|  |  |
| --- | --- |
| **Capstone Project: Plot and Navigate a Virtual Maze**  **Machine Learning Engineer Nanodegree** | Xu Shiyan  June 23, 2016 |

# **Definition**

### **Project Overview**

This project aims to solve virtual mazes by designing a robot (in program) such that it can explore and navigate to goal cell inside the mazes. The robot is initially placed at one corner of the maze and allowed to explore it for the first time. The robot is supposed to find an optimal path to the goal based on the map sensed during exploration. At second time, the robot will be placed back to its initial position. It should be able to navigate through the optimal path and reach the goal as fast as possible.

### **Problem Statement**

#### Maze

The virtual maze is a *n* x *n* grid of squares, where *n* is 12 or 14 or 16. The below picture illustrates a corner of a sample maze. Black lines indicate walls that robot cannot pass through. Grey dotted lines indicate open sides of grid cells. A complete maze will have walls all along the perimeter and random walls among the grid cells. The goal cell is a 2 x 2 grid cell at the center with no wall inside. Number inside a grid cell is a numerical way to describe which side of the cell is open and which is close. Let *t*, *r*, *b*, *l* be Boolean (1 or 0) to indicate whether a wall exists at top, right, bottom, left side of a grid cell, where 1 indicates open (no wall) and 0 indicates closed (walled). Then number inside a cell, , will be a sum such that

where *t*, *r*, *b*, *l*, is either 1 or 0.



Figure : Left bottom corner of virtual maze

The bottom-left cell will be the starting cell with top open and other sides walled. Therefore, its cell value .

#### Problem

As mentioned in the overview section, the robot will run twice. At the beginning of each run, robot will be placed at the bottom-left cell and always faces up (face to the top side of cell). Robot cannot pass through walls; it can only go to another cell through an open side, by first rotating -90, 0 or 90 degree then followed by moving forward or backward over 1 to 3 cells. At each time step, robot will receive a set of distance information, indicating which sides of its current cell are open (robot’s front facing side and robot’s left and right sides) so as to determine the next movement.

The problem is to design robot’s next\_move function, which takes in the sensor distance information, to realize these results: during first run, it can gather and store mapping information of the maze (include center goal cell). During the second run, it should navigate to reach center goal cell as fast as possible.

### **Metrics**

During first run, robot must enter goal cell and then choose either to end the run immediately or to continue exploring. During second run, robot should move as fast as possible to enter goal cell. The total time steps for two runs should not exceed 1000. The performance is evaluated using number of time steps used for each of the two runs. Let *t1* be time steps used in the first run, *t2* be time steps used in the second run.

where

Obviously, as the metrics suggest, smaller the , better the performance. Time spend on first run has much less impact to the final performance. It would sometimes help to improve the performance by spending more time to explore and gather more data of the maze. With more data, the robot would be likely to find a better path so that in the second run, it could navigate to goal cell faster.

# **Analysis**

### **Data Exploration**

The robot is able to sense its front, left and right sides. At each time step, robot first senses distances to walls from those sides, and then makes a move, which consist of a rotation and a following movement. Rotation is -90 or 0 or 90 degrees. Movement is an integer in [-3, 3] inclusive, indicates how many cells it can advance or reverse. It advances or reverses until a wall blocks the way.

At time 0, robot will always rotate 0 degree and move forward at least one cell because the initial cell is always open at top side only and robot always initially faces up.

During first run, the robot will always process sensor data to complete the map and will not take forward or backward movement larger than one cell because its belief of how the maze is structured usually changes at next time step, and it should take every opportunity (i.e., every one movement step) to map out the maze as much as possible.

During second run, the robot will not process sensor data any more. Rather, it should have already found out an optimal path to the goal based on the map data it gathered during first run. The job of second run is to navigate to the goal as fast as possible hence there will be sometimes more than one cell advancement at single time step.

#### Example: test\_maze\_01

This is a 12 x 12 maze. The goal cell (2 x 2) has only one half open side on the upper right; this is the only entry to the goal. By observation, there are many paths that can lead to the goal cell. The optimal path takes 17 steps, which is shown in next section. There are also quite a few dead-end cells with only one side open; once robot enters it, a reverse is needed to move out.

### **Exploratory Visualization**

As shown in the pictures below, the initial cell at left bottom corner is fixed in the way that only upper side is open, and the initial heading of the robot is always upward.

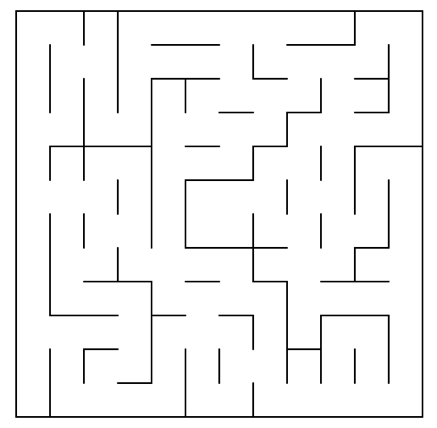


Figure : Sample virtual maze visualization

There are many paths that can lead to goal cell in test\_maze\_01. As shown in pictures below, red path is one possible path to the goal, however, it is not the best optimal path. The green path shown below is the optimal path, which takes 17 steps in total considering it takes 3 or 2 cell advancements at some segments, which shortens the time to reach goal.

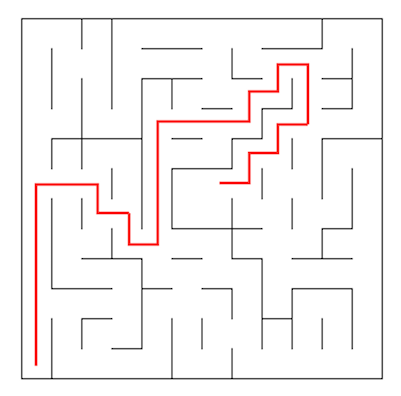


Figure : A path to goal

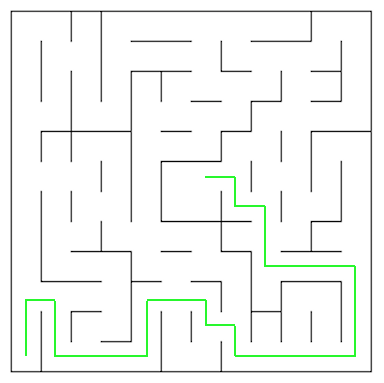


Figure : The optimal path to goal for this maze

### **Algorithms and Techniques**

A\* search algorithm will be used for the first run. The first run aims to map out the maze in preparation for the second run. The first run aims to explore unknown environment and record data such that the environment can be reconstructed in the robot’s “memory” as much as possible. The most important thing to achieve during the first run is to reach the goal cell (which is also a hard rule) so that a path can be found from start cell to it. In the context of this problem, starting cell is always at left bottom corner and goal cell is always at the center of the maze.

A\* search algorithm matches the characteristic of this problem. Firstly, it is a search algorithm that guarantees to find a path to goal if one exists. It expands search path by trying out all possible next movements from one cell and maintains a list of unvisited cells for subsequent expansions. Secondly, A\* prefers to work with a heuristic matrix that guides the search path to expand towards the goal cell more quickly. This problem’s context provides exactly a heuristic for A\* to use, which is the distance to the center goal cells. It can set that heuristic value being the minimum for goal cells and the maximum for cells along the perimeter and decaying as moving closer to the center. Thirdly, to carry out the search within finite time, some constraints of the maze have to be applied, such as boundary of the maze and location of walls that could block movements. These information has been provided as input data in text files.

Dynamic programming will be used for the second run. The robot takes input data from its first run’s mapping results. As observed from maze visualization, there are many variations of the path leading to the goal; as the robot may have chance to take different paths to goal (due to equal evaluations on paths done by A\*), it could map out different part of the maze from time to time. It is not likely to map out the entire maze because heuristic matrix helps A\* algorithm work faster to find the goal by not visiting unnecessary cells. Therefore, since reconstructed map based on first run result could vary from time to time, the robot needs “dynamic” policies to follow during second runs; dynamic programming is a match to this characteristic.

Dynamic programming considers all possible combinations of location and heading of the robot and assigns value to each combination based on minimum cost over all possible next movements. The values propagate from goal location (with 0 value) to other connected cells. No matter how different each time the robot maps out the maze, dynamic programming could always find an optimal policy to follow. Although it has high computation complexity, in this problem’s context, maximum maze size is only 16 x 16, which gives only 1024 combinations of robot’s states (with 4 possible headings). With 16 possible movements at each state (4 directions and maximum 3 unit distances gives 16), any modern computer can easily calculate those values within looping logic. (1024 x 16 = 16384 value calculations per loop).

### **Benchmark**

In this section, you will need to provide a clearly defined benchmark result or threshold for comparing across performances obtained by your solution. The reasoning behind the benchmark (in the case where it is not an established result) should be discussed. Questions to ask yourself when writing this section:

* *Has some result or value been provided that acts as a benchmark for measuring performance?*
* *Is it clear how this result or value was obtained (whether by data or by hypothesis)?*

# **Methodology**

### **Data Preprocessing**

No data preprocessing is needed because the starter code has implemented class and methods (maze.py) to construct the maze environment for robot to explore. The starter code also implements sensor specifications and processing logic (dist\_to\_wall()) to feed the robot during exploration.

### **Implementation**

In this section, all discussions regarding implementation details will be using test\_maze\_01 as example, which has 12 x 12 cells.

During robot initialization (refer to method: \_\_init\_\_(self, maze\_dim)), the first thing to do is setup heuristic matrix. As discussed previously, heuristic helps A\* search find goal more quickly. In this problem, because the maze is fixed to even number size, it is feasible to increase heuristic values radially from the center, which has 0 values for the 2 x 2 area. For test\_maze\_01, the heuristic matrix looks like this:

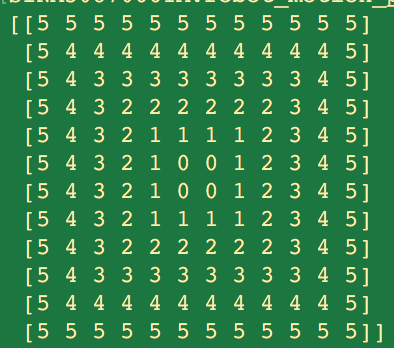


Figure : Heuristic matrix of test\_maze\_01

The heuristic values are certainly admissible for exploration because they are always less than or equal to the cost incurred in reaching goal, given that robot will only advance or reverse one cell during exploration, reason of which has been explained in data exploration section.

There are three other important matrices (all have the same size as maze 12 x 12) to setup during initialization. First is walls, which stores wall information of each cell inferred from sensor data. It is a matrix of dictionary (e.g.: {'u':1,'r':0,'d':0,'l':0}, where 1 means open and 0 means closed.), which is easy to query ability to move from one cell to another given direction string like ‘u’, ‘r’, etc. Second is paths, which stores sequence information of where the robot has visited during exploration. It is not required for program to run successfully but very helpful in debugging and proving A\* search algorithm works as expected. Third is

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?*

### **Refinement**

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?*

# **Results**

*(approximately 2 - 3 pages)*

### **Model Evaluation and Validation**

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?*

### **Justification**

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?*

# **Conclusion**

*(approximately 1 - 2 pages)*

### **Free-Form Visualization**

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?*

### **Reflection**

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?*

### **Improvement**

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?*

Before submitting your report, 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?