CS4710: Artificial Intelligence
Homework 2: Path Finding

Introduction:

For this assignment, you will be implementing a path finding system within a basic simulator. The simulator code will be provided to you. You will first implement a basic pathfinding algorithm as discussed in class. You will then write another algorithm that deals with uncertainty (the robot has imperfect vision, etc.). This assignment will be done in Java. You will conclude by writing a short report on the pros and cons of the various algorithms you implement.

Background – The Simulator:

The simulator (it is VERY basic) will be given to you as a JAR file. You will attach this JAR to your code base for working on this assignment. An example Makefile is provided to help you see how we will run your code. The two classes of interest and their exposed methods are described below. You should study these before beginning to code.

World.java (package world)

World(String filename, boolean uncertainty)	*Constructs a world. A world is a rectangular grid. Every position can be O (the robot can move to this position), an X (the robot cannot move to this position), S (the unique start position), or F (the unique end position). Accepts a filename to a text file describing the layout of the world. I will provide an example text file for your reference. If uncertainty is false, a robot's ping (see Robot.java below) will always return the correct string at that position. If uncertainty is true, then the robots pings will sometimes return the incorrect value (and more likely to the further away the ping is from the robot.*
getStartPos()	*Returns the starting position of the robot*
getEndPos()	*Returns the destination position of the robot*
numCols()	*Returns the number of columns in the grid of the world*
numRows()	*Returns the number of rows in the grid of the world*

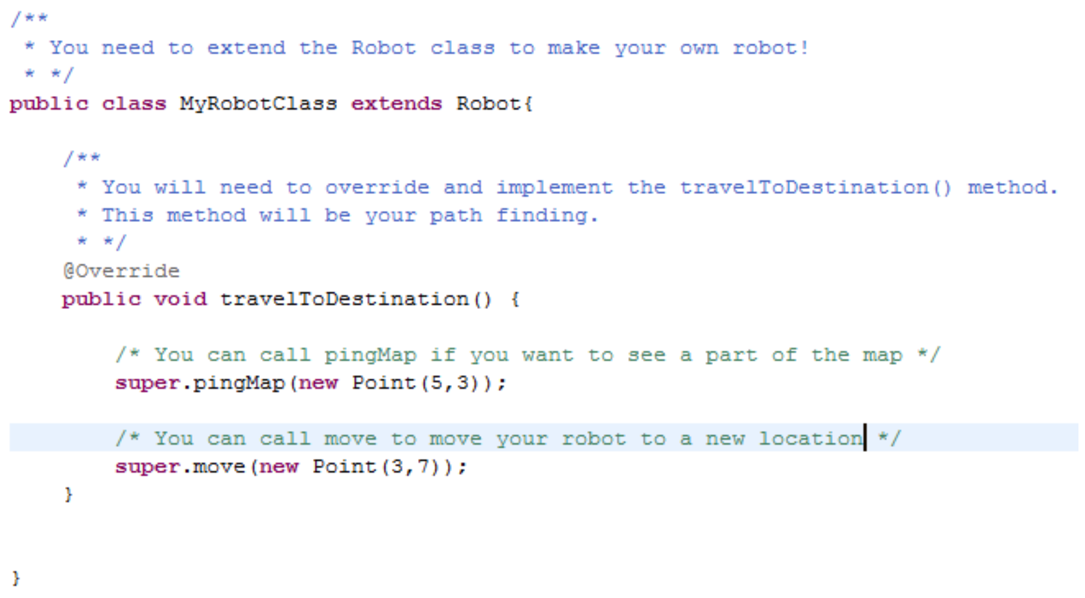
<continues on next page….>

Robot.java (package world)

addToWorld(World world)	*Adds this robot to the given world. Automatically moves the robot to the world's starting position.*
move(Point position)	*Attempts to move the robot to the given position. Position must be adjacent (diagonal allowed) to current position and a legal position. Returns the robot's new position after the move.*
pingMap(Point position)	*Robot attempts to view the map directly at the given position. Returns the string associated with that position. If uncertainty is true, this may return the incorrect result (more likely to be wrong if further away from robot). This position can be any place on the map.*
travelToDestination()	*Abstract method. You will need to override and implement this. Calls pingMap and move until the robot reaches the destination.*
getPosition()	*Returns the robot's position as a Point.*
getX()	*Returns the X position of the robot.*
getY()	*Returns the Y position of the robot.*
getNumMoves()	*Returns the total number of moves made so far.*
getNumPings()	*Returns the total number of pings made so far.*

Getting Started:

First, extend the Robot class and implement the travelToDestination() method. An example appears below:



```
/**
 * You need to extend the Robot class to make your own robot!
 * */
public class MyRobotClass extends Robot{

    /**
     * You will need to override and implement the travelToDestination() method.
     * This method will be your path finding.
     * */
    @Override
    public void travelToDestination() {

        /* You can call pingMap if you want to see a part of the map */
        super.pingMap(new Point(5,3));

        /* You can call move to move your robot to a new location */
        super.move(new Point(3,7));
    }


}
```

Then, test your code as such:

```
try{
    /* Create a world. Pass the input filename first. Second parameter... */
    /* ...is whether or not the world is uncertain.*/
    World myWorld = new World("myInputFile.txt", false);

    /* Create a robot that will run around in the World */
    MyRobotClass myRobot = new MyRobotClass();
    myRobot.addToWorld(myWorld);


    /* Tell the robot to travel to the destination */
    /* you will be implementing this method yourself! */
    myRobot.travelToDestination();

}
catch(Exception e){
    e.printStackTrace();
}
```

Once you call myRobot.move(position) and successfully move the robot to the final position, the simulator will automatically shut down and output a few simple stats to the console (number of moves, number of pings to the map).

Taking Input

For this assignment, input will be taken as a command-line parameter. This one command-line parameter will be the path to the input file that contains (note that the example above hardcodes the filename, but when you submit you'll need to use the command-line parameter here. You MAY NOT open that file yourself and read in the contents. Though this may allow you to pass the auto grader, a human will be looking at your code to make sure you haven't cheated here.

When you submit your code to Gradescope, we will run you file by issuing the following command:

make filename=./path/to/file.txt run

This means you MUST include a Makefile that contains a target called run that invokes your code with $filename as the command-line parameter. The zip file contains an example Makefile that does this for you if you'd like to use it as a reference.

What to submit

You will need to submit:

1) Your MyRobot.java file (you can call it whatever you like) that solves the certainty case
2) Your UncertainRobot.java file (but you might not compile this in with your Makefile)
3) The HW2.jar simulator jar file
4) A working Makefile
5) A pdf of your report (see below)

When you submit, the uncertainty case should be turned off.

Grading the certainty case

The certainty case will be autograded. We have a solution that does the following:

- Uses A* to find the optimal path to the destination location (number of moves is optimal)
- Uses Manhattan distance as the A* heuristic
- Only pings the map when absolutely necessary (i.e., pings only when a location is pulled off the priority queue and the robot is actually considering moving to that location
- Never pings the same location twice.

You will pass the test case if both of the following are true, let ourMoves and ourPings equal the number of moves and pings our Robot uses respectively:

- yourMoves == ourMoves *//Your number of moves is optimal*
- yourPings <= floor(ourPings*1.4 + 0.5) *//Your pings is within 40% of our number of pings*

Uncertainty:

Many interesting AI problems deal with uncertainty (and we will see this more later). When you are done implementing a basic working path finding, you should change your World constructor and pass in 'true' as the second parameter. Now make a second Robot class that can deal with uncertainty. Your robot, when pinging the map, may return the incorrect response. The robot's sensors are more likely to return the correct value though when the ping position is closer to the robot's current position. Keep this in mind.

Writeup:

Produce a document that describes, at a minimum, the following aspects of the assignment:

- Describe your basic path finding algorithm. Show a brief analysis of how well it works on a few different datasets that you produced. What kinds of data sets are more inefficient? Why is that the case?

- Describe how you adapted your algorithm when dealing with uncertain situations. How did you deal with the fact that the robot sometimes incorrectly viewed a space in the world?

- Produce data that shows how well your algorithm performs on different inputs. What happens if you slightly tweak or change your algorithm? How do these changes affect the performance and why?