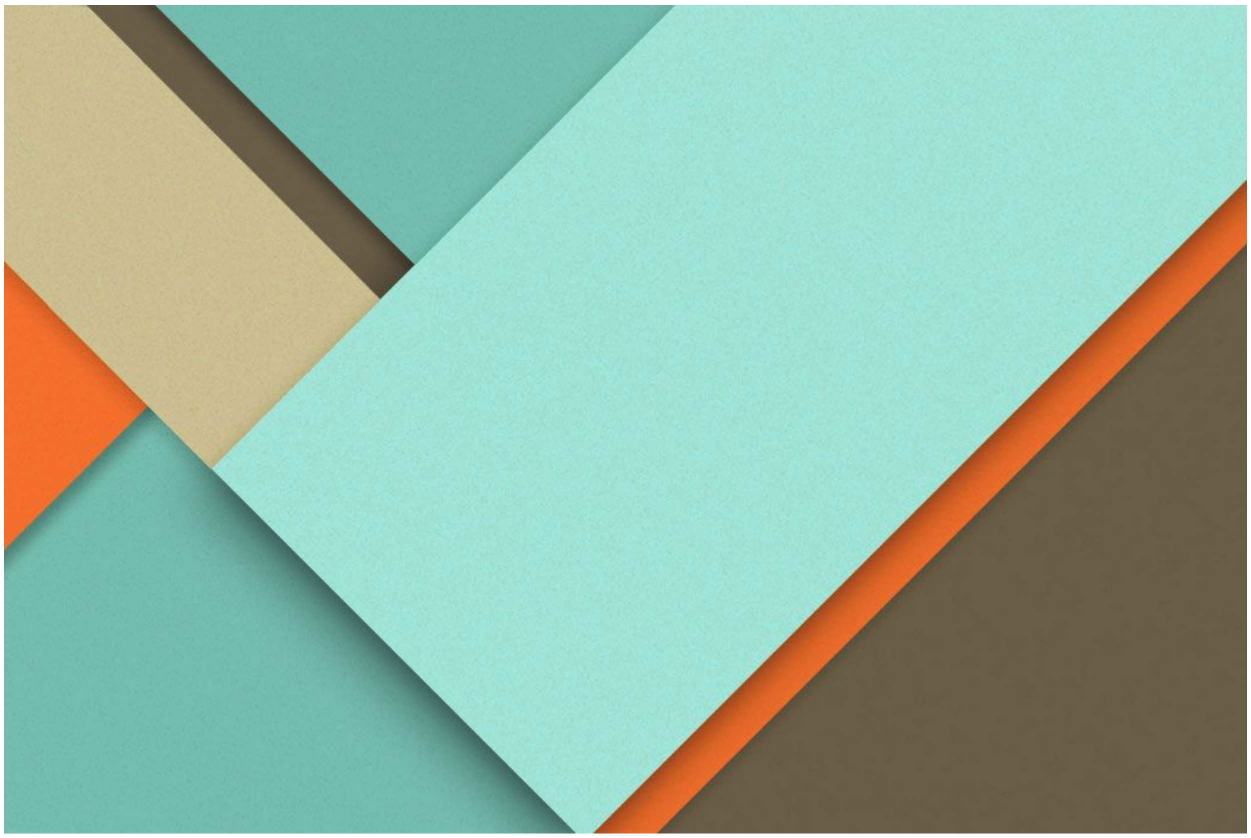


Robot motion Capstone Project

Plot and Navigate a Virtual Maze

04.17.2017

Olivia Natacha
Adipster

DEFINITION

Project Overview

This project in which we are working consists in programming a virtual robot to explore a maze from a predefined corner to its center, and finding the best way to reach this center. There are several path to reach the goal, among them of course, the robot should learn the shortest one. 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. The goal of this project is to define and implement a strategy to consistently discover an optimal path through a series of test mazes that exist within a virtual world inspired by the Micromouse problem.

Problem statement

The maze exists on a grid of n squares in column and n squares in rows (*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.
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 va76lue 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.

Metrics

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:

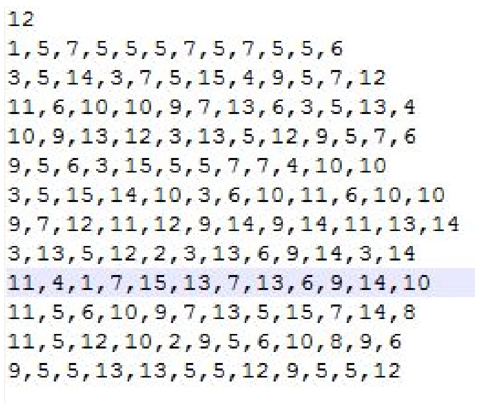
- the number of steps required to execute the second run,
- plus the number of time steps required to execute the first run divided by one thirtieth.

A maximum of 1000 time steps are allotted to complete both runs for a single maze. It is important to note that this scoring metric penalises both robots that fail to find the optimal path to the goal as well as those that explore the maze in an inefficient manner. That said, a robot that limits exploration and fails to find the optimal path to the goal is likely to, but may not necessarily, perform worse than a robot that takes the time to explore the maze sufficiently and finds the optimal path to the goal.

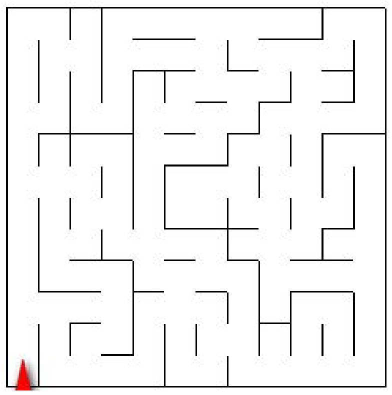
ANALYSIS

Data Exploration

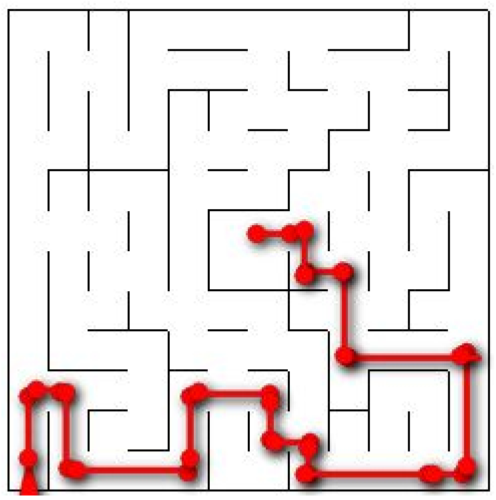
We are now going to discuss thoroughly certain features of the dataset. Our dataset is the maze provided to the system as a file. Here is the maze with represented in file and graphically by turtle python library:



```
12
1,5,7,5,5,5,7,5,7,5,5,6
3,5,14,3,7,5,15,4,9,5,7,12
11,6,10,10,9,7,13,6,3,5,13,4
10,9,13,12,3,13,5,12,9,5,7,6
9,5,6,3,15,5,5,7,7,4,10,10
3,5,15,14,10,3,6,10,11,6,10,10
9,7,12,11,12,9,14,9,14,11,13,14
3,13,5,12,2,3,13,6,9,14,3,14
11,4,1,7,15,13,7,13,6,9,14,10
11,5,6,10,9,7,13,5,15,7,14,8
11,5,12,10,2,9,5,6,10,8,9,6
9,5,5,13,13,5,5,12,9,5,5,12
```

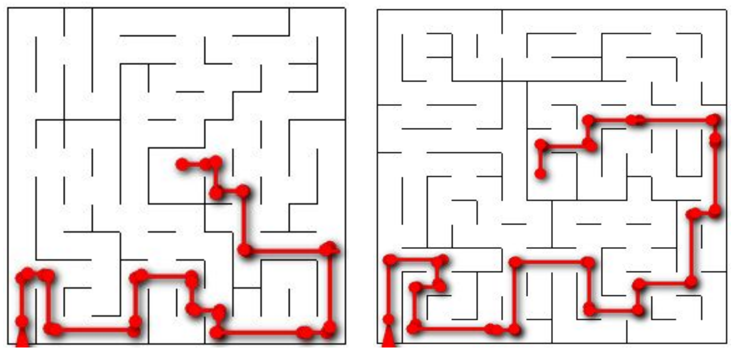


In the right image, the first row (12) in the maze dimension, each of the following rows represents a vertical wall of the maze, where the leftmost number describes the buttom buttom square. The robot is the red triangle. With its three sensors (left-front-right), he receives maze information in the form of a 3-tuple [x,y,z] where x, y and z are integer (< dimension and >=0) representing the number of open wall respectively in the left, front and right.



Here is the best path to the goal using 17 steps.

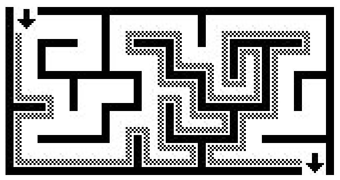
Exploratory visualization



Algorithms and techniques

There are many algorithms designed to solve the navigation of a maze using the optimal path. Among them we have: Random mouth algorithm, Wall follower algorithm, pledge algorithm, A* algorithm, minimax, alpha beta pruning, Dijkstra, The A*, Dijkstra's, Graph traversal algorithms. This problems will best be solved using reinforcement learning techniques. All this techniques work on a graph. This implies that the maze must first be modeled by a graph. In the following, we are going to explain what

- **Dijkstra's algorithm**: helps finding the shortest or optimal paths from a source vertex v to all other vertices in the graph. When the robot is running the maze for the first time (exploration), he builds a graph at the same time. After the graph is built, for the second run we can use Dijkstra's algorithm to compute the optimal path to the center.
- **Graph traversal algorithms**: Those are algorithms for exploring graphs starting at one of its vertices, i, and finding all vertices that are reachable from i. The maze can be modelized by a graph where vertices are squares. In the exploration phase, the robot goes from the start vertex *i0* and finds all vertices that reachable from i0 in order to *i0*. There exists two variants of this algorithms :
  - ***Depth-first-search*** visits the child vertices before visiting the sibling vertices. It means, it traverses the depth of particular vertices before exploring its breadth. This algorithm takes a Graph G and vertices v as inputs and returns a labeled graph G' from G.
  - ***Breadth-first search:*** The simplest of the graph search algorithm, it visits the neighbour vertices before visiting the child vertices. They often use it to find the shortest path from a vertex to another
- **Tree traversal algorithms**: Algorithms to access node of a tree. It is special case of graph traversal. A tree is a particular graph while a graph is not necessarily a tree.
- ***A* algorithm*** : is a modification of Dijkstra's Algorithm that is optimized for a single destination. Dijkstra's Algorithm can find paths to all locations; A* finds paths to one location. It prioritizes paths that seem to be leading closer to the goal. Dijkstra's Algorithm works well to find the shortest path, but it wastes time exploring in directions that aren't promising. Greedy Best First Search explores in promising directions but it may not find the shortest path. The A* algorithm uses *both* the actual distance from the start and the estimated distance to the goal.
- **Wall follower algorithm** :



The robot runs following the walls.

Benchmarc

As enumerated in the previous section, there are multiple algorithms, each with different properties and unique solutions. The performance criteria of this problem is based on the score provided by the algorithms that are used. What we call score of the robot is formulated as follows : ***score = n_step2 + n_step1/30***.
Our benchmark will be based on this formula. The parameter n_step2 is the number of time steps required by the robot to reach the goal. It depends on the optimal path that have been previously calculated by the robot in the first run. So for a particular maze, best path will always be the same whatever the number of time that the robot executes the second run.
The parameter n_step1 is the number of step to execute the first run. This parameter can vary according to the chosen algorithm. So it the keystone of our benchmark model.

METHODOLOGY

Data preprocessing

In this project, there is no data preprocessing needed. Robot gathers the data that it needs online by interacting with environment and sensing walls with its sensors. Once robot has accumulated sufficient data for its algorithm to work, it starts using it to solve the maze. Therefore there is no data preprocessing in this project.

Implementation

What we are going to implement with the help of algorithms and techniques discussed above, is to **store and update the robot knowledge of the maze**, to **determine how far the robot should move and in which direction to maximise knowledge of the maze while exploring yet minimise the number of steps taken during exploration**, to **find the goal location and make sure the Robot has visited it** (this is a requirement to allow the robot to end the exploration run early), to **determine if the path to the goal is optimal**, once the optimal path has been found, save it, end the exploration run, and **execute the second run**.

First, the file describing the maze is loaded and converted into a two dimensional matrix describing the permittivity of wall. This algorithm also have to avoid the robot running an unsolvable maze so it performs some consistency checks for wall positioning.

The first run consists of exploring the maze in order to build a graph up to the goal. After discovering the goal, the algorithm start to calculate costs off all possible path from the origin i to other nodes (including the goal), and outputs the best found path using Dijkstra's algorithm.

RESULTS

Model Evaluation and Conclusion

The model perfectly run with the provided test mazes as follows:

	Maze_test_01 (12x12)	Maze_test_02 (14x14)	Maze_test_03 (16x16)
score	20.667	28.567	31.767
steps	17	22	25

The evaluation of this model will be quite reasonable if we have to use other maze than those given for test. It means a way to verify whether the robot can generalize what is seen in the 3 test mazes to a new set of maze. So we have downloaded a set of various generated mazes with even size between 12 and 16.

	12x12 mazes	14x14 mazes
100 mazes. No results for some of them	18.433 10.267 16.400 5.200 9.733 16.000 16.067 17.167 13.767 9.167 8.867 13.667	6.300 23.700 10.333 19.433 8.500 14.633 10.600 13.433 9.600 8.767 14.667 19.533

	14.867 13.433 11.867 11.833 7.600 4.467 13.333 11.367 9.033 14.733 10.767 13.667 11.267 26.033 6.500 16.667 7.033 11.967 16.167 11.133 14.567 13.267 8.800 15.667 18.900 14.733 7.800 10.233 13.433 10.933 9.200 11.267 11.867 7.267 5.733 8.300 12.367 9.267 12.733 7.767 16.533 9.567 14.333 10.100 12.700 12.333 9.833	13.367 6.567 8.700 13.500 14.967 10.933 19.167 8.833 10.933 6.933 16.067 14.167 7.767 17.933 12.400 19.367 15.533 14.533 16.067 16.767 14.000 16.867 14.200 8.900 12.033 14.467 16.200 10.533 15.767 10.400 21.167 14.467 18.300 10.200 14.233 7.467 14.733 16.700 6.733 11.833 11.100 19.067 13.600 20.800 12.100 9.167 13.933

	14.133	14.833
	13.100	13.033
	9.233	8.467
	14.933	17.500
	13.500	15.333
	9.267	13.800
	15.767	12.233
	18.400	13.033
	12.733	13.733
	5.400	20.133
	6.600	16.200
	16.267	19.267
	17.000	11.167
	13.900	16.667
	15.800	12.700
	6.667	20.833
	11.600	9.967
	10.333	11.267
	7.900	9.667
	15.767	7.900
	10.300	15.833
	6.667	7.100
	11.400	12.533
	16.733	17.300
	11.933	8.367
	11.300	17.133
	16.833	14.967
	10.300	11.600
	14.633	8.633
	7.567	11.800
	15.200	14.100
	12.700	20.767
	14.100	9.467
	16.367	22.133
	12.633	8.667
	9.933	12.133
	12.600	9.900
	11.500	
	17.233	
	16.433	

For all those maze, the robot completed well, that means the model is robust.

Justification

However, we noticed that for each dimension, there were mazes in which the robot was doing a loop in the exploration step. This could be possible due to the nature of those particular mazes.

CONCLUSION

Free-Form Visualization

Here are the visualisation of the robot best found path running the test_maze_02:

(0, 3), (2, 3), (2, 2), (1, 2), (1, 0), (4, 0), (5, 0), (5, 3), (8, 3), (8, 1), (10, 1), (10, 2), (12, 2), (12, 5), (13, 5), (13, 8), (13, 9), (10, 9), (8, 9), (8, 8), (6, 8), (6, 7)

Maze 1 : (0, 2), (1, 2), (1, 0), (4, 0), (4, 2), (6, 2), (6, 1), (7, 1), (7, 0), (10, 0), (11, 0), (11, 3), (8, 3), (8, 5), (7, 5), (7, 6), (6, 6)

Maze 3 : (0, 3), (2, 3), (2, 4), (0, 4), (0, 7), (0, 10), (0, 13), (0, 15), (3, 15), (6, 15), (7, 15), (7, 12), (8, 12), (8, 11), (11, 11), (11, 10), (8, 10), (6, 10), (6, 8), (5, 8), (5, 7), (6, 7), (6, 5), (8, 5), (8, 7

Reflection

After realising this project, we noticed that the cost of the robot is most influenced by the first run where the robot has to explore the maze and to build a map. The algorithms we used has brought us to good results. But we ask ourselves whether there is no way to explore the maze in a better way than all those discussed above. If the process of exploring maze and calculating cost of each path could be improved, then the would be also be significantly improved (int means reduced).

Concerning maze where the robot loops before discovering that there are no ways, we ask ourselves how we can do to let him discover the loop earlier.

Improvement

All our improves concern the answer given the questions asked above.