SEPTEMBER 25, 2018

ROBOT MOTION PLANNING CAPSTONE

PLOT AND NAVIGATE A VIRTUAL MAZE

JAMES TOOLES III

Table of Contents

Capstone Proposal 2

I. Definition 2

Project Overview 2

Problem Statement 2

Metrics 3

II. Analysis 4

Data Exploration 4

Exploratory Visualization 5

Algorithms and Techniques 7

Benchmark 8

III. Methodology 9

Data Preprocessing 9

Implementation 9

Refinement 13

IV. Results 14

Model Evaluation and Validation 14

Justification 17

V. Conclusion 18

Free-Form Visualization 18

Reflection 18

Improvement 18

References 19

Robot Motion Planning Capstone

Plot and Navigate a Virtual Maze

Capstone Proposal

James Tooles
September 25, 2018

I. Definition

Project Overview

This project's core outline[1] was created by Udacity and the idea for this project is mainly derived from Micromouse[2] competitions. In the competitions, a robot mouse agent is given the task of planning and plotting the best path from a corner of the environment (a maze) to its center. The purpose of the competition was for the agent to discover the environment during the first run and to use the best planned path to reach the center of the environment in subsequent runs. The goal of this project is to emulate these constraints of the completion and achieve the goal of the agent obtaining the fastest times possible in a series of test environments. The competition places the same restrictions on the agent as this project's restrictions, which will be covered in the following sections.

Problem Statement

This project is tasked with solving the problem of a robot mouse agent plotting a path from the bottom left corner of a maze to its center. The agent can embark on two runs of any given maze. The first run of the maze, the agent will explore, map, and analyze the environment to determine the best path plans to reach the center of the maze. The second run of the same maze, the agent will attempt to navigate the environment as quickly and efficiently as possible to reach the center of the maze.

Metrics

The agent will explore multiple environments and must complete two runs on each. During the first run of the environment, the agent will be allowed to freely explore to build a map of the environment. At someone point during the exploration, the agent must enter the goal area but, the agent is free to continue exploring the environment after entering the goal area.

The second run of the environment, the agent will be returned to the starting position and orientation. The agent's goal is to then navigate to the goal area in fastest time possible, minimizing actions (time steps) taken by the agent.

The agent's score is the number of time steps required to execute the second run, plus one thirtieth the number of actions taken during the first run. The maximum actions allowed for a completion of both runs is one thousand actions for both runs for a single environment.

$$\text{Score} = \#\text{actions}_2 + (1/13) * \#\text{actions}_1$$

where lower Score is better

II. Analysis

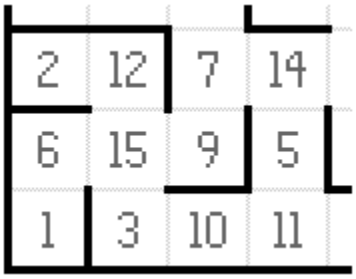
Data Exploration

The dataset and inputs for this project consists of the environment, the maze. The maze is an environment of an n x n grid of squares, where n is even. The environment can have an n value range of twelve to sixteen. The state space for this problem is equal to n * n * 4, where n is the size of the environment and 4 is the number of actions the agent can take in any given location within the environment. The state space is between, 576 and 1024, for environment sizes of 12 x 12 and 16 x 16 respectively.

The environments are accessed through text files. The first line of the text file is a number which describes the number of n squares within the environment. The first data row in the text file is the leftmost column of the environment, with the first element being the bottom-left corner of the environment. The remaining n lines will be n comma-delimited numbers which describe the passible (open) edges of the environment. Each number represents a four-bit number that has a bit value of 0 if an edge is impassible (closed) and 1 if an edge is passible (open). The 2^0 is the upward side, 2^1 is the right side, 2^2 is the bottom side, 2^3 is the left side.

Ex. 1010 $\Leftrightarrow$ (2^3 + 0 + 2^1 + 0) $\Leftrightarrow$ (8 + 0 + 2 + 0) $\Leftrightarrow$ 10

{Left: Open, Bottom: Closed, Right: Open, Upward: Closed}

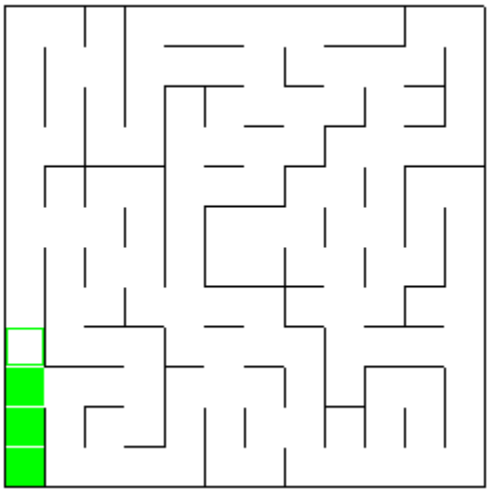


The environment will always have impassible walls on the outer board of the grid, which blocks all movement. Walls will be placed throughout the environment to block movement of the agent, creating a maze. The robot agent will always start in the bottom-left corner square of the environment grid, and be oriented upwards. The starting square will always have walls on the left, right, and bottom, which will allow for the first action to always be upwards. The environment will have a center within the environment grid consisting of a 2 x 2 square, this is the goal for the agent to position itself within the center 2 x 2 square.

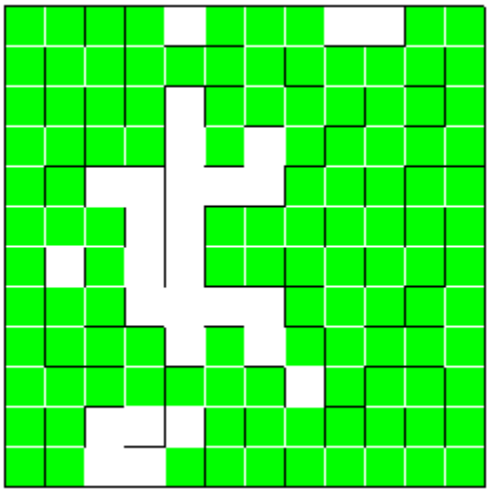
Exploratory Visualization

This section will detail visualizations produced during training and testing of the robot agent through the maze.

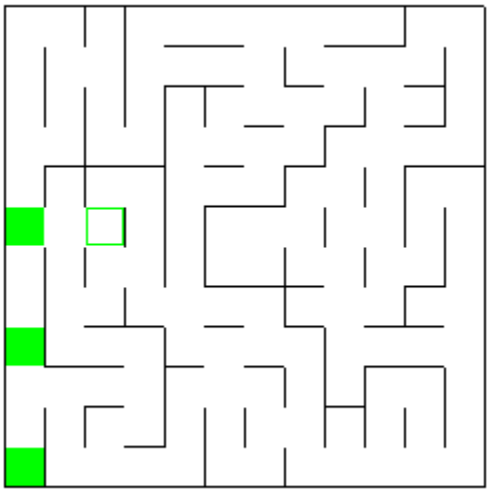
The start of the exploration robot agent run, the agent begins by exploring the legal actions available for the given maze cell. While the agent is exploring, random actions are taken to further explore the environment scholastically, decreasing the probability of a random action being taken over the course of the exploration run, run 1. Below displays this behavior.



Extensive exploration is also done in this run of the agent exploring the environment, the maze. The number of actions taken by the agent is predetermined within the code and can be changed by changing a simple variable. The agent then continues exploring until the follow requirements are met, the minimum amount of actions must be taken, and the goal must be visited. Below displays this behavior.



Once the conditions are met, the agent will then move into a training period, in which it develops a path planning model. Once the model is created, run 2 is started. During this run, the agent takes what it determined was the shortest path to the goal. The visualization below is the agent starting the second run of the maze, navigating the maze based on the training and machine learning the agent model performed during the first run and subsequent training epochs.



Algorithms and Techniques

The agent is given the problem of solving a randomized maze environment in the least amount of actions. The agent is given access to the environment dimensions initially, then allowed to explore the environment in the first run. The second run, the agent should utilize the optimal path to take the minimum actions necessary to reach the goal. Thus, receiving the best (lowest) score possible during the second run. The following theoretical high-level workflow will be taken:

1. Dimension of environment will be taken in on the initialization of the agent
2. The goal location area will be derived from the environment dimension
3. First run of the environment:
    a. The agent will explore the environment
        i. This could be done with random actions, actions that prefer to move towards the middle until encountering a wall, or another means of selecting actions
    b. The agent will record the environment during exploration, i.e. where there are walls

    c. The agent will not attempt to run into walls discovered
    d. The agent will reach the goal
        i. The agent can stop exploration
        ii. The agent can continue exploration until the environment is completely discovered
4. The path planning algorithm will take in the recorded environment data and decide the optimal path for the agent
5. Second run of the environment:
    a. The agent will follow the path planned by the algorithm
    b. The agent will stop once the goal is reached

Benchmark

The Benchmark Model for the agent directly relates to how the evaluation metrics are done for scoring the agent's path planning regarding steps taken within the environment. The agent is restricted to a total of one thousand time steps total for both runs within the environment. One baseline model would be that of taking five hundred actions for each run, maximizing the allowed one thousand time steps. These actions could be random actions to create a true baseline model, which would be random guessing path planning. If the solution for the agent does better than this baseline model, it would be a successful model. The following score would result for the baseline model:

$$Score = \#actions_2 + (1/13) * \#actions_1$$

$$Score_{Baseline} = maze\_dim + (1/13) * 700$$

$$Score_{Baseline} = maze\_dim + 53.85$$

III. Methodology

Data Preprocessing

This project did not require data preprocessing, as part of the problem is the robot agent being exposed to the “data” environment for the first time, every time.

Implementation

The logic of the robot agent is handled exclusively within the robot.py file. The Robot class first pulls in the maze dimensions when initialized within tester.py. During the initialization class attributes are created that are later used within the Robot class.

```
class Robot(object):
    def __init__(self, maze_dim):
        """
        Use the initialization function to set up attributes that your robot
        will use to learn and navigate the maze. Some initial attributes are
        provided based on common information, including the size of the maze
        the robot is placed in.
        """
        self.heading = 'up'
        self.maze_dim = maze_dim
        self.location = [maze_dim - 1, 0]
        self.goal_area = [self.maze_dim/2 - 1, self.maze_dim/2]
        self.dir_grid = [[0 for row in range(0, self.maze_dim)] for col in range(0, self.maze_dim)]
        self.count_grid = [[0 for row in range(1, self.maze_dim + 1)] for col in range(1, self.maze_dim + 1)]
        self.action_grid = [[0 for row in range(1, self.maze_dim + 1)] for col in range(1, self.maze_dim + 1)]
        self.model = [[0 for row in range(0, self.maze_dim)] for col in range(0, self.maze_dim)]
        self.cell_count = 0
        self.action_count = 0
        self.goal_success = False
        self.training = False
        random.seed(0)
```

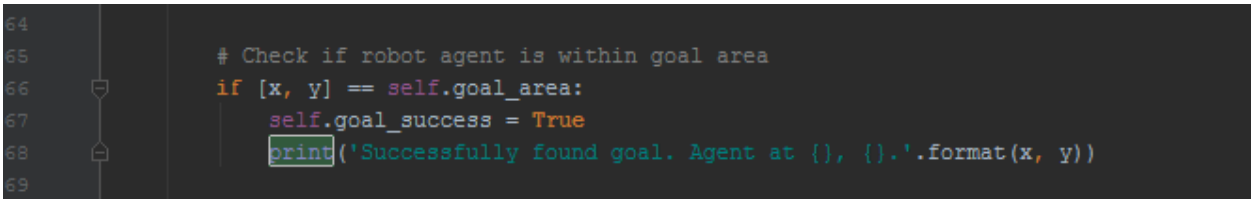
Next, the agent begins run 1. At each maze cell location sensor values are read into the update() Robot class function. These sensor values are given from tester.py, and consist of a list of integers which informs the agent of the left, forward, and right wall Booleans. These sensor values are recorded when the map_cell() Robot class function is called.

```
40      def map_cell(self, sensors):
41          """
42          Records the directional values for each given cell based on the sensor
43          input data.
44
45          :param sensors: the sensor values of agent for a given cell
46              (a list of ints, i.e. [0, 0, 1])
47
48          :return: NULL
49          """
50          x, y = self.location
51          headings = ['left', 'up', 'right', 'down']
52          directions = [8, 1, 2, 4]
53
54          if self.dir_grid[x][y] == 0:
55              for i in range(len(headings)):
56                  if self.heading == headings[i]:
57                      self.dir_grid[x][y] += directions[(i + 2) % 4]
58                      if sensors[0] > 0:
59                          self.dir_grid[x][y] += directions[i - 1]
60                      if sensors[1] > 0:
61                          self.dir_grid[x][y] += directions[i]
62                      if sensors[2] > 0:
63                          self.dir_grid[x][y] += directions[(i + 1) % 4]
64
65          self.dir_grid[self.maze_dim - 1][0] = 1
66
```

After the cell is mapped for walls given the sensor values, the Robot class function breadcrumb() is called. This function records the agent's unique visited cells.

```
66
67      def breadcrumb(self):
68          """
69          Counts the number of unique cells visited.
70
71          :param: NULL
72
73          :return: NULL
74          """
75          x, y = self.location
76
77          if self.count_grid[x][y] == 0:
78              self.count_grid[x][y] = 1
79              self.cell_count += 1
80
```

The code then checks if the agent is currently in a goal cell within the environment. If the goal has been reached, that is recorded as a Boolean value and stored as a Robot class attribute.



During the first run, the agent will continue visiting cells until the pre-determined number of maximum actions has occurred. The Robot class function make_model() will handle the agent action[make_model()], motion planning[make_action_grid()], and training[update_model()].

```
    def make_model(self):
        """
        Creates ML model for agent based on the first training run recorded
        sensor data and cell information

        :param: NULL

        :return: NULL
        """
        opened = []
        tune = 1
        trans = [[-1, 0], [0, 1], [1, 0], [0, -1]]

        x = self.goal_area[0]
        y = self.goal_area[1]

        # Store adjacent directional options based on current cell location
        opened.append(((x, y), tune))
        opened.append(((x + trans[2][0],    y + trans[2][1]), tune))
        opened.append(((x + trans[3][0],    y + trans[3][1]), tune))
        opened.append(((x + trans[2][0],    y + trans[3][1]), tune))

        # Update agent's model's cell values
        for cell in opened:
            self.update_model(cell[0], cell[1])

        # Check all valid possible directions for model agent
        while self.model[self.maze_dim -1][0] == 0 and len(opened) != 0:
            location, tune = opened.pop(0)
            actions = self.act_legal(location)

            if 'up' in actions and self.count_grid[location[0]][location[1]] != 0:
                x2 = location[0] + trans[0][0]
                y2 = location[1] + trans[0][1]
                new_location = x2, y2
                if self.model[x2][y2] == 0 and self.count_grid[location[0]][location[1]] != 0:
                    opened.append((new_location, tune + 1))
                    self.update_model(new_location, tune + 1)

            if 'right' in actions:
                x2 = location[0] + trans[1][0]
                y2 = location[1] + trans[1][1]
                new_location = x2, y2
                if self.model[x2][y2] == 0 and self.count_grid[location[0]][location[1]] != 0:
                    opened.append((new_location, tune + 1))
                    self.update_model(new_location, tune + 1)

```

```
174
175             if 'down' in actions:
176                 x2 = location[0] + trans[2][0]
177                 y2 = location[1] + trans[2][1]
178                 new_location = x2, y2
179                 if self.model[x2][y2] == 0 and self.count_grid[location[0]][location[1]] != 0:
180                     opened.append((new_location, tune + 1))
181                     self.update_model(new_location, tune + 1)
182
183             if 'left' in actions:
184                 x2 = location[0] + trans[3][0]
185                 y2 = location[1] + trans[3][1]
186                 new_location = x2, y2
187                 if self.model[x2][y2] == 0:
188                     opened.append((new_location, tune + 1))
189                     self.update_model(new_location, tune + 1)
190
```

After the exploration and training phase, run 1, the agent will transition into testing phase, run 2. This phase the actions cost more points, than within the training phase. The agent will determine the best action to make at each cell based on the agent model created in the previous steps of the algorithm. This code will take place in make_model(), it is extensively commented within the code and is excessively long to post within this report.

Refinement

During the refinement process the following variables were tuned, changed, and experimented with the intention of improving the performance of the agent on the given environments. Both the min_actions and the make_model() tuning parameters were refined during the development process. Possible further improvements on these parameters is discussed in the Improvement section of this report.

IV. Results

Model Evaluation and Validation

These results were done with a min_action value of 700, if the agent explores all of the unique cells within the environment before this condition is met, the agent begins the testing run. Otherwise, the agent will make a minimum of 700 actions during exploration.

Test_maze_01.txt

```
Starting run 1.
Location: [9, 0]        Rotation: 0     Movement: 2     Action Count:498
Location: [9, 1]        Rotation: 90    Movement: 1     Action Count:499
Location: [11, 1]       Rotation: 90    Movement: 2     Action Count:500
Location: [11, 4]       Rotation: -90   Movement: 3     Action Count:501
Location: [9, 4]        Rotation: -90   Movement: 2     Action Count:502
Location: [9, 6]        Rotation: 90    Movement: 2     Action Count:503
Location: [10, 6]       Rotation: 90    Movement: 1     Action Count:504
Location: [10, 7]       Rotation: -90   Movement: 1     Action Count:505
Location: [11, 7]       Rotation: 90    Movement: 1     Action Count:506
Location: [11, 10]      Rotation: -90   Movement: 3     Action Count:507
Location: [11, 11]      Rotation: 0     Movement: 1     Action Count:508
Location: [8, 11]       Rotation: -90   Movement: 3     Action Count:509
Location: [8, 8]        Rotation: -90   Movement: 3     Action Count:510
Location: [6, 8]        Rotation: 90    Movement: 2     Action Count:511
Location: [6, 7]        Rotation: -90   Movement: 1     Action Count:512
Location: [5, 7]        Rotation: 90    Movement: 1     Action Count:513
Location: [5, 6]        Rotation: -90   Movement: 1     Action Count:514
Goal found; run 1 completed!
Task complete! Score: 33.600
```

Test_maze_02.txt

```
Starting run 1.

Location: [10, 0]       Rotation: 0     Movement: 3     Action Count:701

Location: [10, 2]       Rotation: 90    Movement: 2     Action Count:702

Location: [11, 2]       Rotation: 90    Movement: 1     Action Count:703

Location: [11, 1]       Rotation: 90    Movement: 1     Action Count:704

Location: [13, 1]       Rotation: -90   Movement: 2     Action Count:705

Location: [13, 4]       Rotation: -90   Movement: 3     Action Count:706

Location: [13, 5]       Rotation: 0     Movement: 1     Action Count:707

Location: [12, 5]       Rotation: -90   Movement: 1     Action Count:708

Location: [12, 6]       Rotation: 90    Movement: 1     Action Count:709

Location: [10, 6]       Rotation: -90   Movement: 2     Action Count:710

Location: [10, 8]       Rotation: 90    Movement: 2     Action Count:711

Location: [12, 8]       Rotation: 90    Movement: 2     Action Count:712

Location: [12, 10]      Rotation: -90   Movement: 2     Action Count:713

Location: [11, 10]      Rotation: -90   Movement: 1     Action Count:714

Location: [11, 12]      Rotation: 90    Movement: 2     Action Count:715

Location: [10, 12]      Rotation: -90   Movement: 1     Action Count:716

Location: [10, 13]      Rotation: 90    Movement: 1     Action Count:717

Location: [7, 13]       Rotation: -90   Movement: 3     Action Count:718

Location: [7, 11]       Rotation: -90   Movement: 2     Action Count:719

Location: [6, 11]       Rotation: 90    Movement: 1     Action Count:720

Location: [6, 10]       Rotation: -90   Movement: 1     Action Count:721

Location: [4, 10]       Rotation: 90    Movement: 2     Action Count:722

Location: [4, 8]        Rotation: -90   Movement: 2     Action Count:723

Location: [5, 8]        Rotation: -90   Movement: 1     Action Count:724

Location: [5, 6]        Rotation: 90    Movement: 2     Action Count:725

Location: [6, 6]        Rotation: -90   Movement: 1     Action Count:726
Goal found; run 1 completed!
Task complete! Score: 49.367
```

Test_maze_03.txt

```
Starting run 1.
Location: [12, 0]       Rotation: 0     Movement: 3     Action Count:701
Location: [12, 2]       Rotation: 90    Movement: 2     Action Count:702
Location: [13, 2]       Rotation: 90    Movement: 1     Action Count:703
Location: [13, 1]       Rotation: 90    Movement: 1     Action Count:704
Location: [15, 1]       Rotation: -90   Movement: 2     Action Count:705
Location: [15, 4]       Rotation: -90   Movement: 3     Action Count:706
Location: [15, 5]       Rotation: 0     Movement: 1     Action Count:707
Location: [14, 5]       Rotation: -90   Movement: 1     Action Count:708
Location: [14, 6]       Rotation: 90    Movement: 1     Action Count:709
Location: [13, 6]       Rotation: -90   Movement: 1     Action Count:710
Location: [13, 9]       Rotation: 90    Movement: 3     Action Count:711
Location: [13, 10]      Rotation: 0     Movement: 1     Action Count:712
Location: [14, 10]      Rotation: 90    Movement: 1     Action Count:713
Location: [14, 7]       Rotation: 90    Movement: 3     Action Count:714
Location: [15, 7]       Rotation: -90   Movement: 1     Action Count:715
Location: [15, 10]      Rotation: -90   Movement: 3     Action Count:716
Location: [15, 13]      Rotation: 0     Movement: 3     Action Count:717
Location: [15, 15]      Rotation: 0     Movement: 2     Action Count:718
Location: [12, 15]      Rotation: -90   Movement: 3     Action Count:719
Location: [10, 15]      Rotation: 0     Movement: 2     Action Count:720
Location: [10, 14]      Rotation: -90   Movement: 1     Action Count:721
Location: [8, 14]       Rotation: 90    Movement: 2     Action Count:722
Location: [8, 11]       Rotation: -90   Movement: 3     Action Count:723
Location: [8, 10]       Rotation: 0     Movement: 1     Action Count:724
Location: [9, 10]       Rotation: -90   Movement: 1     Action Count:725
Location: [9, 8]        Rotation: 90    Movement: 2     Action Count:726
Location: [8, 8]        Rotation: 90    Movement: 1     Action Count:727
Goal found; run 1 completed!
Task complete! Score: 50.367
```

Justification

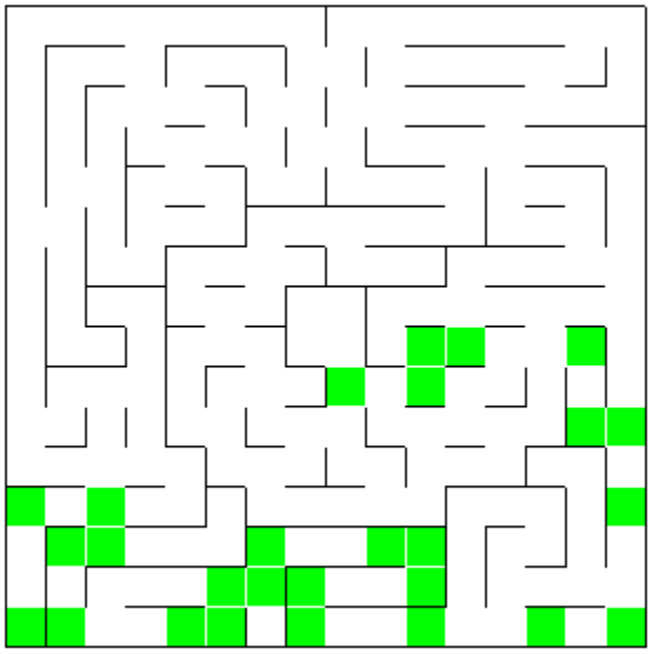
The test runs of the 3 given test maze environments went surprisingly well, given the minimum exploration actions was set to 700. This does not apply to the 10x10 due to the agent visiting every unique cell during the exploration run.

Maze	Minimum Base	Agent Results
test_maze_01.txt	63.85	33.6
test_maze_02.txt	64.85	49.37
test_maze_03.txt	65.85	50.37

V. Conclusion

Free-Form Visualization

This visualization displays the agent's planned and executed path when given the 'test_maze_03.txt' maze as an environment input.



Reflection

This report was an attempt to explain an abstraction and high-level view of the how the agent observers, plans, and actions within the given maze environment. The steps taken within the code have been outlined, based on the input given to the agent.

Improvement

The simplest improvement for score performance would be to decrease the min_actions taken by the agent. This would allow for the "baseline" to be lower than the

programmatically set 700 actions. The downside to this implementation is that the agent might:

- Never reach the goal
- Not visit the required cells for an optimum path, resulting in a higher final score

Alternatively, the agent make_model() tuning parameters could be changed, to increase/decrease the rewards given to the model's actions. This could increase the convergence during the exploration phase.

Also, the randomness could be decreased/increased during the exploration phase. Doing this could result in faster exploration but slower exploration could also occur.

This agent has been tested and the results have been based upon, the three test maze files provided by the publisher of this project. These files can also be found in the GitHub repository of this report.

References

1. https://docs.google.com/document/d/1ZFCH6jS3A5At7_v5IUM5OpAXJYiutFuSIjTzV_E-vdE/pub
2. https://en.wikipedia.org/wiki/Micromouse