project_report

October 26, 2016

---

4 days left ***********

1 Machine Learning Capstone Project

1.1 Plot and Navigate a Virtual Maze

Machine Learning Nanodegree (Udacity) Project submission by Edward Minnett (ed@methodic.io). October 30th 2016. (Revision 1)

1.2 I. Definition

1.2.1 Project Overview

The primary purpose of this project is to program a virtual robot to explore a simple maze, find a path from a corner of that maze to its center, and traverse the path from start to finish. Of course, there will be an optimal path and less optimal paths as well as efficient and inefficient ways of finding those paths. The virtual robot's performance will be scored according to how many steps it uses to complete the maze. The parameters and context of this project are inspired by the Micromouse robot competition [1] and is, in effect, a virtual version of the Micromouse problem. The robot mouse is given the opportunity to traverse the maze twice. The first traversal offers the opportunity to explore and map the maze while the second run requires the robot mouse attempt to reach the center of the maze as quickly as it can given the knowledge acquired while exploring the maze. The goal of this project is to find 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.

1.2.2 Problem Statement

Each maze used in the project follows a strict specification. The maze is a fully enclosed square with each edge being an even number of units long. The test mazes one, two and three are sized 12x12, 14x14, and 16x16 respectively. At the center of each maze is a 2x2 space enclosed by 7 walls and one entrance. This space is the goal of the maze and the robot must enter the space in order to successfully complete a run of the maze. The robot will always start in the bottom left corner of the maze and this space will always have three walls with a single opening at the top of the space.

Each 1x1 space within the maze that is occupiable by the robot can have one of 16 possible shapes defined by the presence or absence of a wall on each side. There is a 17th possible shape where all four walls are present, but this space would not be occupiable by the robot and as a result isn't included in any of the test mazes. It is worth noting that, as mentioned previously, the start space is always the same shape and edge spaces as well as the center goal spaces have fewer than 16 potential shapes given the constraints that the maze is fully enclosed and the goal spaces only have a single entrance.

In the case of the virtual environment, the mazes are defined by a text file where the value on the first line states the dimension of the maze followed by a series of lines of comma separated values. Due to array indexing, the first line is the left side of the maze and the first value in the first line is the bottom-left corner of the maze. Each value represents a space within the maze and the value for each space is a four-bit integer from 1 to 15 (0 represents the unoccupiable, fully enclosed space). Each bit in the four-bit integer represents a wall or opening, 0 or 1 respectively, on the side of the space. The bit sequence starts with the 1s at the top of the space with additional bit (2s, 4s, and 8s) representing the edge clockwise around the space. As an example, the starting space which has a wall on every side except the top which is an opening would be represented as follows:

```
1*1 + 0*2 + 0*4 + 0*8 = 1
```

In order to simplify the problem, the robot is considered to be in the center of the space it occupies and can only face in one of the four cardinal directions. The robot is capable of taking perfect sensor readings and making perfect movements resulting in a fully deterministic agent, but its sensors can only detect the distance to the next wall in each of the three directions, forward, left, and right. The robot is only capable of moving forward and backward, but is capable of moving up to three spaces in either direction. At the beginning of each time-step, the robot receives its sensor readings based on the direction it is facing after the last movement. It can then chose to keep its current direction or rotate either left or right by 90 degrees before choosing to move forward or backward. If the robot hits a wall before completing its movement, the robot will remain where it is facing the wall that blocked its path. The act of movement ends the time-step allowing the robot to receive its new sensor readings starting the next sense-move loop.

More specifically, the sensor reading are passed to the robots `next_move` function as a list of 3 integers representing the distances to the left closest wall, forward closest wall, and right closest wall in that order. If the wall is an edge of the currently occupied space, the distance is 0. The `next_move` function must then return two values representing the robots rotating and movement in that order. The rotation value can be one of -90, 0, or 90 representing counterclockwise, no rotation or, clockwise rotation respectively. The movement value must be an integer between -3 and 3 including those values where a negative integer is a backward movement and positive is forward.

In its first run of the maze, the robot may move freely within the maze in order to explore and map it. If the robot chooses to end the exploration run, it can return 'Reset' for both its rotation and movement values. This will reset the robots position top the bottom left corner or the maze and begin the second run of the maze. The second run ends when the robot reaches the goal at the center of the maze.

In order to solve this problem, the robot will need to explore the maze sufficiently and maintain a map of the explored spaces of the maze. In order to successfully attempt the second run of the maze, the robot will need to discover at least one path to the goal. If the robot explores the whole maze, then it should be able to determine the optimal path to the goal and use that path in the second run of the maze. The ideal robot strategy for this problem successfully and consistently finds the optimal path to the goal while minimising the steps required to explore the maze. There are a variety of ways this can be attempted and they will be discussed in detail in the section on algorithms and techniques later on in this report.

1.2.3 Metrics

For each of the test mazes, the virtual robot is given the opportunity to move through the maze twice. In the first run, the virtual robot is allowed to move freely through the maze in an attempt to explore and map it. It is free to continue exploring the maze even after entering the goal at its center. Once the virtual robot has found the goal, it may choose to end the exploration run at any time. For the second run, the virtual robot is expected to traverse the maze and reach the goal at its center as quickly as it can. The score awarded to the robot for each maze is calculated as the sum of the following:

- The number of steps taken to reach the goal during the second run of the maze.
- 1/30th of the number of steps taken while exploring the maze in the first run through.

A virtual robot with a smaller score performs better than one with a larger score.

Each run of the maze is limited to 1000 steps.

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 through the maze is likely to, but may not necessarily, perform worse than a robot that takes the time to explore the maze sufficiently to find the optimal path to the goal.

1.3 II. Analysis

1.3.1 Data Exploration

In order to explore how the mazes are constructed, we will take a closer look at Test Maze 1. Below is the maze showing the optimal path to the goal which takes the robot 17 steps to follow.

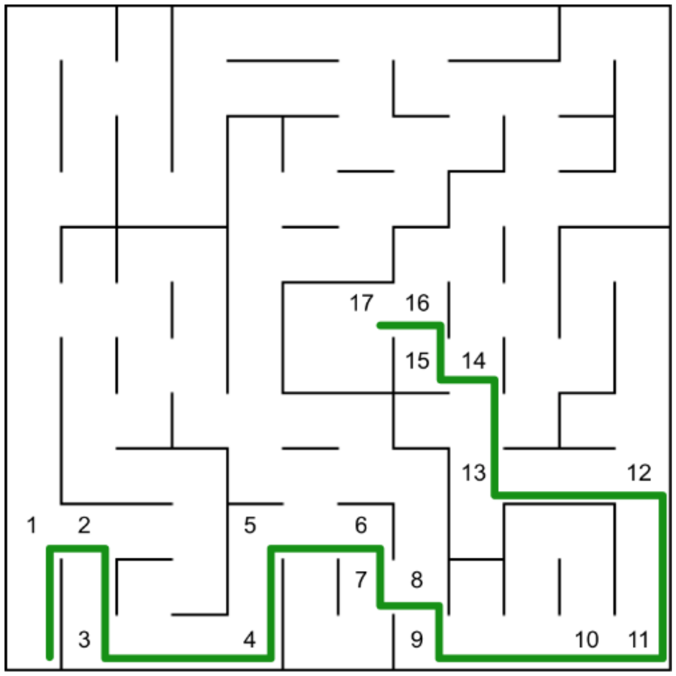

Figure 1: Test Maze 1 with a green line showing the optimal path from the start to goal positions and the sequence of steps along that path.

There are a few structural elements present in this maze that are worth noting. The zig-zag walls that meet the two right corners of the goal area means that all paths to the goal must first travel to one of the right corners of the maze before reaching the goal. Given that the top right corner of the maze is the furthest point from the start position, it is not surprising the optimal path passes around the bottom right corner. It is also interesting to note that there are very few horizontal walls in the left third of the maze allowing the robot to explore that part of the maze more efficiently than if there were more corners that would prevent the robot from seeing down long ‘corridors’. This is a particularly important observation as it shows just how much of the maze can be explored with just a few steps. The first sensor reading alone will provide enough information to show that there is not a single horizontal wall in any of the left most spaces of the maze.

1.3.2 Exploratory Visualization

The structure of Test Maze 1 discussed in the previous section is even more obvious when the maze is displayed as a heatmap. The value for each space is the minimum number of steps required for the robot to reach that space.

The heatmap makes it very clear how the robot needs to visit one of the right-hand corners before traversing the right-most triangular wedge of the maze that leads to the goal. The heatmap also shows how few steps are required for the robot to traverse the left third of the maze. Even without the the green line showing the optimal path to the goal, it would be very clear that the robot’s best strategy is to head to the bottom right corner rather than the top right corner. That said, this is just an illustration showing how the optimal path compares to the alternatives. In reality, the robot will calculate the optimal path and use an efficient strategy to explore the maze that takes advantage of any long corridors within the maze.

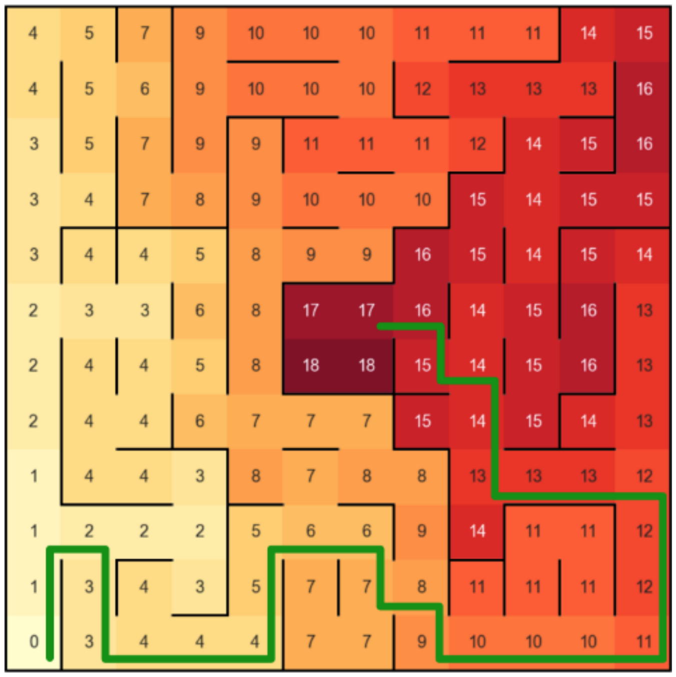

Figure 2: Test Maze 1 with a heatmap showing the smallest number of steps needed to reach each space along with the optimal path from the start to goal positions.

1.3.3 Algorithms and Techniques

The act of first exploring the maze and then finding the optimal path through it are best treated as two separate problems. Both problems, at their root, are search problems. There are a number of algorithms that can be used to solve search problems. These include Djikstra's algorithm [2], A* [3], as well as a host of graph [4] and tree [5] traversal algorithms. The following will explain which, if either, search problem the algorithm would be suitable for.

- **Djikstra's algorithm** is used to find the shortest path between nodes in a graph [2]. It would not, however, help the robot explore the maze in the first place. Once the maze has been fully explored and the forks in the maze are converted into a graph, then Djikstra's algorithm could be used to find the optimal path through the maze.
- __A__ *is an algorithm used for pathfinding and graph traversal [3]. Like Djikstra's algorithm, A* could be used to find the optimal path through the maze once it has been explored, but unlike Djikstra's algorithm, A* could be used to help the robot explore the maze even when the robot's knowledge of the maze is incomplete. The robot could, given whatever it's knowledge of the maze, attempt to move toward the center of the maze. As walls blocking the robot's path are discovered and the robot's knowledge of the maze increases, the algorithm can be used to consistently circumvent the obstacles. This strategy will find a path to the goal, but may not find the optimal path. Once the goal is reached, A* could then be used to reach the unexplored areas within the maze. At face value, this sounds perfect, but may not be very efficient as the robot would likely pass through already explored areas incurring a scoring penalty by taking additional time to explore the maze.
- **Graph traversal algorithms**. There are two types of graph traversal algorithms, depth-first search and breadth-first search [3]. Both types of algorithms could be used to both, though independently, explore the maze as well as find the optimal path once the exploration is complete. Both approaches would complete these tasks, but they would not receive the same score. The breadth-first search, which is a special case of Djikstra's algorithm, would require that the robot continually revisit explored parts of the maze in order to access the different parts of the unexplored boundary. This strategy would result in a very high exploration scoring penalty.

- **Tree traversal algorithms** are a subset of graph traversal algorithms. These traversal algorithms visit each leaf node exactly once [4]. Though this would be fine for finding the optimal path, this strategy fails for exploration for the same reason that the graph traversal algorithms are not suitable for exploration, the robot has to travel to each leaf node resulting in an inefficient exploration.

As we have seen, the majority of these algorithms would be suitable to find the optimal path once the exploration is complete, but are not ideal for the actual exploration of the maze. The other observation is that all of these algorithms, as would be expected, are naive to the specific details of this maze problem. For example, the robot's sensors provide information about more than one space. As we observed in the Data Exploration section of this project, the first sensor reading for the Test Maze 1 gives the robot information about 12 spaces within the maze. This information allows the robot to explore the maze more efficiently than the naive version of the above algorithms would allow. It would be more efficient to taylor one of the search algorithms to use this additional information.

In reality, each sensor reading can collapse the knowledge of multiple spaces simultaneously. This knowledge may allow the robot to know exactly the shape of an unvisited space or at the very least limit the number of possible shapes an unvisited space may take. this knowledge in combination with additional sensor readings may then allow the robot, with all certainty as the sensor readings are perfect, the shape of the unvisited space. The implication of this is that the robot could know the shape of the whole maze without having to visit all of the spaces. The exploration process could be further optimised by determining that the optimal path has been found even with incomplete knowledge of the maze. If the unknown elements of the maze could not result in a more optimal route, then the current optimal path is the path that robot should take. At this point, the robot would maximise its score by terminating the exploration run immediately and proceed to the second run of the maze.

The ideal strategy is to program the robot to keep track of the what is known about the maze, the map, as well as keep track of possible shapes for each space given what has been learnt about the maze so far. A* search can then be used to explore the maze directing the robot to the areas the robot knows least about. This should maximise knowledge of the maze while taking advantage of the fact that the robot can learn about the unexplored areas without visiting them. It should also minimise the number of steps required to learn enough about the maze to find the optimal route. Djikstra's algorithm can then be used to find the optimal path through a graph representation of the maze given what is known.

1.3.4 Benchmark

The benchmark score for a given maze will be the sum of a reasonable path to the goal and a reasonable number of steps to explore the maze given its size. The trick is to define what is 'reasonable'. There is no reason why the optimal path through the maze can not be found as long as the virtual robot completes its exploration of the maze. Any other distance for the final path portion of the benchmark would be entirely arbitrary so the optimal path will be used for the benchmark. For Test Maze 1, that was discussed earlier, the optimal path is 17 steps.

The exploration portion of the benchmark is much harder to define. In order to explore a maze in its entirety, it would be impossible for a robot to traverse each square of the maze without stopping at a square more than once. This means the benchmark for exploration must be more than the size of the maze divided by 30. Test Maze 1 is 12 squares by 12 giving a total size of 144 squares so the lower boundary of the benchmark is 4.8. It is reasonable for a robot to explore the entire maze without, on average, stopping at every square twice. Given a robot could never do better than the benchmark for the final path, it makes sense to choose an exploration benchmark that a robot could improve upon so we will settle on a benchmark exploration score of twice the number of squares in the maze divided by 30. In the case of Test Maze 1, this gives us a benchmark of 9.6.

This leaves us with a total benchmark score of 26.6 for Test Maze 1. Considering the algorithms and techniques discussed in the previous section, the virtual robot should not have any trouble achieving a score lower than this benchmark for Test Maze 1.

1.4 III. Methodology

(approx. 3-5 pages)

1.4.1 Data Preprocessing

The nature of this project results in an absence of data preprocessing. The virtual robot sensor specification and design of the virtual environment have been given as a part of the project definition.

1.4.2 Implementation ******

From the criteria: The process for which metrics, algorithms and techniques were implemented has been thoroughly documented. Complications that occurred during the coding process are discussed in some detail. In addition, student's robot code consistently completes mazes (one learning run and one fast run) within a one thousand time step limit. This includes the three sample mazes provided in the starter code, the three mazes provided by evaluators, and (?). *******

In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process. Questions to ask yourself when writing this section: - *Is it made clear how the algorithms and techniques were implemented with the given datasets or input data? - Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution? - Was there any part of the coding process (e.g., writing complicated functions) that should be documented?*

1.4.3 Refinement ******

From the criteria: The process of improving upon the algorithms and techniques used is clearly documented. Both the initial and final solutions are reported, along with intermediate solutions, if necessary. *******

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section: - *Has an initial solution been found and clearly reported? - Is the process of improvement clearly documented, such as what techniques were used? - Are intermediate and final solutions clearly reported as the process is improved?*

1.5 IV. Results

(approx. 2-3 pages)

1.5.1 Model Evaluation and Validation ******

From the criteria: The final model's qualities — such as parameters — are evaluated in detail. Some type of analysis is used to validate the robustness of the model's solution. *******

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model's solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section: - *Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate? - Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data? - Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results? - Can results found from the model be trusted?*

1.5.2 Justification ******

From the criteria: The final results are compared to the benchmark result or threshold with some type of statistical analysis. Justification is made as to whether the final model and solution is significant enough to have adequately solved the problem. *******

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section: - *Are the final results found stronger than the benchmark result reported earlier? - Have you thoroughly analyzed and discussed the final solution? - Is the final solution significant enough to have solved the problem?*

1.6 V. Conclusion

(approx. 1-2 pages)

1.6.1 Free-Form Visualization ******

From the criteria: Free-Form Visualization: Use this section to come up with your own maze. Your maze should have the same dimensions (12x12, 14x14, or 16x16) and have the goal and starting positions in the same locations as the three example mazes (you can use test_maze_01.txt as a template). Try to make a design that you feel may either reflect the robustness of your robot's algorithm, or amplify a potential issue with the approach you used in your robot implementation. Provide a small discussion of the maze as well. *******

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section: - *Have you visualized a relevant or important quality about the problem, dataset, input data, or results? - Is the visualization thoroughly analyzed and discussed? - If a plot is provided, are the axes, title, and datum clearly defined?*

1.6.2 Reflection ******

From the criteria: Student adequately summarizes the end-to-end problem solution and discusses one or two particular aspects of the project they found interesting or difficult. *******

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section: - *Have you thoroughly summarized the entire process you used for this project? - Were there any interesting aspects of the project? - Were there any difficult aspects of the project? - Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

1.6.3 Improvement ******

From the criteria: Improvement: Consider if the scenario took place in a continuous domain. For example, each square has a unit length, walls are 0.1 units thick, and the robot is a circle of diameter 0.4 units. What modifications might be necessary to your robot's code to handle the added complexity? Are there types of mazes in the continuous domain that could not be solved in the discrete domain? If you have ideas for other extensions to the current project, describe and discuss them here. *******

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section: - *Are there further improvements that could be made on the algorithms or techniques you used in this project?* - *Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?* - *If you used your final solution as the new benchmark, do you think an even better solution exists?*

1.7 References

[1] Wikipedia contributors, "Micromouse," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Micromouse&oldid=709118923 (accessed March 9, 2016).

[2] Wikipedia contributors, "Dijkstra's algorithm," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&oldid=745950368 (accessed October 24, 2016).

[3] Wikipedia contributors, "A* search algorithm," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=A*_search_algorithm&oldid=744637356 (accessed October 16, 2016).

[4] Wikipedia contributors, "Graph traversal," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Graph_traversal&oldid=703421335 (accessed February 5, 2016).

[5] Wikipedia contributors, "Tree traversal," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Tree_traversal&oldid=745817776 (accessed October 23, 2016).

General project criteria:

Quality

Presentation Project report follows a well-organized structure and would be readily understood by its intended audience. Each section is written in a clear, concise and specific manner. Few grammatical and spelling mistakes are present. All resources used to complete the project are cited and referenced.

Functionality Code is formatted neatly with comments that effectively explain complex implementations. Output produces similar results and solutions as to those discussed in the project.

Before submitting, ask yourself. . .

- Does the project report you've written follow a well-organized structure similar to that of the project template?
- Is each section (particularly **Analysis** and **Methodology**) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?
- Would the intended audience of your project be able to understand your analysis, methods, and results?
- Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?
- Are all the resources used for this project correctly cited and referenced?
- Is the code that implements your solution easily readable and properly commented?
- Does the code execute without error and produce results similar to those reported?

```
In [ ]:
```