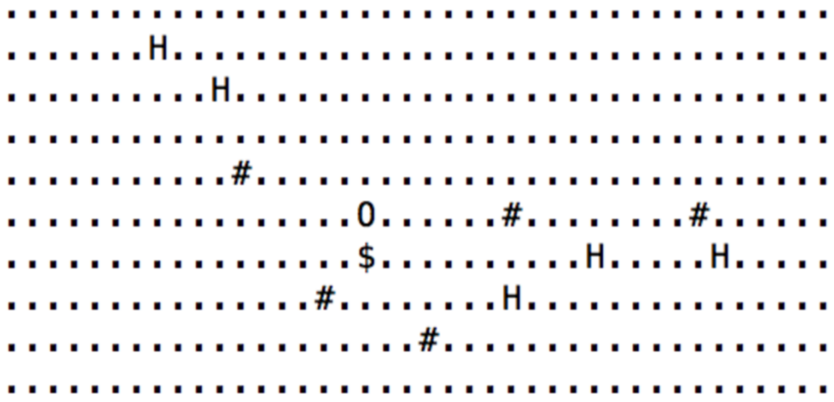
Project 1 (CSCI 360, Spring 2018)

7 Points

Due date: January 31st, 2018, 11:59:59PM PST

1 Introduction

In this part of the project, you will implement an A* search to navigate Wheelbot to its goal location on a map with obstacles. Some of these obstacles are initially known to the agent while some of them are hidden to the agent.



```
........................................
.......H................................
..........H.............................
........................................
...........#............................
.................0......#........#......
.................$..........H.....H.....
...............#........H...............
....................#...................
........................................
```

Figure 1: A screenshot of the text-based simulator. The environment is laid out in a discrete grid. Each period represents an unoccupied location in the environment. The robot's location is represented by the 0, while the target location is represented by \$. Known obstacles are represented by # and hidden obstacles are represented by H.

Before executing a move action, Wheelbot should use its local obstacle sensors to discover any hidden obstacles around it. Wheelbot should always follow a shortest path to its goal, assuming that the grid only contains the initially known obstacles and the discovered hidden obstacles (any hidden obstacles that have not been discovered yet, or the knowledge that there might be hidden obstacles, should not affect the path that Wheelbot follows). All movements have a cost of 1, and Wheelbot should minimize the total cost of the edges along its path to the goal. Wheelbot can move to any cell that does not contain an obstacle, and it dies if it moves to a cell that contains an obstacle. Above is a screen shot of the text-based simulator that you will use for this project. It has been extended to represent known obstacles (#) and hidden obstacles (H).

2 Simulator

2.1 Wheelbot

In this course, you will be working with a robot called **Wheelbot**, an abstract mobile robot encoded in the *Robot* class. At each simulation step, Wheelbot can read its sensors and then take one action, moving it one space in the given direction. For the purposes of this assignment, Wheelbot has the following actions and sensors:

2.1.1 Wheelbot Actions

1. **MOVE_UP**: Moves Wheelbot one grid space up (North)
2. **MOVE_DOWN**: Moves Wheelbot one grid space down (South)
3. **MOVE_LEFT**: Moves Wheelbot one grid space left (West)
4. **MOVE_RIGHT**: Moves Wheelbot one grid space right (East)
5. **STOP**: Stops Wheelbot for one time step in the current grid cell

2.1.2 Wheelbot Sensors

1. **Wheelbot Position Sensor**: Provides the (x, y) coordinate of Wheelbot on the grid. To use, call *this→simulator.getRobot()→getPosition()*.
2. **Target Position Sensor**: Provides the (x, y) coordinate of the target on the grid (the $). To use, call this→simulator.getTarget().

3. **Target Distance Sensor**: Provides the Euclidean distance between the target and the robot's current location on the grid. To use, call this→simulator.getTargetDistance().

4. **Local Obstacle Sensor**: Returns the 2D positions of all the obstacles in the four cells around Wheelbot. To use this sensor, call this→simulator.getCurrentLocalObstacleLocations(), which returns a vector of 2D points.

2.1.3 Other functionality

The location of the robot, target, and each obstacle are returned as instances of the *Point2D* class that has the member variables x (row) and y (column). The top-left corner of the map has coordinates (0,0). We have also added several new functions to the *Simulator* class, which you can use to obtain information about the environment:

1. You can get the dimensions of the grid by calling this→simulator.getWidth() and this→simulator.getHeight(). Both functions return integers.

2. You can get a list of all the initially known obstacles by calling this→simulator.getKnownObstacleLocations(), which returns a vector of 2D points. Note that this function will not return any hidden obstacles that have been discovered by the agent.

3 Programming Portion

This project will require you to modify the files Project.h and Project.cpp in the project source code (again, this includes filling out the **Project::getOptimalAction()** function). **You are not to modify any other file that is part of the simulator. You can, however, add new files to the project to implement new classes as you see fit.** Feel free to look at Robot.h, which defines Wheelbot, Simulator.h, which defines the environment, and Vector2D.h, which provides 2D vectors and points. We will test your project by copying your Project.h and Project.cpp files, as well as any files you have added to the project, into our simulation environment and running it.

Feel free to use the C++ STL and STD library data structures and containers. Additionally, if you have previously implemented data structures you wish to re-use, please do. However, you must not use anything that trivializes the problem. For instance, do not use a downloaded A* algorithm package. You must implement A* and the extensions yourself.

The provided Project.h and Project.cpp files include a skeleton implementation of the Project class, with the following functions:

1. *Project(Simulator &sim)*: This is the constructor for the class. Here, you should query the simulator for the dimensions of the map and all the initially known obstacles, and store this information. main.cpp will call the constructor before the simulation begins.

2. *RobotAction getOptimalAction()*: This is the function that will be called by main.cpp at each step of the simulation to determine the best action that Wheelbot should execute. In this function, you should first check Wheelbot's local obstacle sensors to discover any hidden obstacles around it and update your representation of the environment with the discovered obstacles (if any). Then, you should run an A* search on this (updated) environment to find a shortest path for Wheelbot and return the first action it should execute to follow this path. Use the Euclidean distance to the goal as the heuristic for A^*.

You should modify these two functions and implement an A^* search (either as a new class, or part of the *Project* class). For full credit, your robot must always follow a shortest path to the goal **with respect to the obstacles it currently knows about**.

4 Theory Questions

1. If there are no hidden obstacles in the environment, what guarantees can one make about the optimality of the path returned by the A^* algorithm? Would Wheelbot always follow a shortest path to the goal?

2. Now, consider the addition of hidden obstacles. Does Wheelbot still always follow a shortest path (from its first starting position to the target)? Explain why or why not.

3. How would your analyses in the last two questions change if we had used Breadth-First Search (BFS) rather than A^* for this project?

5 Submission

For this project, please submit a zip archive of your modified code via Blackboard by the date and time listed at the top of this document.