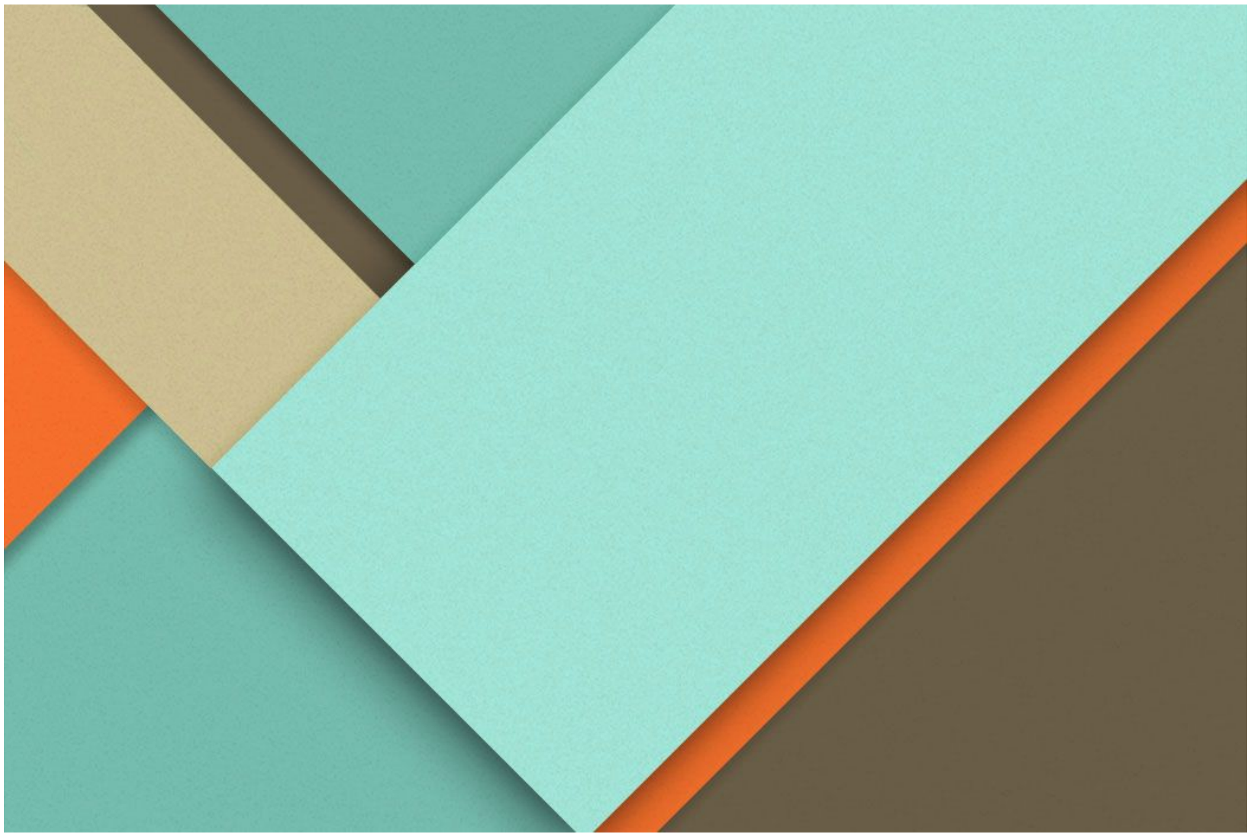


# Robot motion Capstone Project

**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:

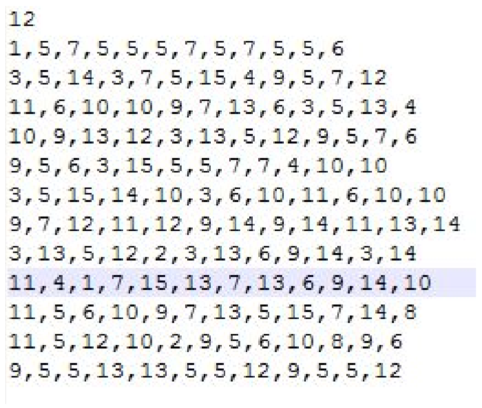
- the number of steps required to execute the second run,
- plus the number of time steps required to execute the first run divided by one thirtieth.

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.

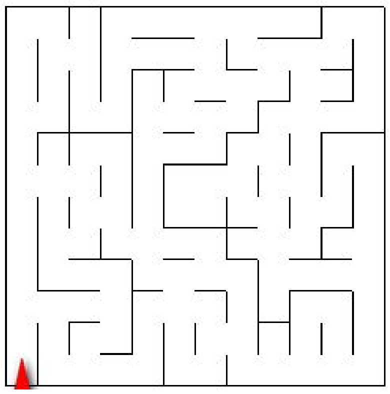
## ANALYSIS

## 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:

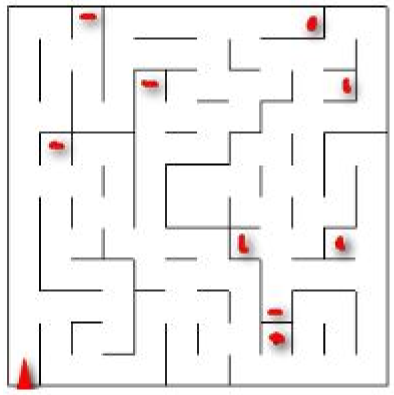


```
12
1,5,7,5,5,5,7,5,7,5,5,6
3,5,14,3,7,5,15,4,9,5,7,12
11,6,10,10,9,7,13,6,3,5,13,4
10,9,13,12,3,13,5,12,9,5,7,6
9,5,6,3,15,5,5,7,7,4,10,10
3,5,15,14,10,3,6,10,11,6,10,10
9,7,12,11,12,9,14,9,14,11,13,14
3,13,5,12,2,3,13,6,9,14,3,14
11,4,1,7,15,13,7,13,6,9,14,10
11,5,6,10,9,7,13,5,15,7,14,8
11,5,12,10,2,9,5,6,10,8,9,6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



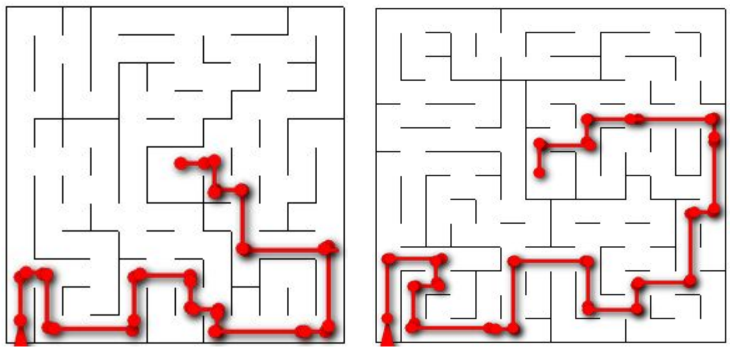
In the left image, the first row (12) in the maze dimension, each of the following rows represents a vertical wall of the maze, where the leftmost number describes the bottom square. The robot is the red triangle in the right image. With its three sensors (left-front-right), he receives maze's informations 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. The starting point of the robot is the point of coordinates (0, 0): especially the point where the red triangle is located in the above. The robot can encounter two major difficulties : the loop and the dead-end.

- In a loop, the robot will run through a same path many times until it exhausted its allotted time-step. This lead to ask ourselves if there is no way to prevent looping instead of looping before realizing there is loop.
- In the exploratory run of the robot, it enter all dead-end squares. Among dead-end squares. A dead-end square is one that only has one opened wall. So the robot sensor see this open wall, but cannot know if at least one of its adjacent walls are also open. The problem of those point is that the robot cannot detect them before having a whole knowledge of the maze. While, avoiding them would greatly improve the speed of the first run!!!



See the red point : They represent dead-end square. In this maze we have 10 (the robot current location is also counted). If the robot could avoid them it would reduce 10/30 = 1 / 3 = 0.33 in the score of the first run that is a remarque increase, and better if there are more than 10 dead-end.

## Exploratory visualization



Here are presented two mazes of respective dimensions 12x12 and 14x14. And the path in red is the best found path, resulting of the second run for each of them. Each edges of this path measures one step. So for the first maze we can see 17 steps while in the second we see 22 steps. The exploratory run has a great cost resulting of building all possible path from a square to another. This step ends with graph of the maze containing paths and their cost.

## Algorithms and techniques

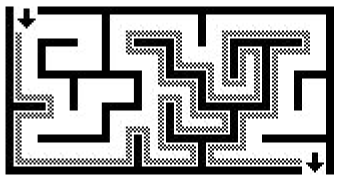
There are many algorithms designed to solve the navigation of a maze using the optimal path. Among them we have: Random mouth algorithm, Wall follower algorithm, pledge algorithm, A* algorithm, minimax, alpha beta pruning, Dijkstra, The A*, Dijkstra's, Graph traversal algorithms. This problems will best 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 i 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.

## Benchmark

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***.

The n_step1 value is the result of the first run. The benchmark model should place an attention to its value because it the one that really influences the score. A basic way of exploring the maze is going to lead the robot to visit each squares two times (in the worth case). That means n_step1 = n*n*2. For test maze 1, n_step1=12*12*2=288, when we divide by 30 we have 9.6.

The optimal exploration will be done by visiting each node 1 time at most. It means n_step1 = n*n and 144 for test maze 1. That give 144/30 = 4.8 score for the first run. while the best is 3.667. The second run score is 17 for this maze. Our solution will be acceptable if, for the maze one, the score is close to 17+4.8 = 21.8. It will not be a good solution if the score is close to 17+9.6=26.6.

## 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 an adjacency list describing the permittivity of wall. After it another algorithm is used to check whether the maze is solvable or not by checking inconsistencies between neighbour walls.

The first run consists of exploring the maze in order to build a graph up to the goal. The coordinates of the goal location are known for every given maze dimension, because they can be deducted knowing the goal is in the center of maze. This location consists of a square formed by four square where there is no inner wall and on open wall. In each square the robot is, he checks if coordinates machte with those of the center (goal location). For an nxn maze, the goal is located in coordinates :

- (dim/2 +1, dim / 2 + 1),
- (dim/2 + 2, dim / 2),
- (dim/2 + 2, dim / 2 - 1),
- (dim/2 + 1 , dim / 2 - 1),

- (dim/2 -1 , dim / 2 - 1),
- (dim/2 - 2, dim / 2 - 1),
- (dim/2 - 2 , dim / 2),
- (dim/2 - 1, dim / 2 + 1)

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. Among algorithm and techniques listed above, A* and Dijkstra's algorithm are the best to choose because they are specialized in finding the best path (that is also what we are searching for) while other will only work to lead to the goal whatever the coast.

It is real A* is an optimized version of Dijkstra's algorithm for single path. But our choice is based on the second besond of its simplicity in implementing it.

## Refinement

The refinement consist of improving our 1rst and 2nd run strategy algorithm. In our principle, the second run depend directly from the first. In fact the first run ends with a graph with path costs and the best found path. The second run simply goes through the best path provided to the robot. There are no possibilities to the robot to go wrong or to take another path in this run. So there is no refinement to bring to the second run.

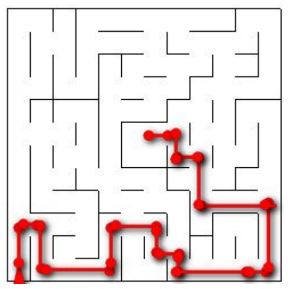
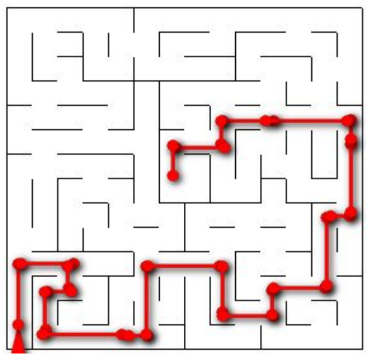
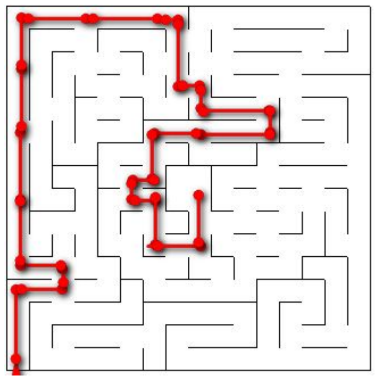
Concerning the first run, we find if there is a way to do it faster. Doing it faster can be possible by preventing robot to enter dead-end, and preventing it enter a loop. The difficulty for the first refinement option was that, there seems to be no way to do discover a dead-end without entering the concerned square (that is what we wanted to do).

## RESULTS

### Model Evaluation and Conclusion

Here are presented result of running test maze 1 2 and 3:

| | Maze_test_01 (12x12) | Maze_test_02 (14x14) | Maze_test_03 (16x16) |
|---|---|---|---|
| score | 22.600 | 30.567 | 35.667 |

| steps | 17 | 22 | 25 |
|---|---|---|---|
| |  |  |  |

| | Benchmak maze1 | Benchmark maze2 | Benchmark maze 3 |
|---|---|---|---|
| best | 21.8 | 28.53 | 33.53 |
| worst | 26.6 | 35.06 | 42.06 |

According to our benchmark model, the scores reached are acceptable because they are closer to best benchmark results..

## Justification

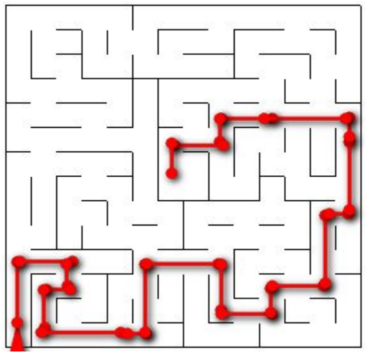
Those results are the best because compared to the benchmark, they reach optimal solution.

# CONCLUSION

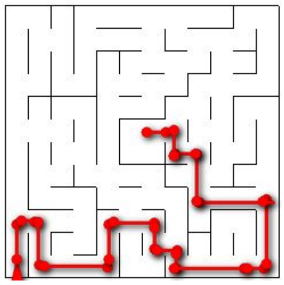
## Free-Form Visualization

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

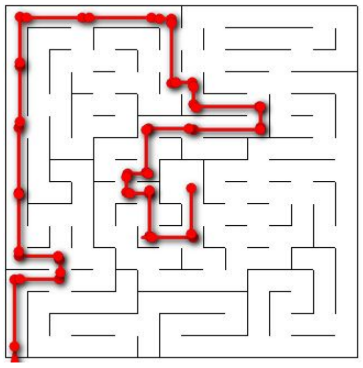
(0, 3), (2, 3), (2, 2), (1, 2), (1, 0), (4, 0), (5, 0), (5, 3), (8, 3), (8, 1), (10, 1), (10, 2), (12, 2), (12, 5), (13, 5), (13, 8), (13, 9), (10, 9), (8, 9), (8, 8), (6, 8), (6, 7)



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

While solving this problem, we have gone through a number of steps. The first step was to understand the problem's specifications and defining the metrics. The analysis step led us to model the problem with appropriate data structures among many possibilities. For the maze graph, we chose an adjacency matrix. The choice of the algorithm and techniques have not been easy. Especially deciding to use Dijkstra's algorithm rather than A* for the best path.  After 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).

## Improvement

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.