to make it stop? Think about using the table edges. But please be careful not to let the robot fall off of the table!!!

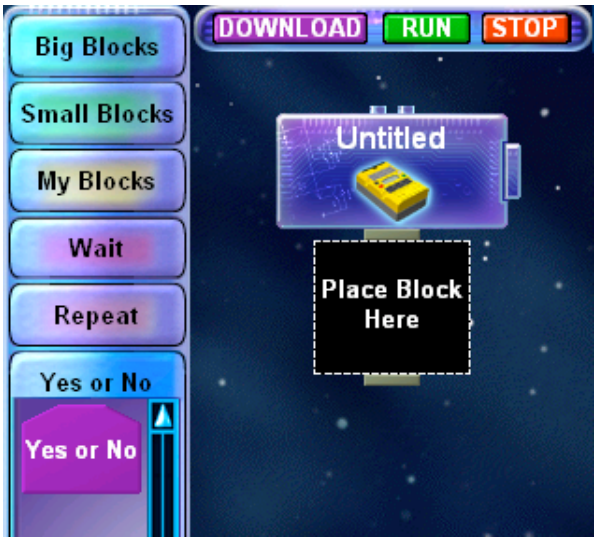


E: YES-NO (CONDITIONALS)

Now that we're experts on using the sensor to let the robot interact with its environment, let's look at another way to do it. We don't always want the sensor to make us stop doing something. Sometimes we want to use the bumper to let us pick between two different options. For example, maybe we want to move backwards if the bumper is pressed but forwards if it is not. We do this with a command called a conditional, also known as a YES-NO command.

YES-NO commands are found in the "Yes or No" drawer on the left side of the screen.

Take a YES-NO command block out of the drawer and put it in your work space. You can see that inside the block there are two separate paths. When the program reaches a YES-NO command, it will test a condition (in this case: is the bumper pressed?). If the answer is YES, the program will follow the YES path (on the left). If the answer is NO, the program will follow the NO path (on the right).



12. Forward-Backward Robot

Use a YES-NO command to write a program that makes the robot move backwards if its bumper is pressed when the program starts, but forward if its bumper is free. Test your program on the robot.

13. On-The-Move Forward-Backward Robot

Add a loop to your program from #11 so that the robot is always moving. If the bumper is free, the robot should be moving forwards. If the bumper is pressed, the robot should

be moving backwards. HINT: You only need to add one block to your program from #11 to make this work!

Test your program on the robot. Place an obstacle in its path to make sure that it moves backwards when the bumper is pressed.

14. Unstuck Robot

When you run your program from #12 with an obstacle in the way, you will notice that the robot keeps moving, but doesn't get very far. Why does the robot keep hitting the obstacle? Why is it stuck?

Modify your program from #12 so that the robot won't get stuck when it hits an obstacle. Think for a minute about what is causing the robot to get stuck, and then do your best to change the program so the robot can escape its pattern.

There are many different ways to do this. Your first idea may not work out quite right. That's ok! Keep trying different things until you can get your robot unstuck. If you need help or ideas, ask one of the instructors.