Robot Motion Planning

Plot and Navigate a Virtual Maze

I. Definition

A. Given an agent and a virtual maze, agent can make 2 runs in the maze. Agent can explore maze freely at the first run and figure out the best path to center, then agent should attempt to reach the center as fast as possible in the second run.

B. 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. This video (Youtube) is an example of a Micromouse competition.

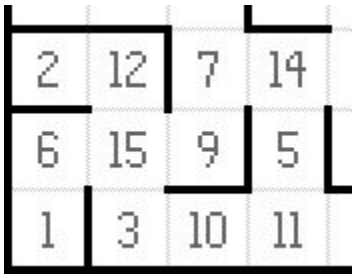
C. project requirements
   1. Python 2.7.X
   2. Numpy

D. A virtual maze means somethings below:
   1. test_maze_02.txt
      14
      1,5,5,7,7,5,5,6,3,6,3,5,5,6
      3,5,6,10,9,5,5,15,14,11,14,3,7,14
      11,6,11,14,1,7,6,10,10,10,11,12,8,10
      10,9,12,10,3,12,11,14,11,14,10,3,5,14
      11,5,6,8,11,7,12,8,10,9,12,9,7,12
      11,7,13,7,14,11,5,5,13,5,4,3,13,6
      8,9,5,14,9,12,3,7,6,3,6,11,6,10
      3,5,5,14,3,6,9,12,11,12,10,10,10,10
      10,3,5,13,14,10,3,5,13,7,14,8,9,14
      9,14,3,6,11,14,9,5,6,10,10,3,6,10
      3,13,14,11,14,11,4,3,13,15,13,14,10,10
      10,3,15,12,9,12,3,13,5,14,3,12,11,14
      11,12,11,7,5,6,10,1,5,15,13,7,12,10
      9,5,12,9,5,13,13,5,5,12,1,13,5,12

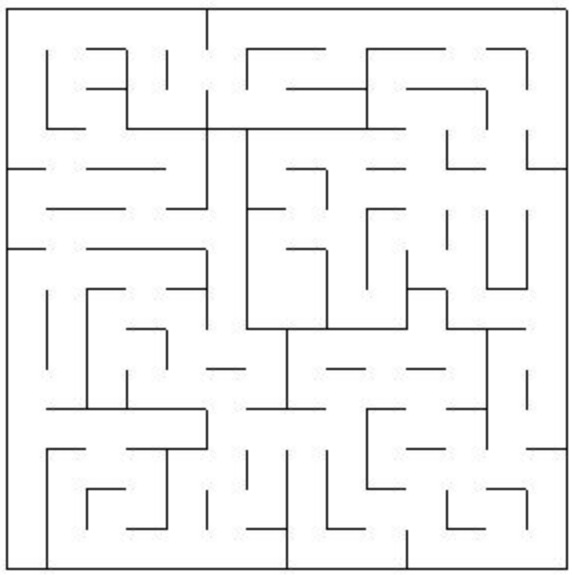
   2. The maze exists on an *n* x *n* grid of squares, *n* even. The minimum value of *n* is twelve, the maximum sixteen. Along the outside perimeter of the grid, and on the edges connecting some of the internal squares, are walls that block all movement. The robot will start in the square in the bottom- 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.

3. 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.



4. visualize test_maze_02.txt



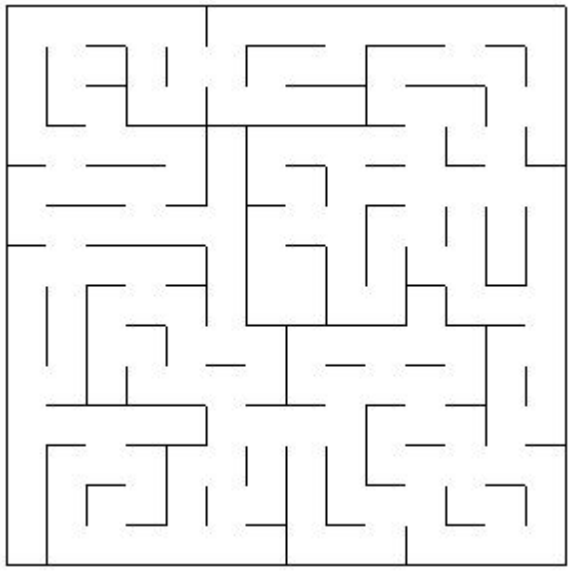
E. Robot specifications
    1. The robot can be considered to rest in the center of the square it is currently located in, and points in one of the cardinal directions of the

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.

2. More technically, at the start of a time step the robot will receive 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.

F. Scoring
    1. 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.
G. # most of context above copy from Plot and Navigate a Virtual Maze - Project Description

II. Data Exploration



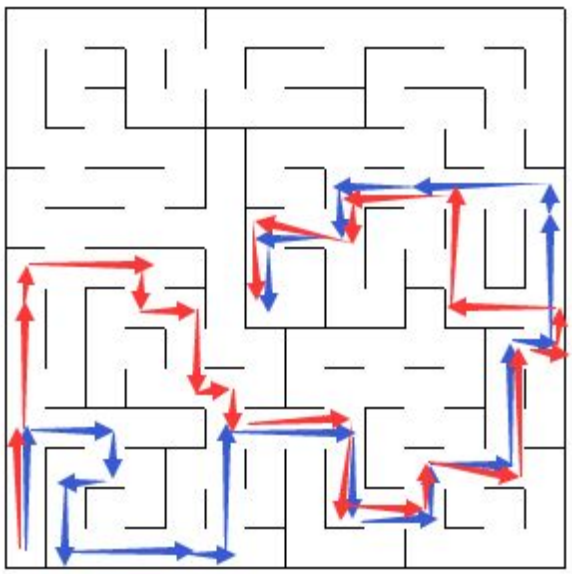
A.

B. The graph above is the visualization of maze 02.

C. Agent start at the bottom-left corner of maze, need to get to the center of maze as fast as possible.

D. For example,when agent in state [3,0,0], agent's sensors will return [0,4,4], means agent can't rotate counterclockwise and move forward, otherwise it will against wall and stay where it are; agent can move forward 4 cells or rotate clockwise then move forward 4 cells, but agent can move forwards or backwards up to 3 units, so furthest cells agent can get to are [6,0,0] and [3,3,90].

E. Because the lesser step agent take, the better score agent get, so it is wise to avoid hit walls or move backwards.



F. two optimal paths

III. Algorithms and Techniques

A. The algorithms I want to apply are DFS(Depth-First Search) and A*.

B. A* is a efficiently algorithm that widely used in pathfinding, it works similar to BFS(Breadth-First Search), but it won't search blindly like DFS or BFS, there are some heuristic functions direct it how to search(Notes that the heuristic function I use to estimates is L1 distance a.k.a Taxicab geometry).

C. The reason why I want to use DFS is the nature of A*.
   1. Because A*'s fundamental process almost as same as BFS, use a queue store open locations, pop open location out in a particular order but not continuous location.
   2. Agent can't teleport to target location but have to move forwards to target location step by step, that will wasts lots of steps.
   3. So I decide to ask agent explore and map the whole maze, and then use A* to compute optimal path from bottom-left corner to center.
   4. In particular, I will modify DFS to non-recursion version in order to adapt the maze system.

IV. Benchmark

A. I am going to assume DFS cost up to $3n^2$ (n is the size of maze, and 3 is an estimate base on my experience), so the score in first run should between [14,26].
B. Agent may need [20,30] steps take itself to center of maze.
C. So, when final score stay betwenn [34,56], it will be a reasonable result.

V. Data Preprocessing

A. This project don't need to run data preprocessing, because it doesn't need or own a large dataset like supervised learning or unsupervised learning.
B. The sensors data provided can use directly, there are well design.

VI. Implementation

A. Please check code in file robot.py directly.
B. Design non-recursion DFS for maze system must be the hardest part of this project. Because only maze system allow to call next_move function, so DFS can't code as recursion, and it have to be called when next_move called by maze system. So I have to design backward and tweak code to make things work. It is hard to say my code are elegant.

VII. result

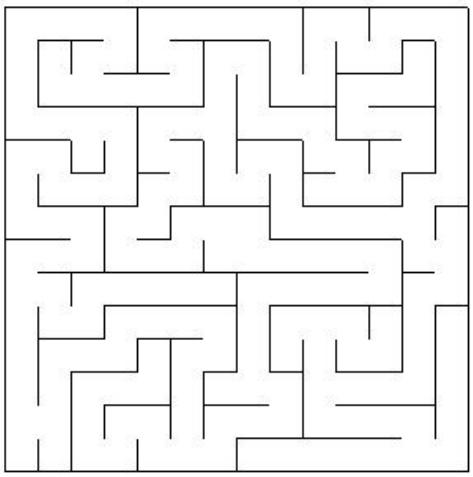
A. test_maze_01
    1. maze_size: 12
    2. cost in first run: 365
    3. cost in second run: 19
    4. final score: 31.167
    5. it is a reasonable result according to Benchmark section
B. test_maze_02
    1. maze_size: 14
    2. cost in first run: 509
    3. cost in second run: 23
    4. final score: 39.967
    5. it is a reasonable result too
C. test_maze_03
    1. maze_size: 16
    2. cost in first run: 643
    3. cost in second run: 25
    4. final score: 46.433
    5. still stay in reasonable range

VIII. Conclusion

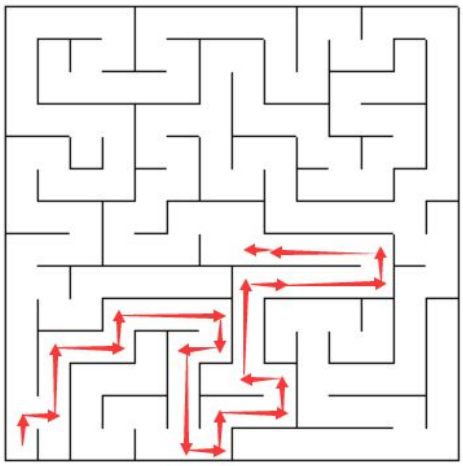
A. Free-From Visualization
    1. my_maze.txt

```
14
1,7,5,5,5,7,6,3,5,6,3,5,5,6
1,13,5,6,3,12,10,10,3,12,10,3,4,10
3,5,6,10,9,6,9,12,10,1,14,11,6,10
9,6,10,9,6,10,3,6,9,5,12,10,9,12
3,12,9,4,10,10,10,9,6,3,6,10,3,6
11,5,5,6,10,10,9,6,9,12,10,9,12,10
9,6,2,9,12,8,3,14,3,5,13,5,6,10
2,10,11,5,5,6,11,12,9,6,3,5,12,10
10,9,12,1,6,10,10,3,5,12,10,3,5,12
10,3,7,5,14,10,10,10,2,3,12,9,5,6
10,10,10,3,12,10,10,10,11,12,3,6,3,12
10,10,10,9,4,9,12,10,9,6,10,10,9,6
11,12,9,5,5,6,3,13,6,9,12,9,4,10
9,5,5,5,4,9,13,4,9,5,5,5,5,12
```

2. visualization



3. solution



4. result
    a) cost in first run: 506
    b) cost in second run: 20
    c) final score: 36.867
    d) reasonable result, this is what i am excepting
5. this maze have two paths can get agent from bottom-left to center, and agent learn to choose the optimal path rather than another much longer path

B.

1. At first, I want to solve this project with model-free reinforcement learning, in particular Q-learning, because I think this project very similar to [P4:Train a smartcab to drive] which can solve with Q-learning. Without much consideration, I begin to code. As process

move forward, I find out that maybe I am wrong, questions keep appearing and I can't solve them, such as how to determine reward? How to train agent in two run? How to design state?

2. Therefore, I stop coding and begin thinking. I still try to solve those question, or at least I still want to solve this project with reinforcement learning techniques, but not much process make.
3. Finally, I give in and try to find other solutions. After take Artificial Intelligence for Robotics (cs373), I think A* will play well in this project, because this whole project's aim is pathfinding.
4. So I start to work on A*, and figure out I should map the maze first, that how this project process.

C. *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 might be necessary to your robot's code to handle the added complexity? Are there types of mazes in the continuous domain that could not be solved in the discrete domain?*

1. If walls have thick and robot have bulk, I have to add some code to justify whether robot might stuck somewhere or not, and determine what action to take if get stuck.
2. There maybe exist some continuous version maze allow robot get through some important square but not in discrete version, then the discrete version maybe unsolvable.