Project 5: Artificial Intelligence for Robotics

Plotting and Navigating a Virtual Maze

Project Description

This project takes inspiration from Micromouse competitions, wherein a robot mouse is tasked with plotting a path from a corner of the maze to its center. The robot mouse may make multiple runs in a given maze. In the first run, the robot mouse tries to map out the maze to not only find the center, but also figure out the best paths to the center. In subsequent runs, the robot mouse attempts to reach the center in the fastest time possible, using what it has previously learned. This video (Youtube) is an example of a Micromouse competition. In this project, you will create functions to control a virtual robot to navigate a virtual maze. A simplified model of the world is provided along with specifications for the maze and robot; your goal is to obtain the fastest times possible in a series of test mazes.

Project Requirements

Requirements

- Python 2.7.X
- Numpy

Deliverables

- pdf report of algorithms used and approach taken to problem (see below)
- Python scripts for testing virtual robot (i.e. `robot.py` and any other created scripts)

Submit all necessary files in a single archive (e.g. .zip file) for evaluation.

Environment Specifications

Maze Specifications

The maze exists on an *n* x *n* grid of squares, *n* even. The minimum value of *n* is twelve, with maximum sixteen. Along the outside perimeter of the grid, and on the edges connecting some of the squares, are walls that block all movement. The robot will start in the square in the bottom- left corner of the grid, facing upwards. The starting square will always have a wall on its right side (in addition to the outside walls on the left and bottom) and an opening on its top side. In the center of the grid is the goal room consisting of a 2 x 2 square; the robot must make it here from its starting square in order to register a successful run of the maze.

Mazes are provided to the system via text file. On the first line of the text file is a number describing the number of squares on each dimension of the maze *n*. On the following *n* lines, there will be *n* comma-

delimited numbers describing which edges of the square are open to movement. Each number represents a four-bit number that has a bit value of 0 if an edge is closed (walled) and 1 if an edge is open (no wall); the 1s register corresponds with the upwards-facing side, the 2s register the right side, the 4s register the bottom side, and the 8s register the left side. For example, the number 10 means that a square is open on the left and right, with walls on top and bottom (0*1 + 1*2 + 0*4 + 1*8 = 10). Note that, due to array indexing, the first data row in the text file corresponds with the leftmost column in the maze, its first element being the starting square (bottom-left) corner of the maze.

Robot Specifications

The robot can be considered to rest in the center of the square it is currently located in, and points in one of the cardinal directions of the maze. The robot has three obstacle sensors, mounted on the front of the robot, its right side, and its left side. Obstacle sensors detect the number of open squares in the direction of the sensor; for example, in its starting position, the robot's left and right sensors will state that there are no open squares in those directions and at least one square towards its front. On each time step of the simulation, the robot may choose to rotate clockwise or counterclockwise ninety degrees, then move forwards or backwards a distance of up to three units. It is assumed that the robot's turning and movement is perfect. If the robot tries to move into a wall, the robot stays where it is. After movement, one time step has passed, and the sensors return readings for the open squares in the robot's new location and/or orientation to start the next time unit.

More technically, the robot will receive at the start of a time step sensor readings as a list of three numbers indicating the number of open squares in front of the left, center, and right sensors (in that order) to its "next_move" function. The "next_move" function must then return two values indicating the robot's rotation and movement on that timestep. Rotation is expected to be an integer taking one of three values: -90, 90, or 0, indicating a counterclockwise, clockwise, or no rotation, respectively. Movement follows rotation, and is expected to be an integer in the range [-3, 3] inclusive. The robot will attempt to move that many squares forward (positive) or backwards (negative), stopping movement if it encounters a wall.

Scoring

On each maze, the robot must complete two runs. In the first run, the robot is allowed to freely roam the maze to build a map of the maze. It must enter the goal room at some point during its exploration, but is free to continue exploring the maze after finding the goal. After entering the goal room, the robot may choose to end its exploration at any time. The robot is then moved back to the starting position and orientation for its second run. Its objective now is to go from the start position to the goal room in the fastest time possible. The robot's score for the maze is equal to the number of time steps required to execute the second run, plus one thirtieth the number of time steps required to execute the first run. A maximum of one thousand time steps are allotted to complete both runs for a single maze.

Project Flow

Task Specifications

You can find the starter code for the project linked in an archive here. Starter code for the project includes the following files:

- `robot.py` - This script establishes the robot class. This is the only script that you should be modifying, and the script that you will be submitting with your project.
- `maze.py` - This script contains functions for constructing the maze and for checking for walls upon robot movement or sensing.
- `tester.py` - This script will be run to test the robot's ability to navigate mazes.
- `showmaze.py` - This script can be used to create a visual demonstration of what a maze looks like.
- `test_maze_##.txt` - These files provide three sample mazes upon which to test your robot. Feel free to create your own mazes using the specifications above.

To run the tester, you can do so from the command line with a command like the following: `python tester.py test_maze_01.txt`. You should perform no modifications to `maze.py` or `tester.py`; the script you should be modifying is `robot.py`. If you create additional scripts, make sure that you include them in your submission.

The maze visualization follows a similar syntax, e.g. `python showmaze.py test_maze_01.txt`. The script uses the turtle module to visualize the maze; you can click on the window with the visualization after drawing is complete to close the window. You are free to modify `showmaze.py`, but it will not be used or looked at in evaluation of your project.

Report Questions

As part of the project submission, in addition to your Python script(s), provide a .pdf document answering the following questions regarding your approach to the project.

1. Document your work on the maze-learning part of your robot. What was your approach to your robot's first run through a maze? What algorithms or techniques did you use to decide where to move the robot next? How did your robot decide to end the exploration run? Were there any approaches that did not work?

2. Document your work on the fast-solution part of your robot. How did your robot make its second run through a maze? What algorithms or techniques did you perform? Did you have to make refinements and changes to your approach?

3. Consider if the scenario took place in a continuous domain. For example, each square has a unit length, walls are 0.1 units thick, and the robot is a circle of diameter 0.4 units. What modifications could you make to your robot's code to handle this added complexity? Are there other extensions for this project that you can think of, and potential ways of approaching these extensions?

Make sure that your document clearly states that your submission is for the "Plotting and Navigating a Virtual Maze" project, as there are many options for the capstone project for the Nanodegree program.

Project Evaluation

Your evaluator will review your submitted robot code and question responses using the capstone project rubric as a baseline. Your robot's code will be evaluated against versions of `maze.py` and `tester.py` as provided by the starter code archive. In addition to the three test mazes provided in the starter code, the robot will be tested against three additional mazes created under the same maze specifications. More specific impacts to the rubric are as follows:

Criteria	Meets Specifications
Problem definition	
Define your problem	In this project, the problem has been defined for you: that you are completing a function to help a virtual robot learn and navigate a maze environment.
Describe a solution	Responses to question 1 and 2 describe the general approaches to the robot-programming task.
Identify metrics	A performance metric has been defined for you: the score for a given maze is equal to the number of time steps required to complete the second run, plus one thirtieth the time steps used to complete the first run.
Exploratory analysis	
Analyze the problem	Responses to question 1 and 2 document the steps and thought processes taken in the student's approach to their problem solution.
Identify suitable machine learning algorithm(s)	Responses to questions 1 and 2 document specific solutions to each part of the problem task. Solutions demonstrate an understanding of algorithms that can efficiently learn and navigate the maze environment.
Solution implementation	
Pre-process the data	Three sample mazes have been provided in starter code to the project, as well as the general framework within which the project will be evaluated.
Implement and measure performance	Student's robot code consistently completes mazes (one learning run and one fast run) within a one thousand time unit limit. This includes the three sample mazes provided in the starter code and three novel mazes provided by evaluators.
Iterate and reflect	Responses to questions 1 and 2 document the revision process the student performed in their work. Responses to question 3 show how the student might expand the task performed in this project to more complex domains.

Resources and Links

Supporting Course

Artificial Intelligence for Robotics (cs373)

Additional Links

APEC Micromouse Contest Rules 2015