|  |  |
| --- | --- |
| Capstone Project  Machine Learning Engineer Nanodegree | Naoki Shibuya  March 18, 2016 |

Plot and Navigate a Virtual Maze

# Definition

## Project Overview

A robot mouse in a virtual maze finding a way to the destination by its own – that’s what I programed in this project that took an inspiration from the [micro mouse competition](micromouse%20competition).

## Problem Statement

The rule is simple: 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. The robot must enter to the goal within the time limit but it is free to continue exploring the maze after finding the goal. In subsequent runs, the robot mouse is brought back to the start location. It must attempt to reach the center in the fastest time possible, using what it has previously learned.

A simplified model of the world is provided along with specifications for the maze and robot. My only objective is to implement a logic that achieves the fastest times possible in a series of test mazes.

## Scoring

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 is allotted to complete both runs for a single maze.

# Analysis

## Data Exploration and Visualization

The shape of every test mazes is square. The start location is always at the left bottom corner (rows-1, 0), and the goal room always occupies 4 cells in the center.

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. It is assumed that the robot’s turning and movement is perfect.

At the start location, both left and right sides have walls so that it can only move forward. The sensors will return the distance between the robot and a wall in a tuple as in (left distance, forward distance, right distance).

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.

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.

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.

### ../../../../Downloads/maze01.pngMaze 01 (12x12)

The start location is (11, 0) and it is shown with the green ball. The goal area is located in the center and it is shown with the red balls. At the start location, the robot’s sensors will return (0, 11, 0).

The shortest path from the start location to the goal area is indicated the blue balls. It takes 30 single steps from the start location to the goal area. The robot is however allowed to take maximum 3 steps in one time step. Therefore, the minimum time steps required to arrive the goal should be 17. This also means the robot should prefer straight path than left/right turns.

### ../../../../Downloads/maze02.pngMaze 02 (14x14)

The robot should identify and avoid dead ends and any one-way path to it.

In this maze visualization, I marked dead ends with red X marks and one way path with purple balls.

### ../../../../Downloads/maze03.pngMaze 03 (16x16)

The robot should avoid going through a loop.

In this maze, I indicated loops in orange lines. There are many potential loops that the robot may fall into.

## Algorithms and Techniques

If the robot explores the entire maze, we can use A\* algorithm to find the shortest path from the start location to the goal area. Therefore, in the first run, the robot should try exploring the maze as much as possible and expand the mapping area. In the second run, the robot should apply the A\* search algorithm on the map.

For the 1st run, I will use the following techniques for the robot controller to explore the maze and expand the mapping area:

* Random move
* Dead-end detection
* Counting number of visits for each location
* Heuristic prediction of the distance to the goal

The random moves will take the robot to different paths randomly. It is not most efficient way to expand the mapping area but it gives the baseline performance to compare with more advanced techniques. Also, this controller will prove that the robot movements and rotations are handling walls properly.

The dead-end detection will prevent the robot to enter dead-ends more than once making it more efficient to expand the mapping area than the random controller.

The counts of the visits for each location will give the robot chances to move to less frequently visited locations. It will also make the robot moves out of loops.

The heuristic values will be used to make the robot move towards the goal area making it faster to reach the goal area. TODO? Should I use flood fill and forget about Heuristic in the first run ???

For the 2nd run, I will use the following techniques for the robot controller to exploit the mapped area and find the optimal path to the goal:

* G values and Heuristic values
* A\* search

G values give the distance of each location in the maze from the start position. Heuristic values gives the distance of each location in the maze from the goal area. A\* search uses the F values which are combination of G values and Heuristic values to find the optimal path from the start location to the goal area.

As the maze structure is well defined, I will use the test maze data to test the A\* search program. I’ll provide a separate python script to run the test without using the robot tester program.

## Benchmark

For the 1st run, I’ll provide different controllers (Random, DeadEnd, Counter, etc) to compare the scores which should improve as more advanced techniques are introduced. For the 16x16 maze, I’d expect the score to be less than 40.

# Methodology

## Data Pre-processing

The maze specification and robot's sensor data is provided and 100% accurate. Therefore, there is no data pre-processing is required.

## Implementation

The process for which metrics, algorithms and techniques were implemented has been thoroughly documented. Complications that occurred during the coding process are discussed in some detail.

In addition, student’s robot code consistently completes mazes (one learning run and one fast run) within a one thousand time step limit. This includes the three sample mazes provided in the starter code, the three mazes provided by evaluators, and .

## Refinement

The process of improving upon the algorithms and techniques used is clearly documented. Both the initial and final solutions are reported, along with intermediate solutions, if necessary.

# Results

## Model Evaluation and Validation

The final model’s qualities — such as parameters — are evaluated in detail. Some type of analysis is used to validate the robustness of the model’s solution.

## Justification

The final results are compared to the benchmark result or threshold with some type of statistical analysis. Justification is made as to whether the final model and solution is significant enough to have adequately solved the problem.

# Conclusion

## Free-Form Visualization

Use this section to come up with your own maze. Your maze should have the same dimensions (12x12, 14x14, or 16x16) and have the goal and starting positions in the same locations as the three example mazes (you can use test\_maze\_01.txt as a template). Try to make a design that you feel may either reflect the robustness of your robot’s algorithm, or amplify a potential issue with the approach you used in your robot implementation. Provide a small discussion of the maze as well.

## Reflection

First challenge for me was to divide the project into smaller problems to tackle with. I’ve decided to have separate test program for the A\* search program as I realized it does not require a running robot to test the search algorithm. Then, I’ve divided each enhancements to the robot controller into different python classes making it easier to add improvements in terms of coding and also the actual test scores. Throughout the project, I was making common tasks into utility classes so that I do not need to write similar logic or handling in different places (i.e. Direction, Steering, Sensor, Grid).

The next challenge for me was to find problems in expanding the maze mapping area. I had issues like dead-ends and looping. Over the time, I’ve added effective logging to analyse the robot behaviour and look for potential issues. This was largely a trial-and-error process for me since I did not have much experience in the maze solving problem before. It was a great learning process for me.

Finally, the most difficult and interesting part was how to expand the mapping area of the maze while reaching to the goal at minimum time required in the first run. In the second run, the robot simply uses A\* search to find the optimal path and travel to the goal accordingly. But the first run was full of challenges due to the unknown area to be explored. Overall, I built a series of controller to add improvement bit by bit making the process simpler than without such structural approach to the problem. I believe the final controller should give a good performance in the real micro mouse scenario with some improvement for continuous domain support.

## Improvement

In this project, everything (time, location, move and turn) is in a discrete domain. In the real micro mouse competition, everything is in a continuous domain.

For example, the distance from the robot to the wall is in continuous which would require the robot to perform SLAM (simultaneous localization and mapping) to explore the maze. The sensors will give the robot continuous values. The robot needs to use PID control to continuously adjust the moves and turns so that it can wander around in the maze without colliding with the walls. The speed needs to be controlled rather than just number of steps. Turns will be continuous rotations. Moreover, the robot may be able to move diagonally rather than zigzag which is not allowed in the discrete domain.

Talking about the real micro mouse competition, the robots are physical which adds many more complexity. The path finding logic is probably one of the easiest part of the whole robot construction. There are many aspects to take care: what sensors to use, what kind of motors and how heavy it can be, physical memory size available to use, etc. I could have a sensor rotating on top of the robot mapping neighboring areas simultaneously.