Lab 02 - Roomba Simulator

Dr. Mark R. Floryan

August 4, 2018

1 PRE-LAB

A while ago, I purchased a Roomba, a small round robot that scurries along my floor and vacuums. The robot is quite cute, and I always wondered precisely how the algorithm for determining its actions worked. For this lab, you will write some code that controls a virtual Roomba vacuum robot, and try to construct an algorithm that best cleans the floors of some virtual rooms. Your summary:

1. Download the starter code and import the project into Eclipse.
2. Implement the makeMove method in the Roomba class.
3. **FILES TO DOWNLOAD:** Roomba.zip
4. **FILES TO SUBMIT:** MyRoomba.java

1.1 DOWNLOAD AND REVIEW CONTENTS OF STARTER CODE

You can download the starter code for this project from the course repository here. Once you have done so, you should import the project into Eclipse.

Importing a project in Eclipse is easy. First, unzip the starter code somewhere on your machine. Then, within Eclipse, simply navigate to the **File –> Import** menu. Several options will appear next. Select **Existing Projects into Workspace** and click *Next*. Another dialog will appear asking you for the path to the folder where the project exists. Press **Browse** next to **Select Root Directory** and navigate to the project folder. Then, simply click **Finish** and you should see the project appear within the left hand bar of Eclipse.

There are several Classes that already exist in the project, that we enumerate here:

- package **main**
  - *Main.java*: Contains the main method. Instantiates a room, determines the room size, creates a Roomba with initial location in room, and starts the simulation.
  - *MyRoomba.java*: Your primary task is to **write the makeMove() method** in this class. See more details below.
- package **world**
  - *Move.java*: An enum that lists the valid moves a Roomba can make (move forward, turn clockwise, or turn counterclockwise). The makeMove() method should return one of these.
  - *RoomTile.java*: The different types of tiles that can exist in a room. A tile is either dirty, clean, or blocked (e.g., furniture is located there).
  - *Room.java*: Defines a full room, made up of many RoomTile objects.
  - *Roomba.java*: Defines the overall behavior of a Roomba. You may look at this class but you cannot change any of the code.
  - *RoombaGui.java*: Code for the visual interface of the project.
  - *RoombaSimulator.java*: Overall simulator that handles the number of cycles before Roomba must be done, handles Roomba's movement, etc.

Note that you should only be writing the **makeMove()** method in **MyRoomba.java**.

1.2 Implement Making Moves

The move making method has the following signature:

```
public Move makeMove();
```

Your only task is to write this method. We've provided a VERY simple implementation to get you started. Your method must return an instance of Move, specifically the move you'd like your Roomba to make this round.

Notice that your Roomba **cannot see the room directly**. This means, you need some other means of detecting the world around you. The actual Roomba uses several sensors to look at the environment. The simulator you are given provides the following:

- *this.frontBumper*: This boolean is set to true if the front of your Roomba has come in contact with a blocked room tile OR the edge of the room.
- *this.infraredSensor*: This integer provides the distance of the closest object. More specifically, the closest object is within this number of steps away from Roomba's center. *Note that for this calculation, a diagonal move is considered the same distance as a cardinal move.*
- *this.wallSensor*: This sensor (boolean) is set to true iff there is an object or wall directly to the right of Roomba's current position. This is useful for tracing along wall and around objects if desired.

You should use the variables listed above in order to make decisions about what your Roomba should do next. Roomba cannot look at the room / furniture layout directly.

1.3 Requirements

For submitting this pre-lab, your makeMove() method must adhere to the following requirements:

- you MUST have at least three if statements in your method.
- your Roomba MUST use some amount of randomness in decision making (i.e., your code must invoke Math.random() and use the result in its decision)
- you MUST use each of the sensor variables at least once in your code.
- your Roomba must ALWAYS clean more of the room than the simple Roomba we provided with the starter code.

2 In-Lab

The goal of this in-lab is to continue improving your coding abilities with some more programming challenges. As before, you will:

1. Get into small groups of two
2. The TAs will present you with various programming challenges. These will not be graded, but you are required to attend and to participate.
3. You may think about the challenges before lab below if you'd like, but we highly recommend that you not solve them ahead of time.
4. The TAs will give you time to solve each problem and lead you in sharing solutions with one another.

2.1 Programming Challenges

The TAs will lead you in attempting to complete the following challenges:

1. Given a two-dimensional square array of integers, return true iff the array is a Magic Square
2. Write a program that given an integer i, prints out the i'th row of Pascal's Triangle.
3. You are given a String containg numbers, letters, and question marks. Write a method that returns true iff there is at least one pair of numbers that adds up to 10 AND every pair of numbers that adds up to 10 contains exactly three question marks in between those numbers. For example, "acc?7??sss?3rr1??????5" would return true, and "aa6?9" would return false.
4. You are given a balance scale and three integer values: the weight currently on the left of the scale, the weight currently on the right of the scale, and an array of available weights that can still be placed on either side. Write a method that returns true iff the scale can be balanced by placing no more than 2 of the available weights onto the scale.

3 POST-LAB

The goal of this post-lab is to write a report comparing two distinct strategies for your Roomba simulator.

1. Construct at least two distinct strategies for your makeMove() method.
2. Run an experiment comparing how well each strategy cleans various rooms.
3. Write a report summarizing and analyzing your findings.
4. **FILES TO DOWNLOAD:** None
5. **FILES TO SUBMIT:** PostLabTwo.pdf

3.1 PERFORMING AN EXPERIMENT

Your first task is to write **two unique** makeMove() methods that incorporate different strategies for cleaning the rooms. You should be able to argue that the two strategies are significantly different AND that it is not obvious to you which is better. You should be geniuinely curious which approach will succeed.

You should then plan to execute your two Roombas several rooms, given the following constraints:

- You will use four total room configurations. two of the rooms will be smaller in size and two will be larger in size (you may choose the exact sizes).
- For each set of two above, one will have a small number of obstacles generated, and one will have a large number of obstacles generated.
- You should run each Roomba in each of the four configurations at least 15 times each. That is $15 * 4 = 60$ total executions minimum.
- For each room configuration, you should average the percentage of the room cleaned for each Roomba strategy.

3.2 REPORT

Summarize your experiment and your findings in a report. Make sure to adhere to these general guidelines:

- Your submission MUST BE a pdf document. You will receive a zero if it is not.
- Your document MUST be presented as if submitted to a professional publication outlet. You can use the template posted in the course repository or follow Springer's guidelines for conference proceedings.
- You should write your report as if it is original novel research.
- The grammar / spelling / professionalism of this document should be sound.
- When possible, do not use the first person. Instead of "I ran the code 60 times", use "The code was executed 60 times...".

In addition to the general guidelines above, please follow the following rough outline for your paper:

- **Abstract**: Summarize the entire document in a single paragraph/
- **Introduction**: Present the problem, and provide details regarding the two strategies you implemented.
- **Methods**: Describe your methodology for collecting data. How many rooms, how many executions, how you averaged things, etc.
- **Results**: Describe your results from your execution runs.
- **Conclusion**: Interpret your results. Which strategy was better? Why was it better? Were you surprised? Was one strategy better in some situations and not in others? Why do you think that is? Notice that I'm not looking for a particular answer here. Show me that you can interpret what happened when you ran your code.

Lastly, your paper MUST contain the following things:

- A table (methods section) summarizing the different experimental groups and how many execution runs were done in each group.

- A table (results section) summarizing each experimental group and the average percent cleaned for each (as well as any other data you decided to collect).

- Some kind of graph visualizing the results of the table from the previous bullet.