Robot Motion Planning Capstone Project

Plot and Navigate a Virtual Maze

Nidhi Nayak

Udacity Machine Learning Nanodegree

September 15th, 2017

|  |  |
| --- | --- |
|  | 1 |
| **TABLE OF CONTENTS** |  |
| [**TABLE OF CONTENTS**](#page2) | [**1**](#page2) |
| [**DEFINITION**](#page3) | [**2**](#page3) |
| [Project Overview](#page3) | [2](#page3) |
| [Problem Statement](#page3) | [2](#page3) |
| [Metrics](#page4) | [3](#page4) |
| [**ANALYSIS**](#page4) | [**3**](#page4) |
| [Data Exploration](#page4) | [3](#page4) |
| [Exploratory Visualization](#page7) | [6](#page7) |
| [Algorithms and Techniques](#page12) | [11](#page12) |
| [Benchmark](#page15) | [14](#page15) |
| [**METHODOLOGY**](#page16) | [**15**](#page16) |
| [Data Preprocessing](#page16) | [15](#page16) |
| [Implementation](#page16) | [15](#page16) |
| [Refinement](#page19) | [18](#page19) |
| [**RESULTS**](#page21) | [**20**](#page21) |
| [Model Evaluation and Validation](#page21) | [20](#page21) |
| [Justification](#page23) | [22](#page23) |
| [**CONCLUSION**](#page23) | [**22**](#page23) |
| [Free-Form Visualization](#page23) | [22](#page23) |
| [Reflection](#page25) | [24](#page25) |
| [Improvement](#page27) | [26](#page27) |
| [**SOURCES**](#page28) | [**27**](#page28) |

2

1. **DEFINITION**

**Project Overview**

This project is motivated by Micromouse competitions that began in the 1970s. A micromouse competition is an event where a small robotic mouse attempts to navigate an unfamiliar grid maze. The micromouse starts in a corner and must find its way to a designated goal area commonly located near the center. The robot is allocated two attempts with the maze - the first attempt is an exploratory attempt where the robot will review and plan out its environment. The second attempt is where the robot will attempt to traverse an optimal route to the goal area as fast as possible. Using Python 3, our micromouse will attempt to navigate a virtual maze and determine the fastest times possible to traverse an optimal route to the goal area located in the center in a series of test mazes.

**Problem Statement**

The objective is simple - get to the middle of the maze as fast as possible. The robot will start in the bottom-left corner of the maze and must navigate from the origin to the goal in the shortest possible time, and in as few moves as possible. The maze exists on an n x n grid of squares, n even. The minimum value of n is twelve, the maximum sixteen. The whole boundary of the maze is surrounded by walls preventing the micromouse from traveling outside the maze. The initial position will have walls on the left, right, and back sides, so the first move will always be forward. 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.

The micromouse will route a first trial that is exploratory in class where its main aim is to build information about the structure and shape of the maze, that includes all possible routes to the goal area. The micromouse need to travel the goal area in order to complete a successful exploration trial, but is permissible to continue its exploration after reaching the goal. In the second trial, the micromouse will recollect the information from the first trial and will try to reach the goal area in an optimal route. The micromouse’s performance will be recorded by adding the following:

* Total number of steps in the exploration trial divided by 30
* Total number of steps to reach the goal in the optimization (second) trial

Each of the trial is capped at 1000 steps. The micromouse can only make 90 degree turns (clockwise or counterclockwise) and can move up to 3 spaces forward or backwards in a single movement.

**Metrics**

As per the previous sections, the main evaluation metric to quantify the performance of the benchmark and solution models is:

**score = [Number of Steps in Run 2] + [Number of Steps in Run 1 / 30]**

**​**This evaluation metric is impacted by boththe exploration run and optimization run, however the optimization run will impact the score significantly more than the exploration run. Our goal is to minimize this score, which would be the result of an optimal run.

For example: let’s assume the micromouse took 400 steps during its exploration run, and 16 steps during its optimization run. The score would be equal to: 16(steps from Run 2) + 400/30(steps from Run 1 divided by 30) = 29. If the optimization algorithm found the optimal route for this particular maze, then 29 would be the optimal score.

1. **ANALYSIS**

**Data Exploration**

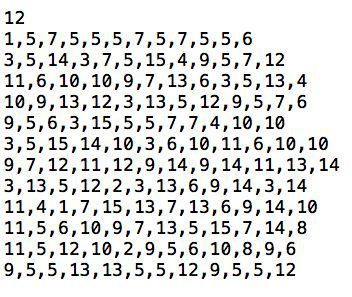
Starter code for this project was provided by Udacity and included:

* **robot.py​**- This establishes the Robot class, but is the main script where modificationswere made to the project.
* **tester.py​**- This script is run to test the robot’s ability to navigate the mazes.
* **maze.py​**- This script is used to construct each maze and interacts with the robotwhenever it is moving or checking its sensors.
* **showmaze.py​**- This script creates a visual layout of each maze.
* **test\_maze\_##.txt​**- These files provide three sample mazes upon which to test the robot.

Each maze is sketched on a square grid containing of either 12, 14, or 16 rows and columns. The maze is surrounded by walls neighboring its perimeter that act as a barrier, blocking all movement.

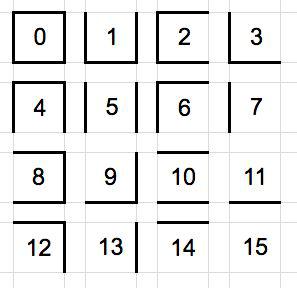
Our robot leads in the bottom left-hand corner of the maze (coordinate (0,0)) facing in an upward direction. It has walls on its left, right, and bottom sides with an opening on its top side, forcing its first move to be forward (or ‘Up’). The robot aim is to navigate to a 2-by-2 goal area located in the center of the maze. A successful run of the maze is not complete without the robot getting to the goal at least once.

The relevant information on the structure of each maze is delivered by the test\_maze\_##.txt file, which, to the average eye, will just look like several rows of numbers.

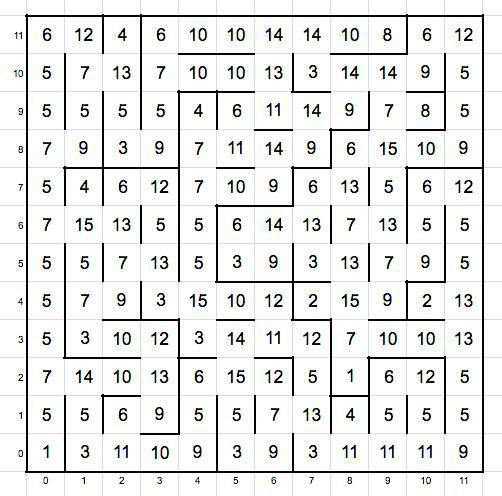


**Figure 1: Contents of ​*test\_maze\_01.txt***

The first line of the text file is a number (12, 14, or 16) which defines the length of one side of the square maze. The subsequent lines contain comma-delimited numbers ranging from 1 to 15. These numbers define the position of walls and openings for each cell in the maze grid. 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 relates 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 7 translates to a square that has walls on its left and right sides and openings on its bottom and top sides (1\*1 + 1\*2 + 1\*4 + 0\*8 = 7). Because of array indexing, the text file defines the maze column-by-column from left to right, so the first number actually represents the starting cell in the bottom-left corner. Maze cell labels follow a Cartesian coordinate system with the rows and columns ranging from 0 to 11.



**Figure 2: Wall Description Numbers**

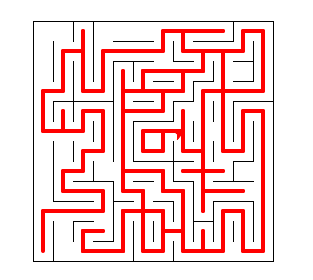
****

**Figure 3: Test Maze 1 with Wall Description Numbers**

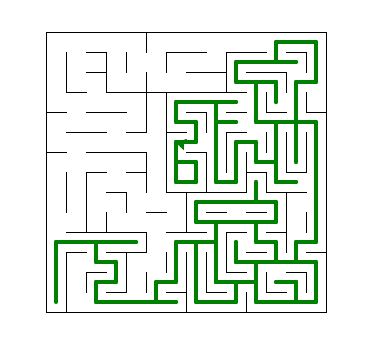
It can be presumed that the robot is positioned directly in the center of each cell and is facing either left, up, right, or down. The robot is furnished with three sensors. ​***Sensor 0****​* is on its left side, ​***Sensor 1****​* is on top, and ​***Sensor 2*** *​*in on the right side. Each sensor can sense an opening to an adjacent cell or the existence of a wall blocking its path. Our robot has the ability to precisely rotate clockwise or counterclockwise in 90° intervals and can select to move forward or backward up to three consecutive spaces. One time-step passes after each movement to an open cell, where upon the sensors will update their readings for the new position.

Note: Relevant python function in maze.py is is\_permissible() and *init*(), is\_permissible() checks whether or not a cell is passable in the given direction. Cell is input as a list. Directions may be input as single letter 'u', 'r', 'd', 'l'. init() reads the maze as text file and build a maze object based on above explained logic.

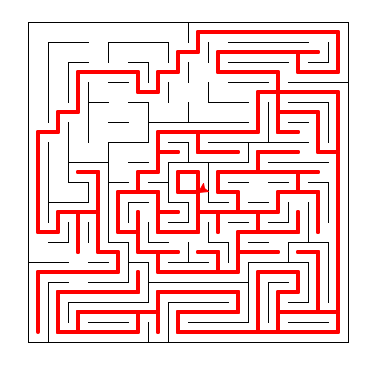
**Exploratory Visualization**

****

**Figure 4: Uncovered cells from ​*Exploration Trial​* in Test Maze 1**

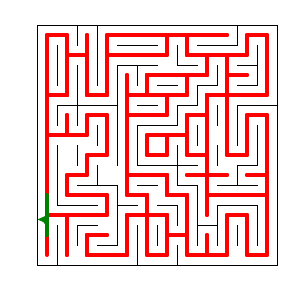
****

**Figure 5: Uncovered cells from ​*Exploration Trial​* in Test Maze 2**

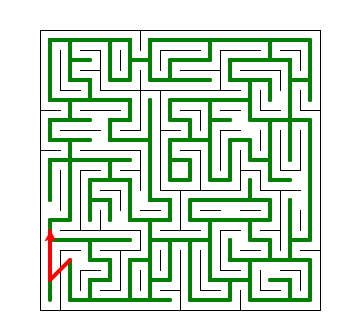
****

**Figure 6: Uncovered cells from ​*Exploration Trial​* in Test Maze 3**

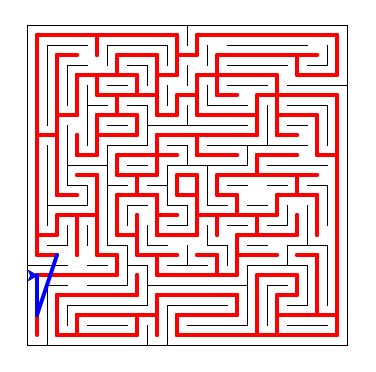
Figure 4,5,6 displays Test Maze, which is 12-by-12 (labeled 0 to 11). The cells in white have all been uncovered by the robot during the ​***Exploration Trial****​*, while the squares highlighted in red were discovered. bottom left-hand corner of the maze is the starting cell and the four cells in the middle in square shape represent the goal area.

****

**Figure 7: Covering all Uncovered cells from ​*Exploration Trial​* in Test Maze 1**

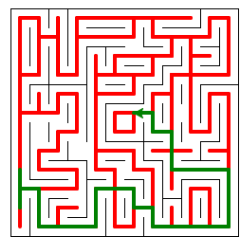
****

**Figure 8: Covering all Uncovered cells from ​*Exploration Trial​* in Test Maze 2**

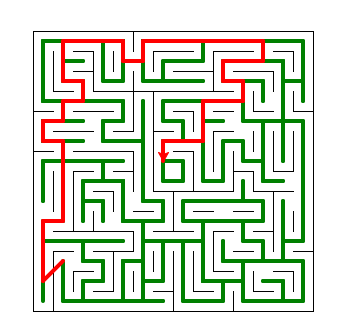
****

**Figure 9: Covering all Uncovered cells from ​*Exploration Trial​* in Test Maze 3**

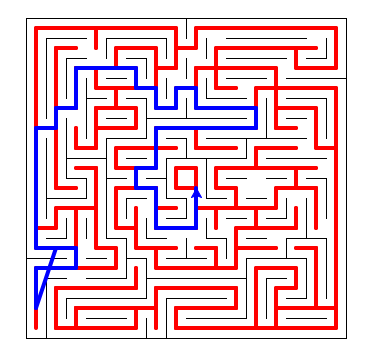
In order to uncover a significant amount of the maze, the robot was directed to continue its ​***Exploration Trial*** *​*(even if it had already found the goal) until it visited 100% of the total maze.



**Figure 10: Covering shortest path from Optimization Trial Test Maze 1**



**Figure 11: Covering shortest path from Optimization Trial Test Maze 2**



**Figure 12: Covering shortest path from Optimization Trial Test Maze 3**

Once the robot has discovered 100% of the maze and has found the goal at least once, it will be directed to reset and move back to the starting position to begin the Optimization Trial.

As the robot maneuvers the maze, being steered by the value function, the robot is able to determine an optimal path (to the best of its knowledge) as shown below:

**Optimal Path for Test Maze 1:**

[[6, 5], [6, 6], [7, 6], [7, 5], [8, 5], [8, 4], [8, 3], [9, 3], [10, 3], [11, 3], [11, 2], [11, 1], [11, 0], [10, 0], [9, 0], [8, 0], [7, 0], [7, 1], [6, 1], [6, 2], [5, 2], [4, 2], [4, 1], [4, 0], [3, 0], [2, 0], [1, 0], [1, 1], [1, 2], [0, 2], [0, 1]]

**Optimal Path for Test Maze 2**

**[[7, 6], [7, 7], [6, 7], [6, 8], [7, 8], [8, 8], [8, 9], [8, 10], [9, 10], [10, 10], [10, 11], [9, 11], [9, 12], [10, 12], [11, 12], [11, 13], [10, 13], [9, 13], [8, 13], [7, 13], [6, 13], [5, 13], [5, 12], [4, 12], [4, 13], [3, 13], [2, 13], [1, 13], [1, 12], [1, 11], [2, 11], [2, 10], [1, 10], [1, 9], [0, 9], [0, 8], [1, 8], [1, 7], [1, 6], [1, 5], [1, 4], [0, 4], [0, 3], [0, 2], [0, 1]]**

**Optimal Path for Test Maze 3**

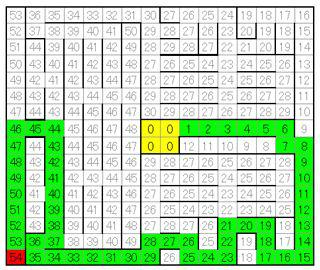
**[[8, 7], [8, 6], [8, 5], [7, 5], [6, 5], [6, 6], [6, 7], [5, 7], [5, 8], [6, 8], [6, 9], [6, 10], [7, 10], [8, 10], [9, 10], [10, 10], [11, 10], [11, 11], [10, 11], [9, 11], [8, 11], [8, 12], [7, 12], [7, 11], [6, 11], [6, 12], [5, 12], [5, 13], [4, 13], [3, 13], [2, 13], [2, 12], [2, 11], [1, 11], [1, 10], [0, 10], [0, 9], [0, 8], [0, 7], [0, 6], [0, 5], [0, 4], [1, 4], [2, 4], [2, 3], [1, 3], [0, 3], [0, 2], [0, 1]**

**Algorithms and Techniques**

When it comes to solving maze problems, there be several different strategies and algorithms one might use while attempting solve a maze. Before trying to program an exact algorithm to use, I found a bit of information on several applicable solutions - some more suitable for the ​***Exploration Trial****​* and others best appropriate for the ​***Optimization Trial****​*.

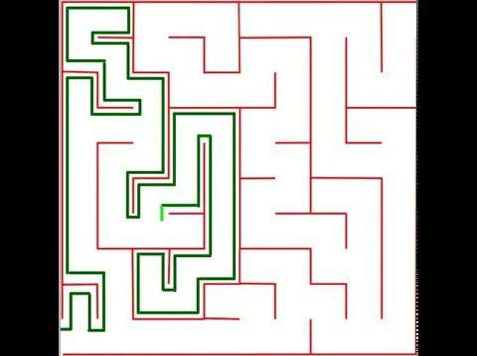
**Random Turn:** Algorithmic rule that brand a random movement decision at each stair, slightly favoring the unexplored path. This is entirely simple and slow, but should eventually discover the goal.

**Flood Fill​**: Algorithmic rule that starts in the middle and will fill each surrounding cell with a number of its relative distance from the end area.



**Figure 13: Example of Flood Fill Algorithm**

**Wall Follower:** A Wall Follower algorithm works by simply leading the micromouse to take every left or right turn possible, which should finally lead to the goal area (assuming the micromouse does not get caught in a circular loop).



**Figure 14: Example of Left Hand Rule**

**A\*:** A\* is a best-first search algorithm that searches for all possible solutions to the goal area and will choose the shortest path from the starting node to the ending node. Starting from a specific node, it creates a tree path of each possible direction until it reaches the goal area.

**Depth-First Search & Breadth-First Search: ​** These two algorithmic rule are known as graphtraversal algorithmic rule. Depth-First Search starts at a root and follows a single path as far as feasible before seeing a different path. Similar to Depth-First Search, Breadth-First Search will start at a tree root, but will consider all neighboring paths before looking on to its next level neighbors. Both algorithms are proper for exploring the maze and finding an optimal path, but Breadth-First Search would require the robot to travel back to previously explored parts of the maze to discover new paths, and would thus likely cause a much larger exploration score.

For this project, I decided to use a modified version of Depth-First Search for the ​***Exploration Trial****​* and used Flood Fill to create an optimal policy to use for the ​***Optimization Trial.***

**Benchmark**

The benchmark model will be calculated by the performance score earlier discussed in the ***Problem Statement****​* section:​**score = [Number of Steps in Trial 2] + [Number of Steps inTrial 1 / 30] ​**. As we have capped the number of achievable steps at 1000, this will be thesimplest form of a benchmark performance. The benchmark model will contain a robot that makes a random movement decision at each step. If the mouse is able to navigate to the goal in less than 1000 steps in its ​***Exploration Trial****​*, a new benchmark should be fixed to the total number of steps in Trial 1. Because the robot has gained knowledge of the maze layout in its ​***Exploration Trial****​*, it should then optimize its route to the goal in the ​***Optimization*** ***Trial****​*. Regardless of the number of steps taken during the​***Exploration Trial****​*, by nature ofdesign, an optimal path exists for each maze - which is the minimum amount of steps needed to get from the starting location to the goal area.

Maze 1, 2, and 3 have optimal paths of 17, 23, and 25 steps, respectively. A functional benchmark would contain the sum of an ​***average****​* path to the goal and one thirtieth of an ***average****​* amount of steps required to explore the maze. The challenge is defining what“average” actually means. Given the size of each maze and the allotted maximum amount of steps for the ​***Exploration Trial*** *​*, an ​***average****​* robot should absolutely be able to discover the goal area during exploration in far fewer than 1000 steps. Because the goal is to find an optimal path to the goal, the optimal routes of 17, 23, and 25 will be used for the ​***Optimization Trial*** benchmark. For the ​***Exploration Trial****​*, a benchmark model must discover every cell, which is impossible to achieve by only visiting every cell once. To discover every cell, a robot must visit a number of cells multiple times. In order to balance the use of the optimal route for the benchmark’s performance in the ​***Optimization Trial****​*, the total number of cells for each maze will be multiplied by 2.5 to generate the number of steps the benchmark model will take during the exploration phase.

**Benchmark Maze 1 Score: ​**17 + (144 \* 2.5) / 30​**= 29.00**

**Benchmark Maze 2 Score: ​**23 + (196 \* 2.5) / 30​**= 39.33**

**Benchmark Maze 3 Score: ​**25 + (256 \* 2.5) / 30​**= 46.33**

1. **METHODOLOGY**

**Data Preprocessing**

No data preprocessing was needed for this project. The robot collects the essential information from its sensors and the maze environment. The robot will collect and store information until it has enough to apply an algorithm to optimally solve the maze.

**Implementation**

After fully describing and understanding the problem, and then considering the uses of varying algorithms, it became apparent that the implementation of this project must be broken into smaller sub-sections.

* 1. **Interpreting Sensor Readings and Recording Gathered Information**
  2. **Robot Movement During ​*Exploration Trial***
  3. **Finding Goal Area During ​*Exploration Trial***
  4. **Identifying Possible and Optimal Paths to Goal**
  5. **Ending ​*Exploration Trial​* and Starting ​*Optimization Trial***

1. **Interpreting Sensor Readings and Recording Gathered Information. ​**The robot’sinformation of the maze is stored in a two-dimensional numpy array, called Maze Grid. This liststores the information gained on the actual structure of the maze walls recognized from the robot’s sensors during its ​***Exploration Trial****​*. This working-memory list, occupied by the ***wall\_locations****​* method, will enable the robot to know the locations of walls and openings in themaze, and what area of the maze is undiscovered.

Beginning each time-step, the robot collects a new set of sensor readings from its left, front, and right sensors. Joining the sensor readings with information defining the direction the robot is facing and its current location, the robot is able to produce and store a four-bit number describing the surrounding maze walls and openings for its specific location (previously discussed in the ​***Data Exploration****​* section).

1. **Robot Movement During ​*Exploration Trial​*. ​**After storing information on the maze walls,the next step is to determine how the robot should move throughout the maze during its ***Exploration Trial****​*. Not only we are concerned with the robot moving through the openings in thewalls, but we also want the robot to favor previously-undiscovered cells while maneuvering in a general direction toward the goal area.
2. **Finding Goal Area During ​*Exploration Trial​*. ​**The next process in implementation wasdetermining how to ensure the robot will discover the goal area at least once during its ***Exploration Trial****​*, and how much of the maze the robot should explore after it has found thegoal (if continue exploring at all).
3. **Identifying Possible and Optimal Paths to Goal. ​**Now that the robot has discovered thegoal area, it is time to determine all possible paths to the goal, and deduce one optimal path for the robot to take during its second trial. Again, borrowing learnings from Udacity’s ArtificialIntelligence for Robotics course, I implemented a Dynamic Programming method that given the map of the maze and the location of the goal area, outputs anoptimal path to the goal from any starting cell.The Dynamic Programming method works by considering the locations of the walls in the maze and creates a Policy Grid that displays the optimal action (move left, right, up, or down) at every single grid cell leading to the goal area. One issue with Dynamic Programming is that it is very computationally involved
4. **Ending ​*Exploration Trial​*and Starting ​*Optimization Trial​*. ​**The last portion ofimplementation is determining when and how to end the ​***Exploration Trial****​*, and start the ***Optimization Trial****​*. Once the robot has visited the goal at least once and has uncovered 100% of the map. Once an optimal path is identified, the robot stores it with the proper movements and rotations to follow the path. ‘Rotation’ and ‘Movement’ are both updated to ‘Reset’ which signals the ​***Optimization Trial***.

**Refinement**

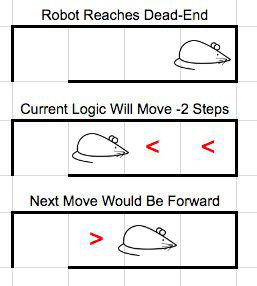
Techniques used for implementation were DFS for exploration and Dynamic Programming for optimization.

I originally allowed the ​***Exploration Trial****​* to end once the goal area was found during exploration, but realized that the robot had not even seen half of the maze before stopping and is potentially missing superior paths to the goal. I then required the robot to uncover 100% of the maze before moving on to ​***Optimization Trial***.

Thinking it would limit the amount of moves in the ​***Exploration Trial****​*, I initially directed the robot to move -1 space every time it found a dead-end, but it had a tendency of wanting to travel right back into the dead-end when its only possible move was still ‘forward’ after it had retreated. This ultimately led to the robot reaching the step limit of 1000 by going back and forth between the two same cells over and over again. I then tried implementing logic that would require the robot to move -2 spaces if it had reached a dead-end, and if the previous cell it had visited had walls on the left and right sides. This solution appeared to work wonderfully on the 12-by-12 maze and the 14-by-14 maze, but the robot was unable to find an optimal path in the 16-by-16 maze.

To further refine my code in the future, I would likely spend time preventing the robot from visiting all four cells in the goal area during the ​***Exploration Trial****​*. Because the ​***Exploration Trial*** is so lightly weighted on the final score, I did not make this adjustment a priority in my current model, and the robot will visit each cell in the goal area multiple times, even though I recognize this to be unnecessary.

Another weakness may arise if the robot was on a different maze that included numerous dead-ends that span more than three cells, which I have dubbed a “deep dead-end”. The robot’s logic to this point only requires it to move backwards a maximum of two spaces, but it could be prone to an endless exploration loop if moving backwards two spaces did not reveal additional options to travel other than forward (see Figure 15 below). This issue does not arise with the current mazes used for this project, but could be an issue with alternative maze designs.



**Figure 15: Robot Dead-End Loop**

**IV.** **RESULTS**

**Model Evaluation and Validation**

The model is robust because it is programmed to dynamically find the goal in any type of maze. Because it requires a coverage of a hundred percent of total cells before moving onto the ***Optimization Trial***. The major benefit of Dynamic Programming is that it will reveal a path to the goal from any cell in the maze - not just the starting point.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Test Maze 1 | Test Maze 2 | Test Maze 3 |
|  |  |  |  |  |
|  | Dimensions | 12-by-12 | 14-by-14 | 16-by-16 |
| Algorithm Used | Benchmark Score | **29.00** | **39.33** | **46.33** |
|  |  |  |  |  |
|  |  |  |  |  |
| Flood Fill | **Score** | **21.70** | **36.43** | **35.56** |
|  |  |  |  |  |
| Depth-First Search | **Score** | **21.70** | **36.67** | **35.56** |
|  |  |  |  |  |
| Breadth-First Search | **Score** | **21.70** | **37.00** | **36.10** |
|  |  |  |  |  |

**Without Optimization:**

Overall, I would argue that the results can be trusted. the results and performance tend to be consistent, robust, and also routinely outperform the benchmark model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Test Maze 1 | Test Maze 2 | Test Maze 3 |
|  |  |  |  |  |
|  | Dimensions | 12-by-12 | 14-by-14 | 16-by-16 |
| Algorithm Used | Benchmark Score | **29.00** | **39.33** | **46.33** |
|  |  |  |  |  |
|  |  |  |  |  |
| Flood Fill | **Score** | **21.70** | **36.43** | **35.56** |
|  |  |  |  |  |
| Depth-First Search | **Score** | **21.70** | **36.67** | **35.56** |
|  |  |  |  |  |
| Breadth-First Search | **Score** | **21.70** | **37.00** | **36.10** |
|  |  |  |  |  |

**With Optimization:**

**Justification**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Min. |  |  |  |  |
| Coverage |  | Test Maze 1 | Test Maze 2 | Test Maze 3 |
|  |  |  |  |  |
|  | Exploration Steps | 146 | 266 | 304 |
| 70% |  |  |  |  |
| Optimization Steps | 17 | 28 | 29 |
|  | **Model Score** | **21.87** | **36.87** | **39.13** |
|  |  |  |  |  |
|  | **Benchmark Score** | **29.00** | **39.33** | **46.33** |
|  |  |  |  |  |

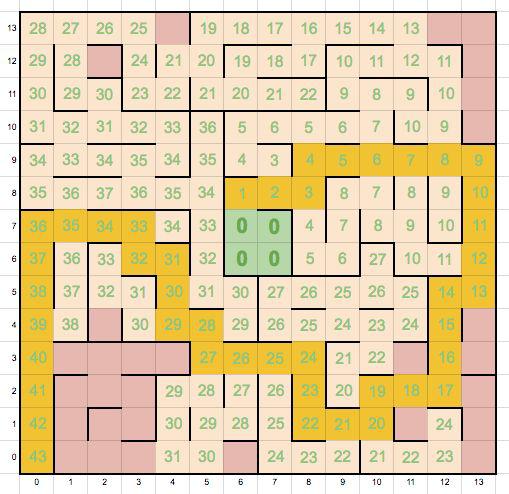
The final results of the model outperformed the benchmark model, however the model failed to find the optimal solution for Test Maze 2 (23 steps) and Test Maze 3 (25 steps), but the routes the model did find were not too far off. By adjusting the preference of taking a route which allows many consecutive steps of two to three spaces, uncovering these routes should be possible.

Although every model can be fine tuned and improved, I would argue that the performance of my model has resulted in a final solution that is significant and robust enough to have solved the problem.

1. **CONCLUSION**

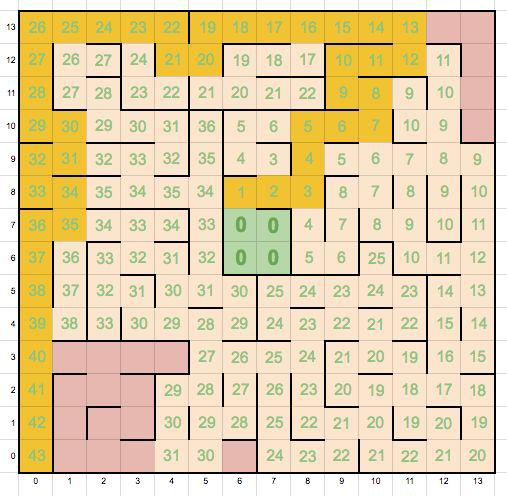
**Free-Form Visualization**

In this section I will dig into the performance of Maze 2 with a required coverage of 85%, and attempt to explain why its path was more optimal than the same maze with a required coverage of 90%. As you can see in Maze 2 with 85% required coverage (Figure 15), the robot only needed 23 steps to reach the goal area, which is far fewer than any other trial. Peering deeper, I can conclude a little bit of luck came into play here. Because the robot did not discover cell (4,13) during the ​***Exploration Trial****​*, the Path Value became skewed, ultimately leading the robot to make a different path decision at cell (1, 7).



**Figure 15: Maze 2, 85% Coverage Required, 23 Steps**

Figure 16 shows the Path Value for Maze 2 where 90% coverage was required and the robot took 27 steps. As you can see, starting in cell (1,7), the robot equally could have travelled to cell (2,7) as opposed to cell (1,8), because either cell had a cost of 34.

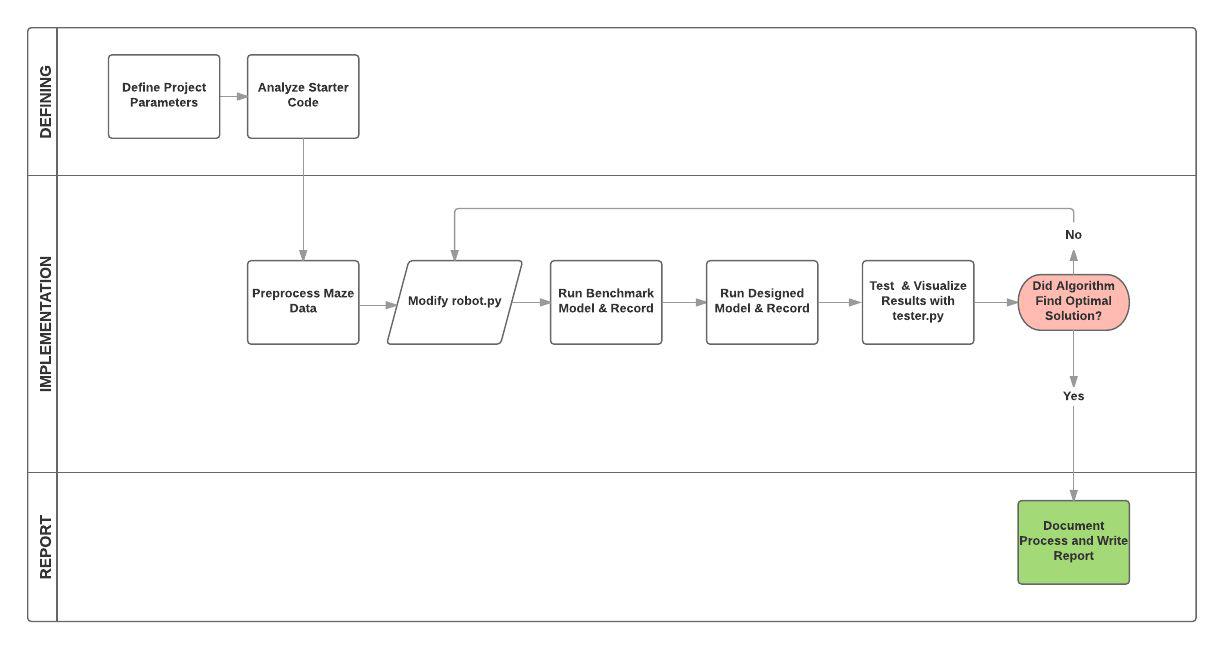


**Figure 16: Maze 2, 90% Coverage Required, 27**

In the ​***determine\_next\_move****​*method, an A\* technique is used to generate a list of all possible moves the robot is able to take in a particular cell. In cases where multiple paths appear to be equal, the robot will choose the path aligned with the first sensor reading. Because the sensors are list in the order of ​***Left, Up, Right****​*, all else being equal, the robot will choose to travel ​***Left*** before ​***Up****​*, and ​***Up****​*before ​***Right*** *​*. Although both paths span a total of 43 cells, the robot is, unfortunately, not programmed to favor a path that allows making many consecutive moves of up to 3 steps. In the 85% path, the robot has the benefit of covering more spaces by taking larger steps more frequently than in the 90% path. This functionality should be addressed in future revisions, as it can make a large impact on the final outcome.

**Reflection**

I was inspired to take on this project because I have always had an interest in robotics. Now, with more and more of the products in the world becoming “smart”, my interest in the world of robotics has never been greater.



**Figure 17: Project Development Process**

The entire process used for this project can be defined in the following steps:

1. Record and translate sensor readings into maze information and incorporate it with the robot’s orientation (heading) and location.
2. As the robot navigates the maze, update the map of the wall locations in the maze.
3. Implement a heuristic map to help lead the robot toward the goal area during exploration.
4. Update the robot’s location after every move and output the rotation angle and movement directions to assist the robot to the next space.
5. Continue the previous steps until the robot has discovered the goal area at least once and has uncovered a minimum of 70% of the possible maze cells.
6. Calculate an optimal path to the goal area.
7. Initiate the second run and ensure the robot follows the optimal path to the goal area.

My final model fits as a solution to the problem, and I would recommend its use in a general setting to solve this problem. I would hesitate to say it is the best possible model, because mazes come in different shapes and forms. Some algorithms are more suited for solving a particular type of maze than others, but this model should serve as a good place to start.

**Improvement**

For specific improvements, I feel that my method is a bit more computationally involved than other potential solutions, so I would like to find a way to help speed up the script processing. As previously discussed, by implementing a logic that allows the robot to favor and identify optimal routes that allow taking multiple consecutive steps could lead to a great improvement. In addition, identifying and preventing the robot from entering repetitive loops could add to the model’s robustness.

I would also be interested in implementing an array of alternative algorithms, mentioned in the ***Algorithms and Techniques****​*section, and comparing their performance against my currentmodel. Because certain algorithms work better with certain maze styles, I would love to implement a model that has an arsenal of multiple path finding algorithms at its disposal. Based on its findings in the ​***Exploration Trial****​*, the model will choose to implement an algorithm that is most suited for a particular type of maze layout. For example, if the goal area is located in a corner of the maze, the model may use one type of algorithm to determine an optimal path, but if the goal is located in the center, it would use a different algorithm.

**SOURCES**

***Udacity Project Files***

<https://docs.google.com/document/d/1ZFCH6jS3A5At7_v5IUM5OpAXJYiutFuSIjTzV_E-vdE/pub>

***Robots Dreams***

<http://isobe.typepad.com/robotics/micromouse/page/12/>

***Maze Solving Algorithms on YouTube***

<https://www.youtube.com/watch?v=NA137qGmz4s>

***A Note on Two Problems in Connexion with Graphs***

<http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf>

***Micromouse USA***

<http://micromouseusa.com/>

***USC Micromouse Project***

<http://robotics.usc.edu/~harsh/docs/micromouse.pdf>

Red Blob Games

<http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>

A Guide to Heuristic Based Planning

<https://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/maxim/files/hsplanguide_icaps05ws.pdf>

Dijkstra’s algorithm revisited: the dynamic programming connexion

<http://matwbn.icm.edu.pl/ksiazki/cc/cc35/cc3536.pdf>

Graph Traversal

<http://people.mpi-inf.mpg.de/~mehlhorn/ftp/Toolbox/GraphTraversal.pdf>

Udacity’s Artificial Intelligence for Robotics Course

<https://www.udacity.com/course/artificial-intelligence-for-robotics--cs373>