Project 0 (CSCI 360, Spring 2018)

3 Points

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

1 Introduction

Moving an agent from a starting state or location to a goal state or location is one of the most fundamental tasks in Artificial Intelligence. In this project, you will simulate a mobile robot performing this task on a discrete 2D grid using a simple text-based simulator. Below is a screenshot of the text-based simulator you will make use of in this course. Each dot represents a location in the environment that is currently not occupied. The 0 represents the location of the robot in the environment, while the $ represents the target location. The goal is to use the actions and sensors of the robot to move it from its current location to the goal location.

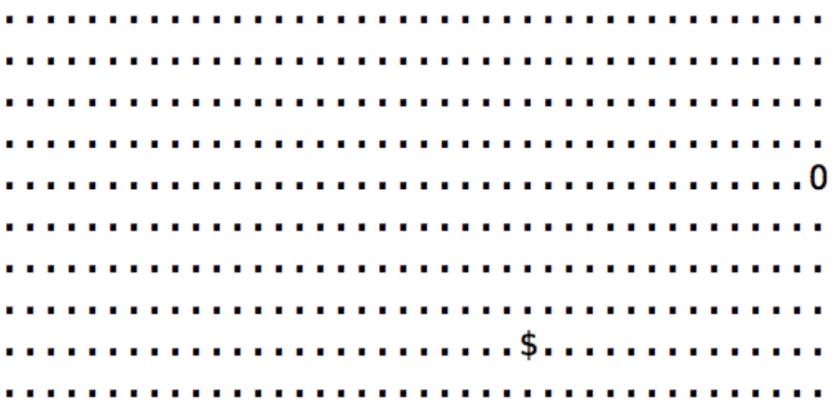

Figure 1: A screenshot of the text-based simulator. The environment is laid out in a discrete grid. Each period represents an unoccupied location in the environment. The robots location is represented by the 0, while the target location is represented by $.

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().

3 Programming Portion

This project will require you to modify the function **Project::getOptimalAction()** in the file Project.cpp in the source code provided for this project. **You are not to modify anything else in this file or any other file that is part of the simulator**. 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 function code into our simulation environment in the designated area and running it.

Any changes you make outside of the designated function will not be saved or counted as part of your submission. Your code must be standard C++ code that compiles using the provided CMakeLists.txt file on Linux or MacOSX. For this project, you should not need any headers, libraries, etc. outside of what is already provided in the Project.cpp file. The requirements of your **Project::getOptimalAction()** function are as follows:

1. Use the above sensors and actions to navigate Wheelbot from its random starting position to the target position.

2. Let a be the number of horizontal steps between the random starting position and the target position in terms of manhattan distance. Let b be the number of vertical steps between the random starting position and the target position in terms of manhattan distance. Your robot must take, at most, $a + b$ steps for full credit.

4 Theory Questions

1. The parameter *transitionProb* at the beginning of the the function *main* in the file main.cpp controls the determinism with which Wheelbot takes actions. If we set *transitionProb* $\leftarrow 0.9$, for example, with probability 0.1 Wheelbot will perform an incorrect motion action (e.g., moving left instead of up). Run your solution code a number of times with different values of this parameter (between 0 and 1) and explain Wheelbot's behavior. Does it always reach the goal in $a + b$ steps (at most). If not, why not? Explain how we might hope for our Wheelbot to overcome this problem. Should Wheelbot ever hope to navigate perfectly (in at most $a + b$ steps) if *transitionProb* < 1.0?

2. How could Wheelbot be made to handle the presence of obstacles on the grid (with known locations)? What if Wheelbot did not know the positions of the obstacles in the grid but had a local sensor that reported the existence of such obstacles when Wheelbot was one grid space away from them. How could Wheelbot be made to handle such "hidden" obstacles?

5 Submission

For this project, all you need to submit is your modified Project.cpp file. Please submit your modified file by the due date listed at the top.