Nanodegree Capstone Project: Plot and Navigate a Virtual Maze project.

By: Mohlomi Pholoana

High-level Overview

The purpose of this project is to develop robot capable of solving a maze by finding the shortest path from start point to the center/ goal. Various path determining algorithms and techniques are used for the robot to navigate the maze and avoid dead ends with the intention of learning the paths that requires minimal moves or few moves to the goal. This project takes inspiration from Micromouse competition.

Description of the Input Data

As an input, we have 3 different mazes of varying sizes,illustrated below with diagrams, which are represented in a text format that are downloadable from the an archive in the resources. Each maze is a grid of cells that contain obstacle, open path or the goal. The robot starts at the bottom left of whichever maze and navigate the maze through to the goal. The complexity of mazes is intended to test the robot's ability to plan and adapt its paths efficiently. Key variables include the robot's starting position, obstacles, and the goal location which are all critical for determining the robot's movement and strategy.

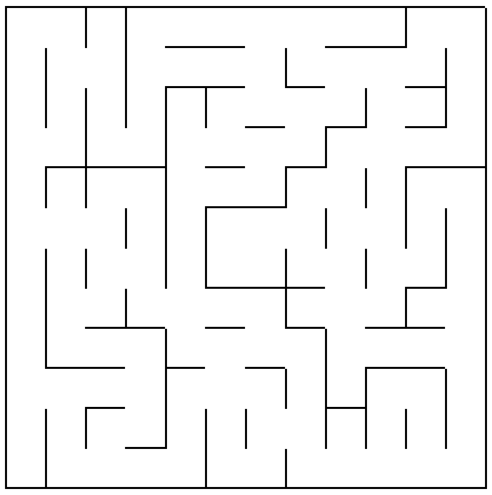


Fig 1. test_maze_01.txt

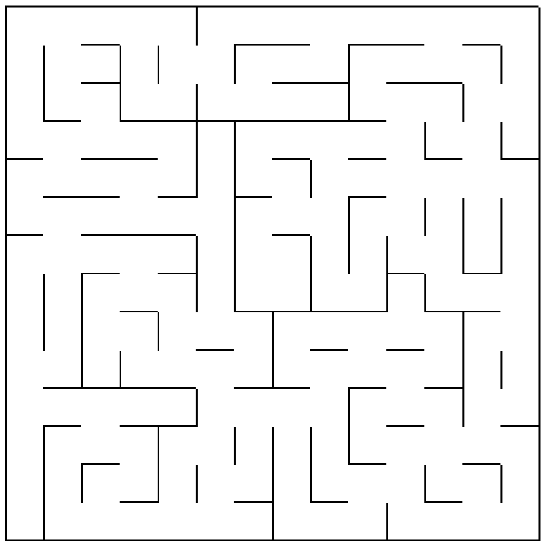


Fig 2. test_maze_02.txt

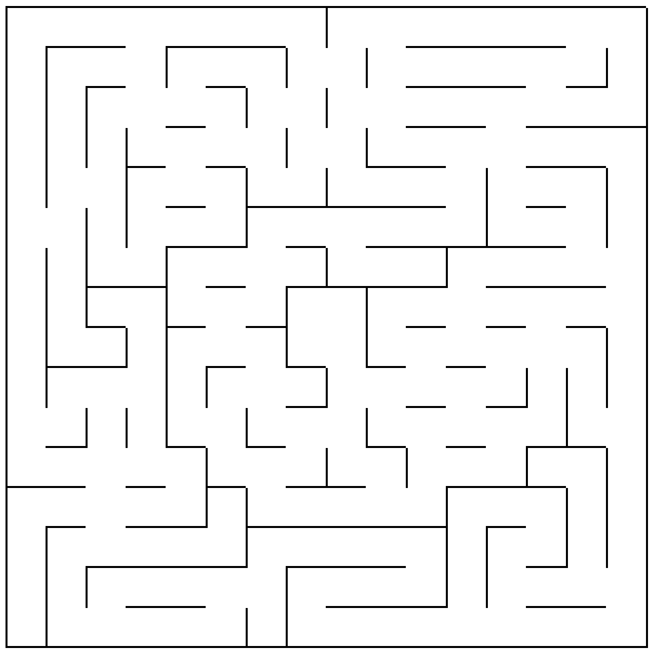


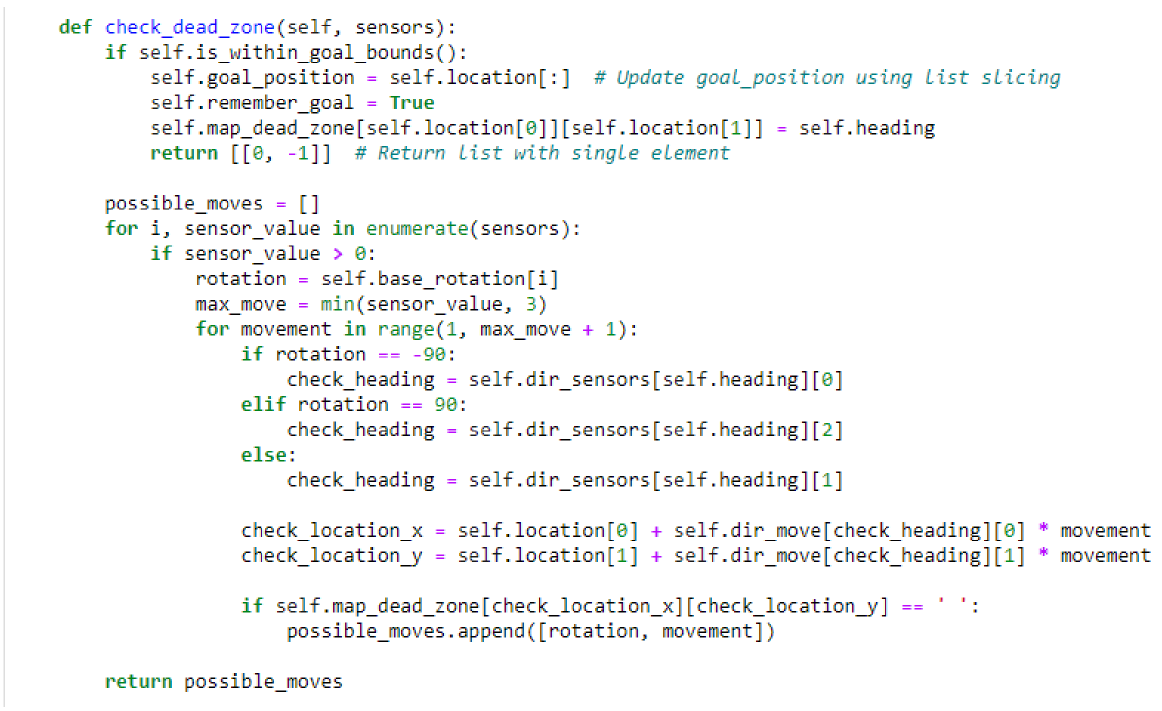
Fig 3. test_maze_03.txt

Strategy for Solving the Problem

The overall approach involves implementing and comparing different pathfinding algorithms including heuristic exploration, Breadth-First search (BFS), Dijkstra’s algorithm and A*. The rationale behind selecting these algorithms is to explore various strategies that makes the robot find the shortest path accurately and with less moves done, and the heuristic exploration was found to be the most efficient and scalable for this project.

Discussion of the Expected Solution

The proposed solution involves a combination of heuristic exploration and learning from previous paths explored to improve efficiency. So, the robot’s navigation is guided by the sensors that detect obstacles and available paths through which the robot evaluates possible moves, prioritizing open paths and then updates its strategy based on the paths it has already explored. The architecture of the robot environment is made up of 3 scripts – tester.py, maze.py and robot.py. The tester script is run with either of the three-test maze that were provided and as the tester runs it gets maze dimension from maze.py and then invokes and initialize the robot – robot.py giving it the maze dimensions to use. The robot through robot.py will make use of exploration model to determine next movement and rotation () while also avoiding the dead ends (check_dead_zone snippet).



```
def check_dead_zone(self, sensors):
    if self.is_within_goal_bounds():
        self.goal_position = self.location[:]  # Update goal_position using list slicing
        self.remember_goal = True
        self.map_dead_zone[self.location[0]][self.location[1]] = self.heading
        return [[0, -1]]  # Return list with single element

    possible_moves = []
    for i, sensor_value in enumerate(sensors):
        if sensor_value > 0:
            rotation = self.base_rotation[i]
            max_move = min(sensor_value, 3)
            for movement in range(1, max_move + 1):
                if rotation == -90:
                    check_heading = self.dir_sensors[self.heading][0]
                elif rotation == 90:
                    check_heading = self.dir_sensors[self.heading][2]
                else:
                    check_heading = self.dir_sensors[self.heading][1]

                check_location_x = self.location[0] + self.dir_move[check_heading][0] * movement
                check_location_y = self.location[1] + self.dir_move[check_heading][1] * movement

                if self.map_dead_zone[check_location_x][check_location_y] == ' ':
                    possible_moves.append([rotation, movement])

    return possible_moves
```

Metrics with Justification

The primary metrics used to access the performance of the solution include:

- ➢ Completion time – which evaluates how the robot can solve the maze.
- ➢ Number of moves – which measures the efficiency of the pathfinding algorithm in terms of the number of steps taken to reach the goal.
- ➢ Path length – which, lastly assesses the effectiveness of the algorithm in finding the shortest path towards the goal.

The above metrics are selected because they directly relate to the project's objectives to efficiently navigate the maze.

Exploratory Data Analysis (EDA)

An exploratory analysis was conducted to understand the structure and complexity of the mazes. The maze files are in text format, with the first line of each file containing a number, n, which describes the number of squares on each dimension of the maze. For example, as shown in figure 5 a 16x16 maze, n=16. Each number represents a four-bit coding that has a bit value of 0 if an edge is closed and 1 if an edge is open; the 1s register corresponds with the upwards facing side, 2s register the right side, 4s register the bottom side and 8s register the left side. That is, 9 means that the square is open on the left and top side (1*1+0*2+0*4+1*8 = 9), see figure 4 below.

Key findings from the EDA include:

- Distribution obstacles: the analysis revealed patterns in how obstacles are placed on different mazes.

- Path complexity: the decision to choose which path finding algorithm to use was made based on which algorithm performed better for the robot to locate the goal.

The robot has three sensors on it that detect the number of open squares in their respective directions. Based on sensor inputs, the robot makes rotation towards open squares, which defines its movement/ direction dictionaries as illustrated in figure 6. The robot as it navigates the maze, needs to avoid the obstacles which are costly to its run time and limited number of moves. This was another challenge I had to bear in mind as I was making changes to the next move function.

The EDA provided valuable insights that influenced the choice of algorithm to use, the self-exploration model = heuristic. Through tests and trials on 3 different mazes, the robot's navigation efficiently improved.

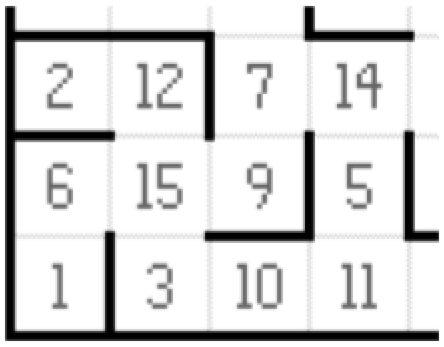
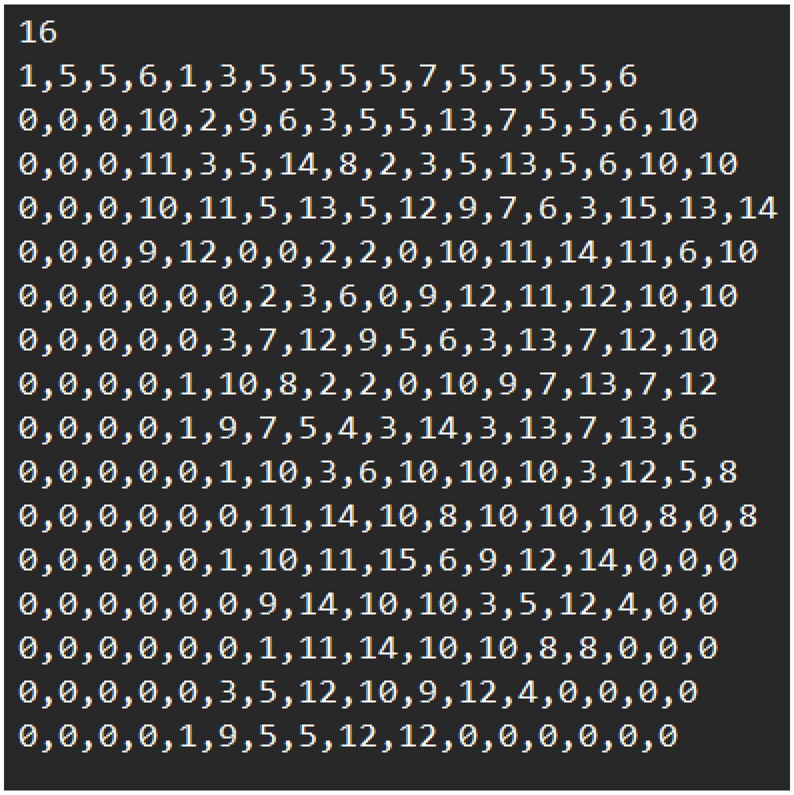
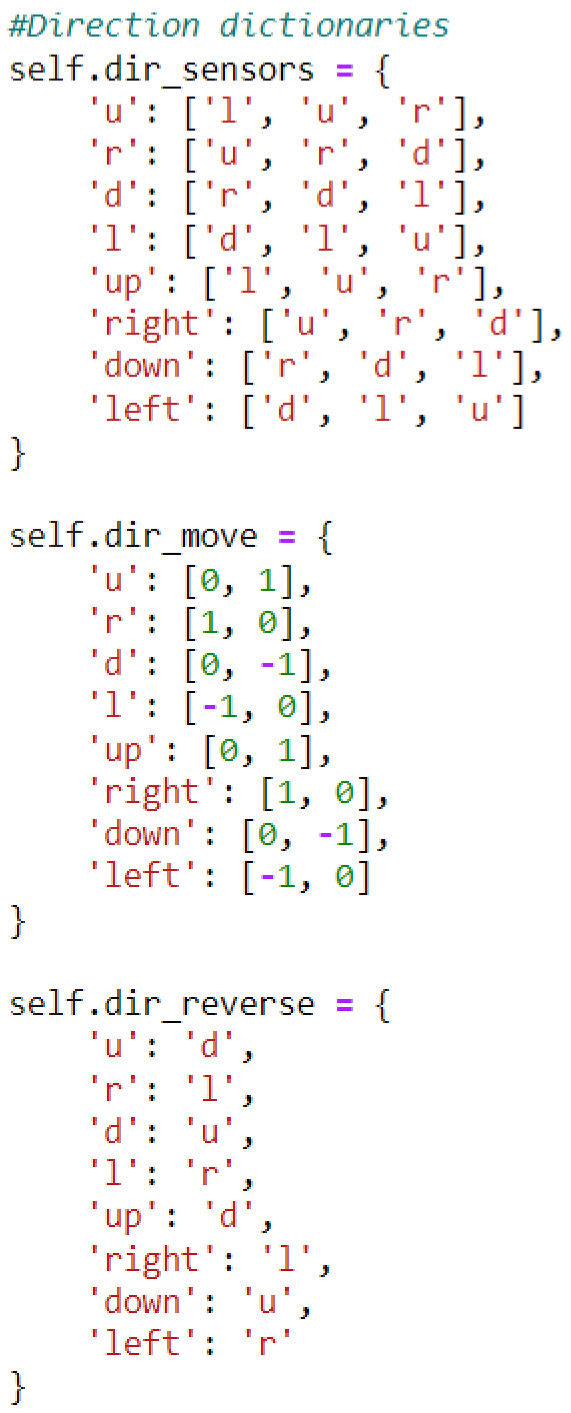


Fig 4. A four-bit coding maze



```
16
1,5,5,6,1,3,5,5,5,5,7,5,5,5,5,6
0,0,0,10,2,9,6,3,5,5,13,7,5,5,6,10
0,0,0,11,3,5,14,8,2,3,5,13,5,6,10,10
0,0,0,10,11,5,13,5,12,9,7,6,3,15,13,14
0,0,0,9,12,0,0,2,2,0,10,11,14,11,6,10
0,0,0,0,0,0,2,3,6,0,9,12,11,12,10,10
0,0,0,0,0,3,7,12,9,5,6,3,13,7,12,10
0,0,0,0,1,10,8,2,2,0,10,9,7,13,7,12
0,0,0,0,1,9,7,5,4,3,14,3,13,7,13,6
0,0,0,0,0,1,10,3,6,10,10,10,3,12,5,8
0,0,0,0,0,0,11,14,10,8,10,10,10,8,0,8
0,0,0,0,0,1,10,11,15,6,9,12,14,0,0,0
0,0,0,0,0,0,9,14,10,10,3,5,12,4,0,0
0,0,0,0,0,0,1,11,14,10,10,8,8,0,0,0
0,0,0,0,0,3,5,12,10,9,12,4,0,0,0,0
0,0,0,0,1,9,5,5,12,12,0,0,0,0,0,0
```

Fig 5. test_maze_02.txt



```python
#Direction dictionaries
self.dir_sensors = {
    'u': ['l', 'u', 'r'],
    'r': ['u', 'r', 'd'],
    'd': ['r', 'd', 'l'],
    'l': ['d', 'l', 'u'],
    'up': ['l', 'u', 'r'],
    'right': ['u', 'r', 'd'],
    'down': ['r', 'd', 'l'],
    'left': ['d', 'l', 'u']
}

self.dir_move = {
    'u': [0, 1],
    'r': [1, 0],
    'd': [0, -1],
    'l': [-1, 0],
    'up': [0, 1],
    'right': [1, 0],
    'down': [0, -1],
    'left': [-1, 0]
}

self.dir_reverse = {
    'u': 'd',
    'r': 'l',
    'd': 'u',
    'l': 'r',
    'up': 'd',
    'right': 'l',
    'down': 'u',
    'left': 'r'
}
```

Fig 6. Direction dictionaries

Benchmark

My expectation was because the max number of moves/ runs had been put as 1000 and considering the fact that this was my first challenge Project, I had anticipated to get "Allotted time exceed" on the first run and I will have to update and try different algorithms and fine tuning until now the goal was found at 900 runs.

Data Preprocessing

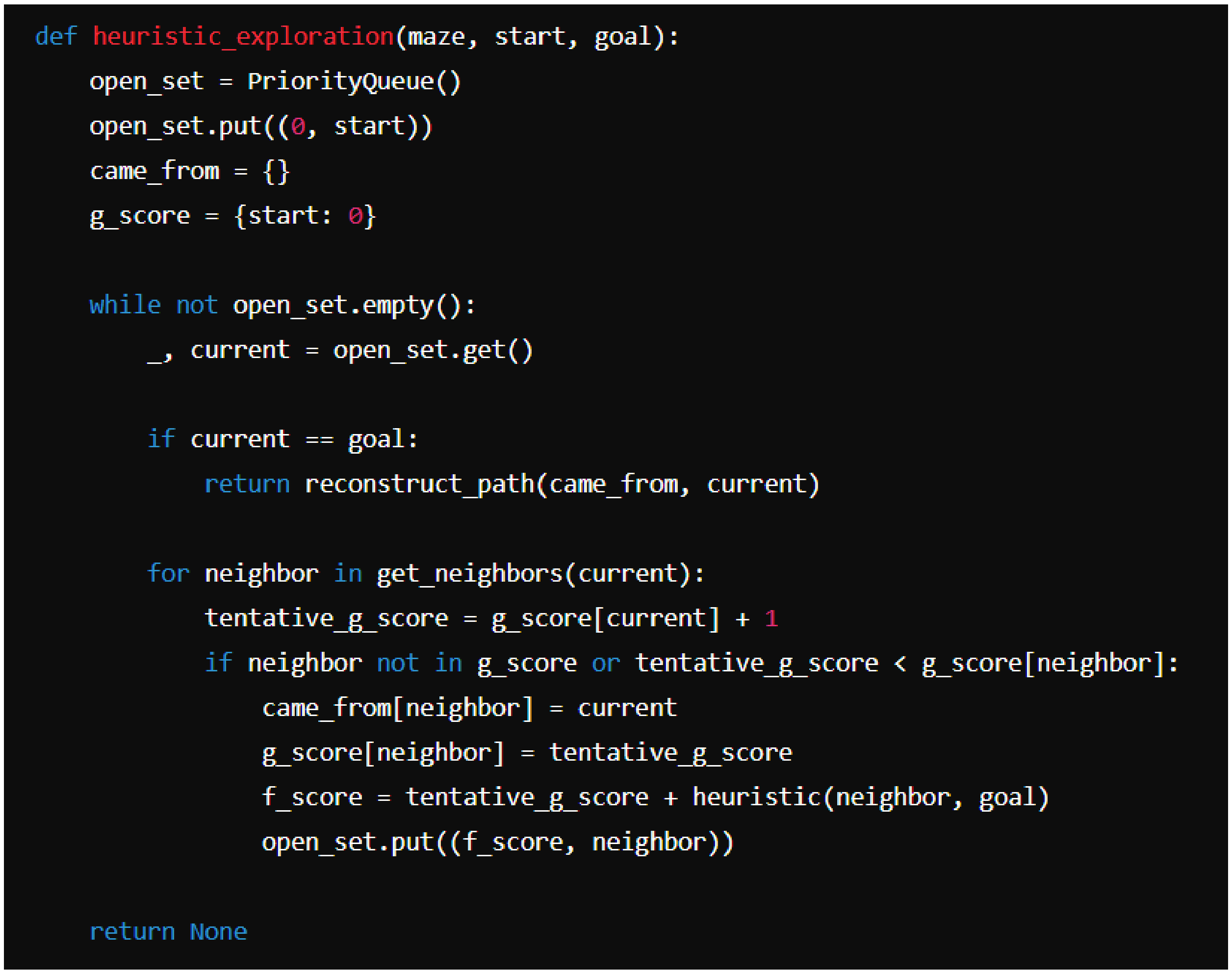
No data preprocessing was required in this case since the sensor specification and environment design were already done and shared to be used as they were. The only changes I had to do was to update few elements of tester.py and maze.py which were in python 2.7 so I had to update syntax such that it aligns with python 3.5.

Modeling

I experimented with several models:

- Heuristic exploration algorithm – prioritizes paths based on estimated distance to goal, leading to faster solution.
- Breadth-First Search (BFS) – explores all possible paths systematically and its very slow due to exhaustive search.
- Dijkstra’s algorithm – finds the shortest path to the goal but was computationally intense. If I was running the project on a server or hosted on cloud, then I would have considered.
- A* algorithm – combines the benefits of Dijkstra’s and heuristic search. A* was however slower in practice in my case, only worked better when I had max_time = 10

Below is a pseudocode snipped for Heuristic exploration algorithm:



```python
def heuristic_exploration(maze, start, goal):
    open_set = PriorityQueue()
    open_set.put((0, start))
    came_from = {}
    g_score = {start: 0}

    while not open_set.empty():
        _, current = open_set.get()

        if current == goal:
            return reconstruct_path(came_from, current)

        for neighbor in get_neighbors(current):
            tentative_g_score = g_score[current] + 1
            if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score = tentative_g_score + heuristic(neighbor, goal)
                open_set.put((f_score, neighbor))

    return None
```

Hyperparameter Tuning

This involves adjusting parameters such as maximum number of moves allowed, dead end function and counter function. The maximum number of moves allowed was adjusted from 900 to 10 000 to 100 to 10 while observing the performance and speed of the robot to find the goal and in how many moves. The dead end and counter function values were also altered while observing the performance.

Results

The heuristic exploration algorithm demonstrated the best performance, solving the mazes in significantly fewer moves and less time compared to other algorithms. Below are the results for a 16x16 maze (test_maze02.txt):

- Heuristic exploration – 25 moves in 10 seconds
- A* - Allotted time exceeded.

- BFS – 300 moves in 140 seconds.

Comparison Table

Algorithm	**Moves**	**Time (Seconds)**	**Path length**
Heuristic exploration	25	10	shortest
BFS	300	140	varying
A*	N/A	N/A	N/A

Conclusion

The project successfully developed a robot capable of solving different mazes using an efficient exploration model -heuristic. The project demonstrates the potential use of heuristic methods in robot's navigation and sets the stage for further enhancements.

Improvement

Future improvements could include:

- ✓ Spend more time on hyperparameter tuning and explore other parameters that would improve the results.
- ✓ Explore other models in depth with and without max time and number of moves allowed included.
- ✓ Have more test mazes with varying dimensions to run and test robot's navigation on in order to improve learning and improve efficiency.
- ✓ Spend more time on EDA so as to get more insights and realize some patterns.

References Acknowledgement:

A special thanks to the Udacity team, the reviewers/ accessors and facilitator for support and feedback while developing this project. A special thank you to my colleagues that we were undertaking the nano degree program together with for their support, our learning and development market SPOC, the group data science team – upskilling team.

1. pdf-libre.pdf (d1wqtxts1xzle7.cloudfront.net) (Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game vol 1, by Silvester Dian Handy Permana1 , Ketut Bayu Yogha Bintoro2 , Budi Arifitama3 , Ade Syahputra4)

2. ml-nanodegree-capstone/Plot and Navigate a Virtual Maze.pdf at master · eminnett/ml-nanodegree-capstone (github.com) (6th June 2024)

3. MLND-Capstone-Project-Plot-and-Navigate-a-Virtual-Maze/report.pdf at master · yychiang/MLND-Capstone-Project-Plot-and-Navigate-a-Virtual-Maze (github.com) (12 June 2024)