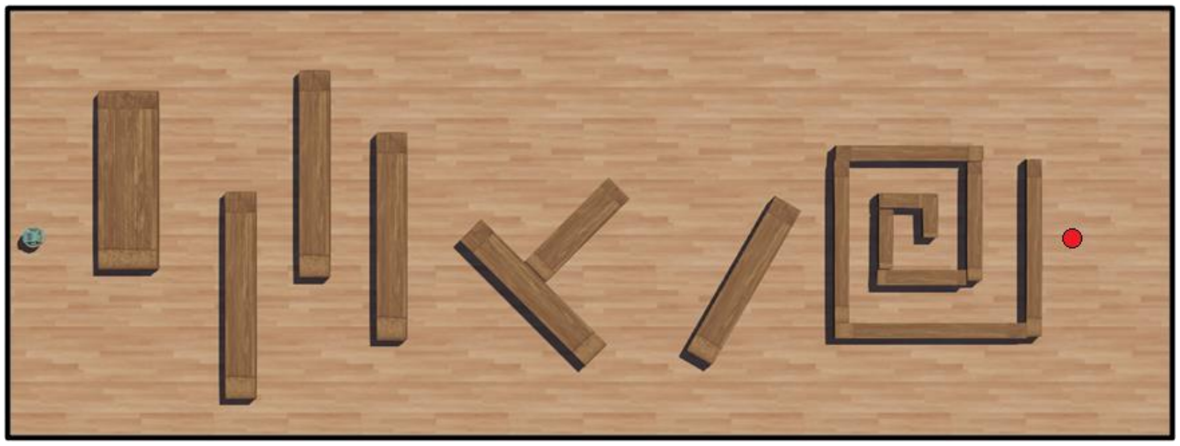
LAB 9 – Navigation in Unknown Environments

(1) Download the **Lab9_NavigationInUnknownEnvironments.zip** file and unzip it. Load up the **ObstacleCourse1** world. The world should appear as shown below. You may have to turn on **3D View** from the **Tools** menu.



The goal of this lab is to get the robot to perform the **Bug2 algorithm** to move from the starting location **(-191, 0)** to the goal location **(170, 0)** shown in red.

(2) The **Lab9Controller** code has been started for you. It causes the robot to perform wall-following, while regularly computing its location by using forward kinematic equations and a compass. You will need to alter the code so that the robot performs the **Bug2 algorithm** as described in the notes. I have made your task simpler by choosing the start and goal locations such that they share the same **y** value … so the "start-to-goal" line is horizontal (i.e., **y** = **0**). The code follows slide **15** in the notes which makes use of the "wall-following" state machine but adds two additional states:

(a) **Head To Goal** – causes the robot to go straight towards the goal unless it encounters an obstacle. This should be the starting state for your robot.

(b) **Orient To Goal** – causes the robot to spin left until it is facing the goal location again (within some angle tolerance, since it is difficult to spin to face exactly towards the goal accurately).

To complete the working state machine, you will need to do the following:

- Start by uncommenting the print statements at the top of the states in the first **switch** statement of the **main()** function. This will help you debug.

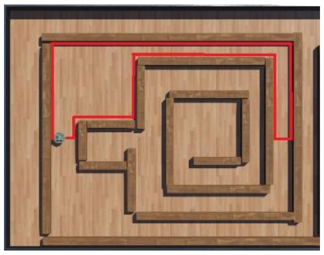
- Fill in the code in the 3 **switch** statements for both these new states. Your first task will be to get into the ORIENT_TO_GOAL state at the right time. This will be when the robot goes around the first obstacle and ends up back at the line (i.e., y = 0). You will need to change to this state from the STRAIGHT state as shown in slide 15. Make sure that works before continuing.

- The new state machine makes use of a **leftTheLine** boolean. This will keep track of when the robot has moved away from the line so that we don't start thinking it arrived back at the line when it actually never left. You will need to add code to the main **while** loop (in between the **switch** statements) that detects when the robot has gone some distance away from the line (in cm) based on some small tolerance value of your choosing. Once the robot has gone that distance away, you can set it to **true**. Then, it needs to be set to **false** when the robot starts wall-following again (i.e., when going from HEAD_TO_GOAL state back to SPIN_L state as shown on slide 15).

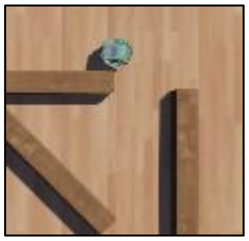
- In the main **while** loop, once it reaches the goal location, you will need to stop the robot by setting the motor speeds to 0, calling **robot.step(timeStep);** and then exiting the **while** loop.

(3) Once your code works, open the **ObstacleCourse2** world and make sure that your code still works. Don't panic when your robot encounters the vertical wall hanging from the top in the middle of the environment and then seems to go all the way back left again and into the spiral again … because that is normal … that is the way the algorithm is supposed to work. It should eventually get past that middle wall. If not, then you may have forgotten to include some special cases (see tips below). Your **Lab9Controller** code MUST work in both worlds without any modifications.

<u>Debugging Tips:</u>



- If your robot gets stuck a loop repeating the path shown here, then you forgot to apply the check mentioned in the yellow caption bubble on slide 15. You are heading out towards the goal again when you reach the left side again even though you are not closer to the goal than you were the last time you headed out towards the goal on the right side.



- If your robot does not reach the goal location with the same **Y** value as the start and stops at the location as shown here … then you likely forgot to check **both** X and **Y** coordinate values (within a certain tolerance).

Submit your **Lab9Controller.java** code. Make sure that your name and student number is in the first comment line of your code.

<u>Tips</u>:

- The robot will move very slowly. Make use of the **fast forward >>** when running. Also, the code runs quite slow when printing. Disable all debug print statements to make it run faster.

- You can always drag the robot sideways to another obstacle before running your code. This will allow you to test special cases. But when you reset the world, the robot will return to its start location on the left side. Do NOT save the world when the robot is moved, otherwise you will lose the original start location.