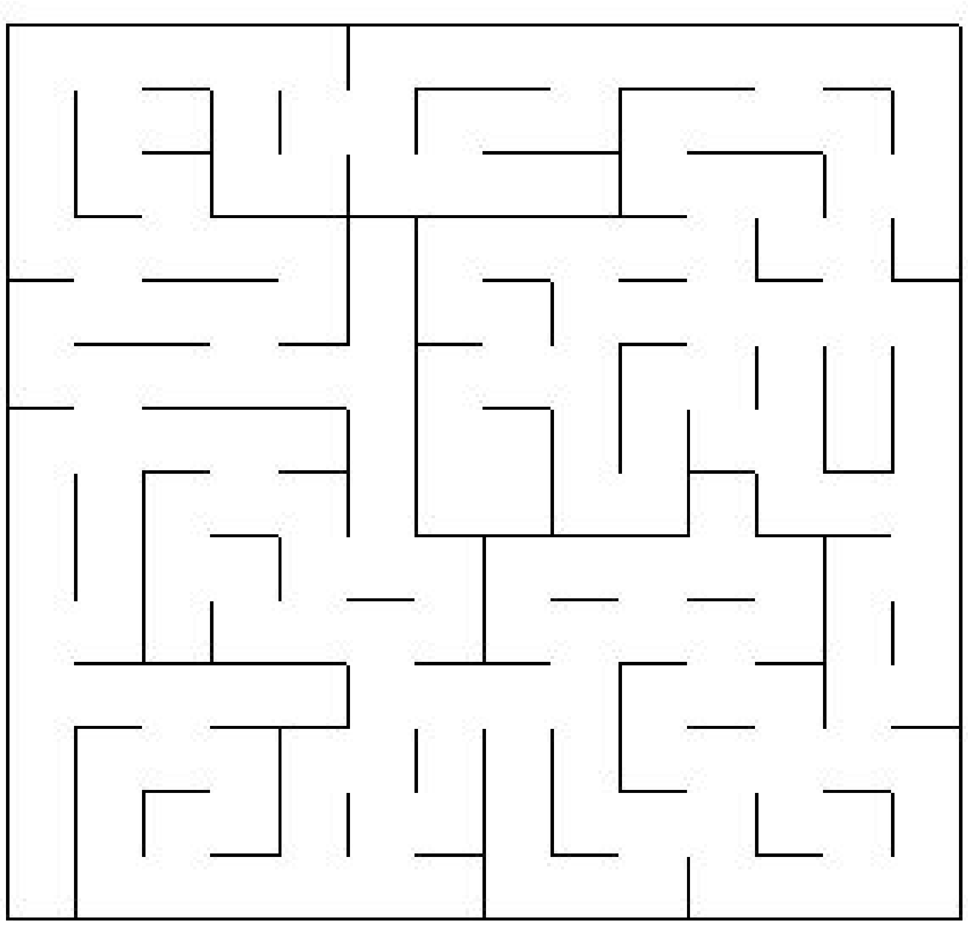
Machine Learning Nanodegree:

Shashwata Mandal

Capstone Proposal

August 8, 2017

Plot and Navigate a Virtual Maze



Domain Background

The Micromouse is a famous competition, 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 is made to map out the entire maze. In the this step, the robot visits each and every navigable portion of the maze no matter how long it takes. The main aim of this step is to figure out the best paths to the center. In subsequent runs, the robot mouse attempts to reach the center in the fastest time possible, based on what it learned previously.
The following video is an example of the Micromouse competition.
Having been interested in autonomous robotics, I have always been motivated to problems of this genre. This problem should be solved because the idea behind it governs a large section of the industrial automation industry, where optimization like this can save up a lot of energy hence cost. Refer to Related Works section.

Problem Statement

The maze in this problem is an n x n grid, where n is either 12, 14 or 16. The mouse begins its journey from the bottom left corner of the maze and must navigate a path to the center of the grid, which is a 2 x 2 square.
In each cell, the mouse can sense its distance from the nearest wall in three directions (left, forward and right). It can move at most 3 units in any direction (depending on where the walls are) and it has the option to rotate 90 degrees to the left or right before moving. A rotation and a movement defines one time step.
The main goal of the mouse is to find the shortest route to the center of the grid. The mouse is allowed a minimum of two runs at the maze. The first run has a scoring of 1/30 time point per cell movement whereas each of the successive runs has a scoring of 1 point per cell movement. The objective is to minimize this score.

Datasets and Inputs

The datasets are provided in the form of **test_maze_##.txt** input files which are provided to the system. 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 are **n** comma-delimited number describing sides of the square are open for navigation. Each number represents a four bit number, where bits represent open (if equal to 1; no wall) or close (if equal to 0; walled). The number is to be converted to binary for evaluation thereby getting a binary in the form of **LDRU** where U (1s register) represents upward-facing side, R (2s register) represents the right side, D (4s register) the bottom side and L (8s register) the left side. For example, the number 10 =$(1010)_b$ 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. For this problem, 3 sample mazes of sizes 12, 14 and 16 shall be used for testing.

Solution Statement

The solution for this problem is a two step one. No algorithm can ever compute the shortest path to the center of the maze in a single run without prior knowledge of the maze. Hence in order to traverse the maze optimally, we need to explore it first. The exploration of the algorithm is a variation of the age old depth first search with a slight modification ie. priority is given to an unexplored cell so that the maze can be navigated as quickly as possible. The program keeps track of the number of cells left to be mapped and ends the run as soon as all the cells are tagged. The main objective of this part is to recreate the maze data available in the input file.
The second part of the one where the mouse finds the optimum solution. For this the mouse needs to preprocess the data acquired before making a move. Initially a heuristic technique is applied which enables us to compute the shortest distance from the starting cell to each cell of the maze. To accomplish this we shall be using the Flood-fill approach similar to the one used in computer graphics. This is done via a simple iterative process. The starting cell is marked as 0. Then in each iteration, a cell marked with a number is used to compute the value of its unmarked navigable neighbours. The value assigned to the neighbours will be 1 plus the value of the current cell. This way at the end of the iterations, the goal cells are marked with a shortest path value. The next step is to compute the direction matrix for the mouse. The goal cell with the least value is selected. The next step is to trace back a step to the starting cell. For each cell the neighbour with a value equal to current cell value minus 1 is selected as its predecessor. In case of a tie between neighbour, one of them is randomly selected. The direction is computed based on the relative position of the current cell from the predecessor cell. Finally when it reaches the starting cell the computations are complete and the robot is ready to run in the shortest path.

Benchmark Model

In order to build a successful algorithm, it is necessary to set up some preliminary benchmarks so that we can determine what to expect from an optimal solution. Since a second round move cost 30 times more than a move in the first round, it is necessary to get as much as possible out of the first round. Depending on the size of the maze, the ideal path is approximated to maze 20-50 time steps. There are a maximum of 1000 time steps allowed and if the mouse uses 900 moves for the first round and 20-50 steps in the second round, then expected scores come to around 50-80. That is considered to be the initial benchmark.

Evaluation Metrics

The mouse has two runs through the maze. It has a limit of 1,000 time steps combined, and it can reset to the starting position any time after it has reached the goal in the first run. In the first run, each step cost 1/30 of a point, where as in the second run each step costs an entire point. Thus, it is beneficial to take the extra steps to map out the entire maze in order to take the minimum number of steps in the consecutive runs.

Project Design

The robot mouse we visualised here is in a virtual environment. Hence the entire environment needs to be simulated. The maze needs to constructed in memory and tested for efficiency. For this we require separate scripts to emulate the maze. Once the maze is setup up and running, a thirds script which implements the aforementioned algorithm is instantiated and the mouse virtually runs in the environment. The third script contains a Robot object into which all the attributes and AI of the mouse is manifested. The initial navigation by the mouse for mapping the maze using Depth first search as well as the preprocessing of the data during the second run for finding the shortest path is all performed in this script as well. As per my understanding, the flood-fill algorithm will give us the most optimum solution for the second run, hence it would be wise to minimize the cost of the initial mapping the maze as much as possible. The output is generated using CLI due lack of GUI but is thoroughly informative. Prior analysis of the data is required for visualizing the data and checking the algorithm via a dry run.
The following files will be utilized:

- robot.py - This script establishes the robot class. This is the script where the main algorithm is implemented.
- maze.py - This script will contain the functions for constructing the maze and for checking for walls upon robot movement or sensing.
- tester.py - This script will contain the logic for testing the robot's ability to navigate the mazes
- showmaze.py - This script will contain the codes used to create a virtual demonstration of what the maze looks like.
- test_maze_##.txt - These files as mentioned before are the input mazes upon which the robot will be tested.

Finally a measure of the score will be shown when the algorithm is tested using the tester.py script.

Related Works

1. http://ijsetr.org/wp-content/uploads/2016/09/IJSETR-VOL-5-ISSUE-9-2844-2848.pdf
2. http://moustafamohamed.com/papers/micromouse.pdf
3. https://www.researchgate.net/publication/224363078_Maze_Solving_Algorithms_for_Micro_Mouse