Project 2

Due March 10th, 2017 11:59PM

1 Navigating Wheelbot in Partially Known Environments - 10 Points

In Project 1, we saw that moving an agent from a starting state or location to a goal state or location using local and global sensors is one of the most fundamental tasks in Artificial Intelligence. 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 and some of them are hidden.

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). **Contrary to Project 1, in this project, diagonal movements have a cost of 1.5 (the rest have cost 1), and Wheelbot should minimize the total cost of the edges along the path, rather than the total number of edges traversed**. 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.

Below 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).

1.1 Simulator Details

In this course, you will be working primarily 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

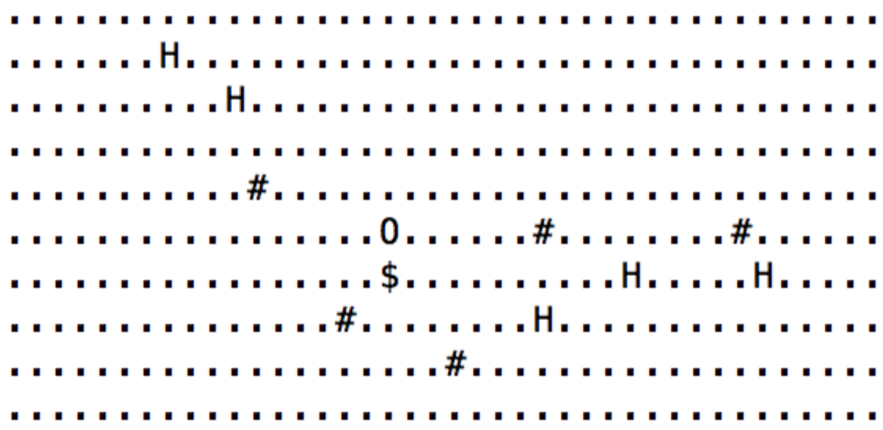

Figure 1: A screenshot of the text-based simulator. The environment is laid out in a discrete grid. Each period represents an unoccupied location in the environment. The robots location is represented by the 0, while the target location is represented by $. Known obstacles are represented by # and hidden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

space in the given direction. For the purposes of this assignment, Wheelbot has the following actions and sensors.

1.1.1 Wheelbot Actions

1. MOVE_UP: Wheelbot moves up one space in the grid (to use, call $r1 \rightarrow setRobotAction(MOVE\_UP)$)
2. MOVE_DOWN: Wheelbot moves down one space in the grid (to use, call $r1 \rightarrow setRobotAction(MOVE\_DOWN)$)
3. MOVE_LEFT: Wheelbot moves left one space in the grid (to use, call $r1 \rightarrow setRobotAction(MOVE\_LEFT)$)
4. MOVE_RIGHT: Wheelbot moves right one space in the grid (to use, call $r1 \rightarrow setRobotAction(MOVE\_RIGHT)$)
5. MOVE_UP_RIGHT: Wheelbot moves up and right diagonally one space in the grid (to use, call $r1 \rightarrow setRobotAction(MOVE\_UP\_RIGHT)$)
6. MOVE_UP_LEFT: Wheelbot moves up and left diagonally one space in the grid (to use, call $r1 \rightarrow setRobotAction(MOVE\_UP\_LEFT)$)

7. MOVE_DOWN_RIGHT: Wheelbot moves down and right diagonally one space in the grid (to use, call $r1 \rightarrow setRobotAction(MOVE\_DOWN\_RIGHT)$)

8. MOVE_DOWN_LEFT: Wheelbot moves down and left diagonally one space in the grid (to use, call $r1 \rightarrow setRobotAction(MOVE\_DOWN\_LEFT)$)

These are the same actions as in Project 1. r1 is an instance of Wheelbot, and it will be provided for you to use in the code you need to modify.

1.1.2 Wheelbot Sensors

1. Robot Position Sensor: Returns the 2D position of the robot. To use this sensor, call $r1 \rightarrow getPosition()$, which returns a 2D point. This is a local sensor of the Wheelbot.

2. Target Position Sensor: Returns the 2D position of the target. To use this sensor, call $sim1 \rightarrow getTarget()$, which returns a 2D point. This is a global sensor, which returns the absolute target location regardless of Wheelbots position.

3. Local Obstacle Sensor: Returns the 2D positions of all the obstacles in the eight cells around Wheelbot. To use this sensor, call $r1 \rightarrow getLocalObstacleLocations()$, which returns a vector of 2D points.

sim1 is an instance of the simulation environment and will be provided for you in the code you need to modify. The location of the robot and the target are returned as instances of the Point2D class that has the 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 $sim1 \rightarrow getWidth()$ and $sim1 \rightarrow getHeight()$. Both functions return integers.

2. You can get a list of all the initially known obstacles by calling $sim1 \rightarrow getKnownObstacleLocations()$, which returns a vector of 2D points. Note that this function will not return any hidden obstacles that have been discovered by the agent. You are responsible for keeping track of discovered local obstacles.

1.2 Project Details/Requirement

This project will require you to modify the files Project2.h and Project2.cpp files in the project source code. 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 Project2.h and Project2.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 reuse, 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 Project2.h and Project.cpp files include a skeleton implementation of the Project2 class, with the following functions:

1. Project2(Simulator* sim1): 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 (preferably by constructing the appropriate 2D grid). main.cpp will call the constructor before the simulation begins.

2. RobotAction getOptimalAction(Simulator* sim1, Robot* r1): 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.

You should modify these two functions and implement an A* search (either as a new class, or part of the Project2 class). For full credit, your robot must always follow a shortest path to the goal.

1.3 Submission

You should submit a zip archive of your modified source code to blackboard by March 10th, 2017, 11:59pm.