Capstone Project Report

Udacity Machine Learning Nanodegree

Paima Marbun
10/27/2021

Robot Motion Planning

Plot and Navigate a Virtual Maze

I. DEFINITION

Project Overview

This project is based on Micromouse competition where a small robot should find autonomously the fastest possible path from predetermined starting node to one of the goals located in the center of an unknown maze. The maze is typically made up of a 16x16 grid of nodes with or without walls on its four edges. The robot can explore and discover the walls in the first run to map out the maze and detect the goals. From the learning of the first run the robot is expected to know the fastest path to one of the goals in the maze center and it attempts to reach it the second run.

Problem Statement

The use case should be simulated by applying virtual robot and virtual maze. The virtual robot should find the fastest path from the starting point in the bottom-left corner of the maze and facing upwards to one of the goals in center of the maze. The maze has n x n dimension of squares with minimum n of 12 and maximum n of 16. Each square or node can have walls on their edges which restrict movement in respective direction. The robot is equipped with three obstacle sensors mounted in the front, left and right side. At any time, robot can move forwards or backwards up to three squares as well rotate clockwise or counterclockwise 90 degree.

The problem of this project is to find an algorithm for the robot as a guide to discover an unknown maze and reaching the goals in the fastest time possible. Several algorithms should be investigated to determine the best optimal one to cope with three sample mazes as per the project requirements. To provide more representative results additional mazes should be taken into consideration for performance evaluation and validation.

Evaluation Metrics

The performance of the robot takes both exploration (run1) and optimization (run2) into consideration and calculated as

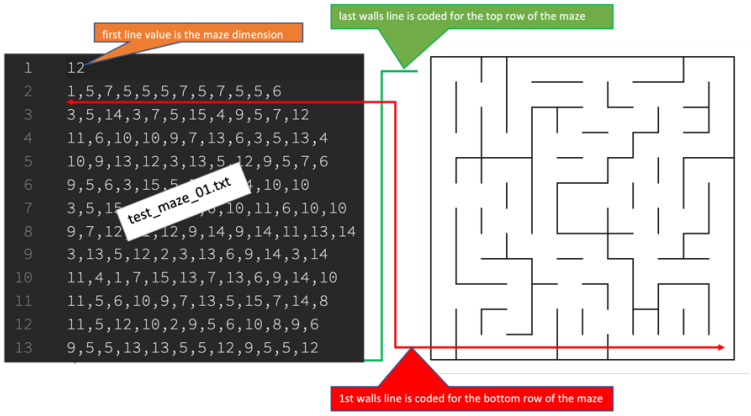
Score = (run1 timesteps / 30) + run2 timesteps

The robot can spend only maximum of one thousand timesteps to complete both run1 and run2 for a given maze. Since the weight for run2 portion is 30 times higher than run1 portion the algorithm will prioritize getting better performance on the optimization run2 compromising higher timesteps caused by increasing exploration in run1.

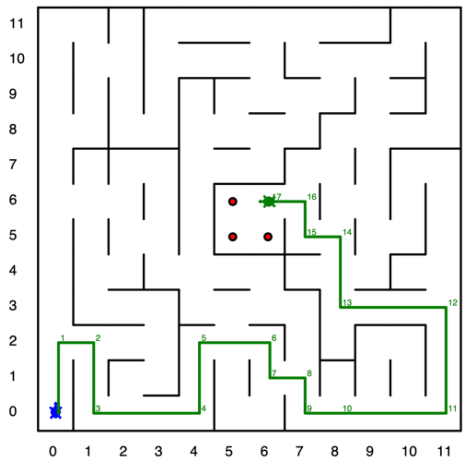
II. Analysis

Data Exploration and Visualization

The only inputs fed into the robot are the maze dimension, start position plus heading, and the four possible goals in the center of a maze. The maze is provided as text file contains comma separated values which outline walls of respective square. The four-bit binary value represents four wall edges on a square. Bit value 0 means edge is open and 1 means edge is closed. The three maze samples for this project are provided by Udacity. Following scheme should illustrate how to create a new maze based on example test_maze_01.txt.



The robot's start position is in the bottom left-hand corner of the maze (coordinate [0,0]) and facing in an upward direction as shown in the right figure. The four goals are centered in the maze marked with red dots. As mentioned earlier the robot can visit a passable node by performing forward or backwards movement of up to three nodes and -90° counterclockwise, 90° clockwise or 0° rotation. The robot should prefer a path utilizing maximal steps of 3 for its movement to a targeted node than a path with many turns to reduce timesteps impacting final score if both paths require same single steps. The run1 can be terminated once the goal is found and the targeted exploration is completed. It then follows run2 to reach the goal in fastest timesteps. The optimal timesteps for the test_maze_01 is 17 as plotted in this figure.



Algorithm and Techniques

There are several proven algorithms for path finding available e.g., the popular A*. They all require some knowledge of the maze to output an optimal path from start to goals. And the best optimal path can certainly be achieved if entire maze is known or 100% explored during the first exploratory run. Thus, the main challenge in this project is to explore the maze as many possible in minimum timesteps to find as many as possible paths to goal so that the second optimization run can achieve optimal path close to the best optimal path popular algorithms would achieve with full knowledge of the maze. Since the robot is allowed to further map out the maze after visiting the goal in the first run two approaches will be performed. The one explores the given maze until particular coverage and the other one explores only until one of the goals found.

For exploratory run1 I came up with following two approaches:

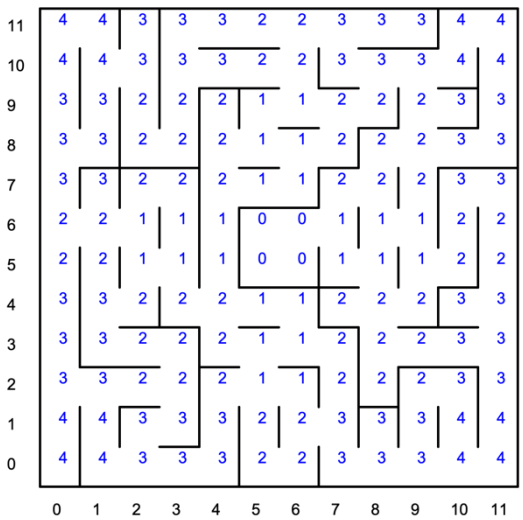
- Short-turn

  With this approach the robot will first explore close neighbors (smallest distance from the robot's actual position) which are unexplored but already detected via sensors reading. A list containing open nodes should guide the robot during the exploration. The node in the list with the smallest distance from the current node should be selected for the next move and if there are several candidates with the same distance then one will be randomly chosen. Already visited nodes should be skipped. This process continues until either the goal is found, or the targeted exploration is reached.

- Heuristic-turn

  The node with having smallest heuristic value plus the distance to the robot's current position will be selected for the next exploration move. The heuristic value for the given maze dimension, start and goals is generated prior the exploration run1. Same termination logic from the Short-turn is also applied here.

For optimization run2 I decided to use A* algorithm. The A* algorithm first examines nodes which are currently having potentially the fastest path to target. The metric or f-cost is calculated from the sum of heuristic value and g-cost.



$$f(x) = g(x) + h(x)$$

The heuristic value $h(x)$ is the theoretical distance of each node to the goal without anticipating any walls as shown in the left figure, calculated for test_maze_01. While $g(x)$ g-cost is the distance from the start to each node in the maze. The node with the smallest f-cost will be prioritized to be expanded further. This loop continues until the goal node is examined. In the final step the predecessor node will be used to backtrack the path until the starting node.

Benchmark Model

The initial benchmark of maximal 1000 timesteps for both run1 and run2 as per the project requirement should be taken as the poorest baseline. To compare run2 performance which is the significant contributor to the final score A* algorithm optimal results should be taken as reference.

With the two references, I will investigate potential solutions as described above and select one candidate having descent performance on three provided mazes and five additional mazes. The main target would be selecting one algorithm which in general also perform on future mazes.

III. Methodology

Data Preprocessing

Maze specification and samples are provided by the project hence no additional data preprocessing is required.

Implementation

The provided starter code will be re-used to and extended. The main functionality will reside in robot.py. The test function will be extended to accommodate various tests and repetition due to randomness and at the end prepare test statistics in a data frame for details analysis. An animation class is added to visualize and follow robot movements required for troubleshooting.

Program workflow:

1. Initialization
   The robot will be initialized with the inputs of maze dimension, start and goal position. Attributes like map and heuristic table are then populated accordingly.

2. Interpreting Sensor Reading
   In each time-step the robot receives sensor information about the distance to wall from its current position in the direction of respective sensor, left, upwards and right. The robot then interprets how many and which nodes are permissible.

3. Replicating Maze
   The maze table with the dimension of the maze is initialized with all zero-value meaning all nodes are initially surrounded by walls. Based on workflow 2 results each actual detected walls of respective and adjacent nodes are updated accordingly. Walls surrounding node are fully uncovered once node is entered hence maze update for such nodes can be skipped.

4. Updating Robot Attributes and Internal States
   At this point the robot is aware of newly detected permissible nodes and append them to a list containing actual unexplored nodes for the next step to assess. Moreover, for Heuristic turn and A* algorithm support g-cost and f-cost will be calculated and populated in each table.

5. Executing Exploratory Run1
   Main result of this step is to identify next possible move of the robot to explore the maze. At first distance between each node in the open list and current robot position must be calculated and depending on actual chosen algorithm for the Short-turn the smallest distance between two nodes will be selected while for the Heuristic-turn the smallest value of distance between two nodes plus the heuristic value of targeted node. Each time calculation of distance between two nodes is required A* search will be triggered to obtain the optimal path. The robot will trigger single or consecutive movement according to the path to enter the targeted node. During the exploration robot must keep tracking of its location and heading, updating the table of node status indicating closed/undetected, open/detected, or done/visited, and updating the table of number of how frequent node is visited.

6. Terminating Exploratory Run1
   The exploratory run1 will be terminated depending on actual chosen algorithm either once the goal is found or until exploration achieves targeted coverage after entering the goal. The Robot will reset its location back to the original start position and heading. To signal this event to calling (test-) unit "Reset" and "Reset" are returned instead of valid movement and rotation.

7. Executing A* search For Optimization Run2
   After gathering sufficient or full knowledge of the maze Run2 should now find the best optimal path from the starting position to the goal using A* search algorithm as described earlier. A dynamic list containing open nodes for expansion will be maintained in the process. New node will be added to the list as result of predecessor expansion. Already expanded node will be recorded in the look-up table and removed from the running list. To identify the next promising node for expansion, distance between each candidate in the list and the robot's current position plus heuristic value must be calculated. The node with the lowest f-cost or sum of both values will be further expanded. If multiple candidates exist, one will be randomly chosen. The expansion continues until the goal node is expanded and backtrack the path to the starting node to compile optimal path. Finally, the robot triggers consecutive movement and rotation until the goal node entered.

Classes / Modules:

- **Robot class** (robot.py). This is the core of the robot controller.
  Main Functions:

  __init__( )
  This will initialize the needed attributes and states as described in the workflow 1.
  Input parameters: Maze dimension, selected algorithm for run1

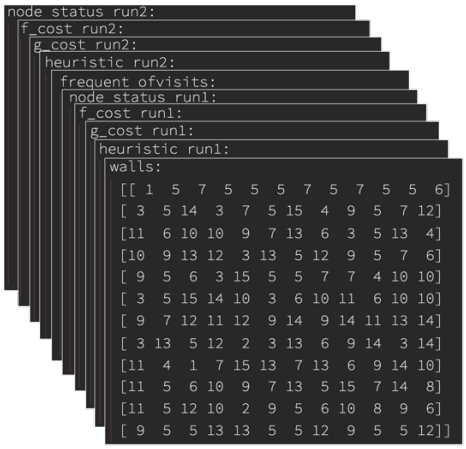
  fill_map_heuristic( )
  This routine will do the tasks defined in the program workflow 2 – 4.
  Input parameters: sensors reading

  next_move( )
  This routine will do the tasks defined in the program workflow 5 – 6.
  Input parameters: sensors reading
  Return parameters: -90°/0°/90° rotation and forward/backward movement

  find_best_path( )
  This routine will do the tasks defined in the program workflow 7.
  Input parameters: start node, heading, and goals node
  Return parameters: -90°/0°/90° rotation and forward/backward movement.

Main Attributes:

- Map consisting of ten stacked two-dimensional numpy array as shown below to store all the necessary information like walls, heuristic value, and other look-up tables.



- Dictionaries
    - *dir_sensors* for translating given heading to next possible heading
    - *dir_move* for translating given heading to adjacent node distance
    - *dir_reserve* for translating given heading to reversed heading
    - *dir_rotation* for translating targeted heading to rotation of the robot
    - *dir_int* for translating edge position to a binary number of walls

- Running list storing open nodes for exploration in run1 and for expansion in run2

- Maze class (maze.py).
  The class is storing maze information required for test purpose.

  Main Functions:

  __init__( )
  This populates walls value in two-dimensional array.
  Input parameter: maze specification text file

  is_permisible( )
  This checks if adjacent node is passable in given direction
  Input parameters: actual node and direction to adjacent node
  Return parameter: True if passable otherwise False

  dist_to_wall( )
  This gives number of open nodes to nearest wall in the given direction
  Input parameters: actual node and direction
  Return parameter: number of passable nodes

  Main Attributes:

  - Two-dimensional array to store walls value and dimension of the maze

- Mazeanim class (mazeanim.py).
  This is an extension of the provided showmaze.py module to animate robot movements using turtle module. It should help troubleshooting to easily follow robot movement through a visualization.

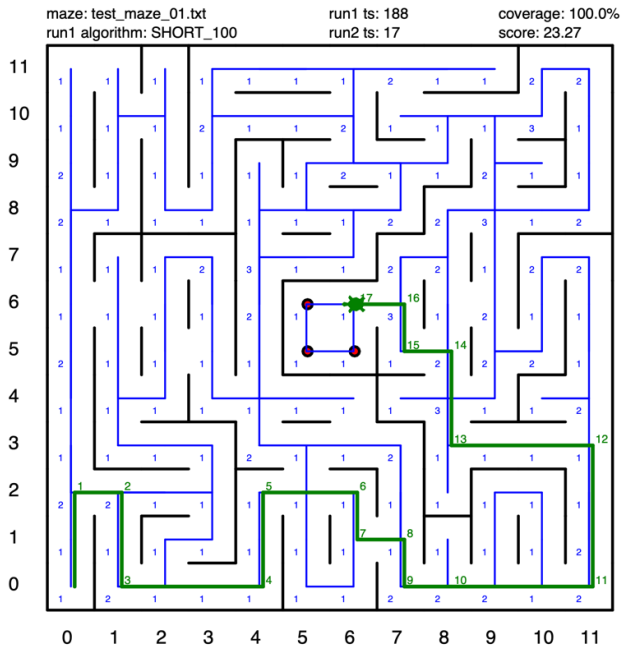
  Main Functions:

  showmaze( )
  This plots the maze grids.
  Input parameter: -
  Return parameter: -

  plot_move( )
  This plots each robot movement in both run1 and run2 with different colors.
  Input parameters: run1 or run2
  Return parameter: -

  plot_summary( )
  This plots the summary of the exploration and optimization run.
  Input parameter: dictionary contains summary
  Return parameter: -

  The figure below is taken from an animation run provided by the class. It illustrates the robot movement during exploratory run1 (plotted with blue line) and optimization run2 (plotted with green line). Each blue number in the center of each node indicates how often node is visited by the robot and the green number show timesteps from the start to the goal spent by the robot. Furthermore, summary of both run and the score are also displayed.



- Algorithm Tester module (alg_tester.py)
  This is an extension of provided tester.py module to accommodate various run1 algorithms tests and repetition test as random selection is a part included in the solutions. All test results should be cumulated in a panda dataframe for a statistical analysis.

Refinement

My initial solution for run1 algorithm was a random movement. It records how often robot visits nodes in the maze. This value is used as the rule for the robot to prefer to move to less visited node. Node with dead-end or path to dead-end will be recorded to avoid an unnecessary revisit. But the results conducted on several runs are not that satisfying. E.g., to uncover complete maze_01 the robot needs in average 600 timesteps. Then I realized that if only adjacent nodes are considered for the next move and ignoring other far passable nodes that either already visited or detected by the sensors are not effective enough to limit the repetitive move and explore more nodes at the same time. The next improvement was addressing this issue. In addition, the robot should now prefer passable detected node but unexplored for the next move. Repetitive move to dead-end or path to dead-and nodes are therefore also automatically avoided. With this improvement timesteps required to explore all nodes in maze_01 go down to 200 timesteps.

For debugging purpose print statements were sufficient in the early implementation phase. But with increasing complexity and solution variants analyzing print output is becoming cumbersome and time-consuming. To reduce the effort logging is added as first step followed by introducing animation in the second step to visualize robot's movement in the maze.

IV. Results

Model evaluation and validation

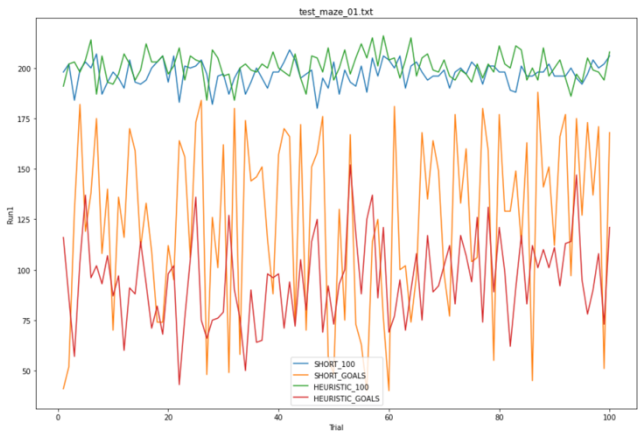
To recap what is described in **Algorithm and Techniques** following potential solutions for run1 will be evaluated and validated.

- **SHORT_X**: Performing Short-turn until the goal is found and x % coverage is achieved.
- **SHORT_GOALS**: Performing Short-turn until the goal is found.
- **HEURISTIC_X**: Performing Heuristic-turn until the goal is found and x% coverage is achieved.
- **HEURISTIC_GOALS**: Performing Heuristic-turn until the goal found.

Before evaluating the overall results, let me go in details how each of algorithms performs first on test_maze_01 (100 trials per algorithm).

Run1 Timesteps

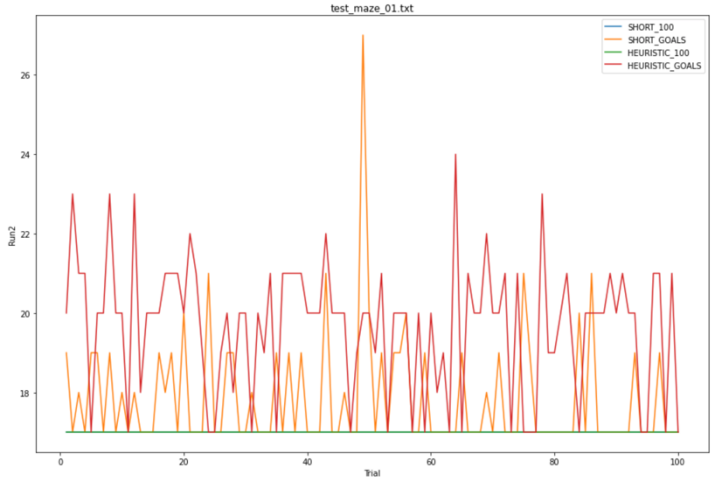
Alg	count	mean	std	min	50%	75%	95%	max
HEURISTIC_100	100.0	200.82	6.757532	184.0	200.0	205.00	212.00	216.0
HEURISTIC_GOALS	100.0	95.05	21.496241	43.0	93.5	110.25	131.25	152.0
SHORT_100	100.0	197.09	5.918802	180.0	198.0	201.00	206.00	209.0
SHORT_GOALS	100.0	124.28	43.462298	40.0	129.5	164.00	180.00	188.0



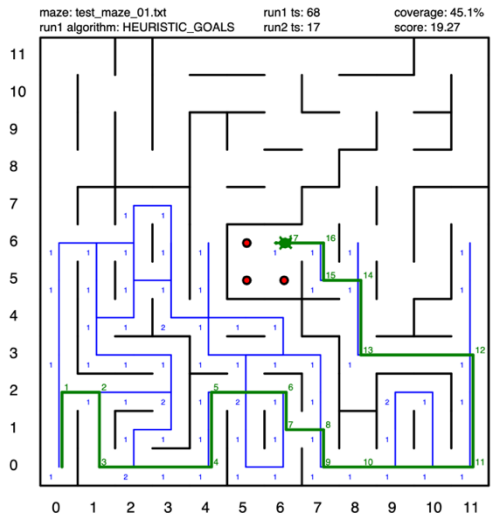
Both algorithm **SHORT_100** and **HEURISTIC_100** shows similar results with comparable standard deviation and quantiles. While for finding the goal **HEURISTIC_GOALS** most of the time outperforms **SHORT_GOALS** and this is explained by the fact that **HEURISTIC_GOALS** is tasked to find the shortest path to the goal using heuristic value while the guidance for **SHORT_GOALS** is to explore the maze and find the goal at some point in time.

Run2 Timesteps

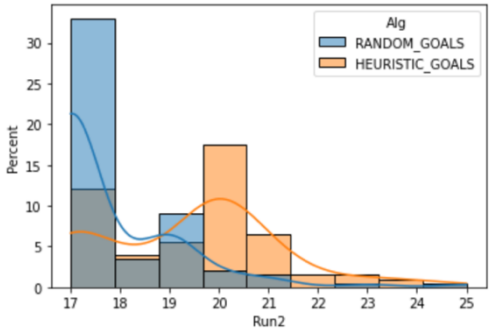
Alg	count	mean	std	min	50%	75%	95%	max
HEURISTIC_100	100.0	17.00	0.000000	17.0	17.0	17.0	17.00	17.0
HEURISTIC_GOALS	100.0	19.65	1.689988	17.0	20.0	21.0	22.05	24.0
SHORT_100	100.0	17.00	0.000000	17.0	17.0	17.0	17.00	17.0
SHORT_GOALS	100.0	17.85	1.472818	17.0	17.0	19.0	20.05	27.0



Looking to the above results algorithms with full knowledge of the maze will find the best optimal path through A* search algorithm used in run2. **SHORT_FULL** and **HEURISTIC_FULL** find same optimal path of 17 timesteps as A* search will produce consistent results given same knowledge of the maze. While results for both **SHORT_GOALS** and **HEURISTIC_GOALS** fluctuate depending on exploration coverage achieved in run1 and the in some trials optimal path of 17 timesteps can be also achieved with low exploration coverage as demonstrated by the two figures below.

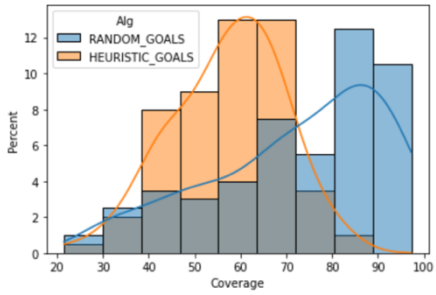


Alg	count	mean	std	min	50%	75%	95%	max
HEURISTIC_GOALS	100.0	19.42	1.837763	17.0	20.0	20.0	23.00	25.0
RANDOM_GOALS	100.0	17.81	1.412033	17.0	17.0	19.0	20.05	25.0



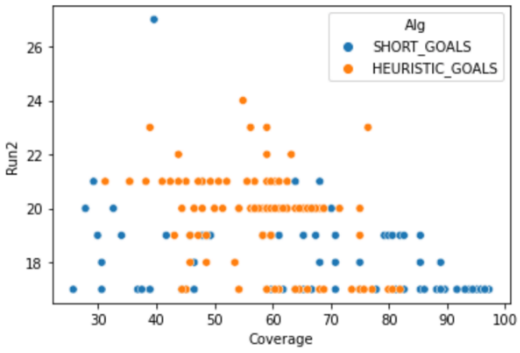
Run1 Coverage

Alg	count	mean	std	min	50%	75%	95%	max
HEURISTIC_GOALS	100.0	57.2089	12.224436	21.53	57.985	65.4525	75.725	84.03
RANDOM_GOALS	100.0	72.5761	18.897401	29.17	76.385	87.5000	95.830	97.22



From the coverage point of view, it can be seen that **HEURISTIC_GOALS** can find the goal most of the time with less timesteps compared to **SHORT_GOALS** as expected. The coverage for **SHORT_100** and **HEURISTIC_100** is not shown as it is always 100% targeted over many trials.

Coverage vs Run2 timesteps



Next question would be if we can find an average minimum of coverage required yet it can get optimal path as we would get with the full knowledge of the maze. But there is no clear pattern emerged. Optimal path can be achieved randomly with coverage range from 30% and to 100%. It really depends on how the Robot expands the exploration. In some trials the robot explores from the start already the right area leading to the goal with best optimal path.

For further evaluation I have added three additional coverage level (70%, 80% and 90%) to both solutions to address above question. As per result below run2 timesteps certainly (95% confidence interval) reduced with increased coverage.

```
Run2 timesteps for maze_01
                count   mean       std   min   50%   75%   95%   max
Alg
HEURISTIC_070   100.0  18.75  1.131505  17.0  19.0  19.0  20.0  23.0
HEURISTIC_080   100.0  18.12  0.945804  17.0  18.0  19.0  19.0  20.0
HEURISTIC_090   100.0  17.38  0.721670  17.0  17.0  17.0  19.0  19.0
HEURISTIC_100   100.0  17.00  0.000000  17.0  17.0  17.0  17.0  17.0
SHORT_070       100.0  17.68  0.993718  17.0  17.0  19.0  19.0  21.0
SHORT_080       100.0  17.35  0.715979  17.0  17.0  17.0  19.0  19.0
SHORT_090       100.0  17.14  0.492776  17.0  17.0  17.0  19.0  19.0
SHORT_100       100.0  17.00  0.000000  17.0  17.0  17.0  17.0  17.0
```

Score

Below results summarize the score tendency of each algorithm tested 100 times on maze_01, maze_02 and maze_03.

The best score of 18.3 is recorded in one of the maze_01 trials with **SHORT_80** while the worst score is 28.9. I take the 95% confidence interval as selection criteria so the **SHORT_80** with the score below 23.7 in 95% of the cases is identified to be the best solution for the maze_01.

```
Score for maze_01
                 count    mean       std   min    50%     75%     95%   max
Alg
HEURISTIC_070    100.0  22.869  1.098548  20.8  22.90  23.225  24.305  27.9
HEURISTIC_080    100.0  22.858  0.973817  21.3  22.90  23.700  24.000  24.9
HEURISTIC_090    100.0  22.956  0.775290  22.1  22.70  23.000  24.605  24.8
HEURISTIC_100    100.0  23.694  0.229545  23.1  23.70  23.800  24.100  24.2
HEURISTIC_GOALS  100.0  22.822  1.542959  19.3  23.05  23.725  25.115  27.4
SHORT_070        100.0  22.333  1.000410  20.7  22.40  23.000  23.905  25.0
SHORT_080        100.0  22.460  0.765282  21.2  22.40  23.100  23.700  23.9
SHORT_090        100.0  22.651  0.561113  22.0  22.50  22.800  24.200  24.5
SHORT_100        100.0  23.570  0.208167  23.0  23.60  23.700  23.900  24.0
SHORT_GOALS      100.0  21.990  1.497506  18.3  22.30  22.825  23.800  28.9
```

On maze 02 the Short-turn algorithms perform overall better than Heuristic-turn. **SHORT_90** with the 95% CI score of 30.7 should be selected.

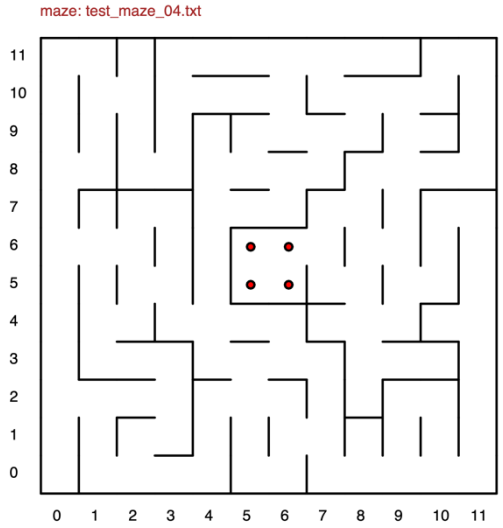
```
Score for maze_02
                 count    mean       std   min    50%     75%     95%   max
Alg
HEURISTIC_070    100.0  30.210  3.158618  27.5  28.50  32.700  36.900  37.4
HEURISTIC_080    100.0  29.855  1.693384  28.1  29.20  29.750  33.705  33.8
HEURISTIC_090    100.0  29.980  0.809290  29.0  29.80  30.300  31.005  33.6
HEURISTIC_100    100.0  31.080  0.245772  30.5  31.10  31.200  31.500  31.9
HEURISTIC_GOALS  100.0  31.332  3.001511  25.5  31.55  32.900  36.300  37.8
SHORT_070        100.0  29.581  1.630002  27.1  29.60  30.925  32.200  32.6
SHORT_080        100.0  29.460  1.111555  28.0  29.05  30.100  31.155  33.4
SHORT_090        100.0  29.709  0.697484  28.7  29.50  30.000  30.705  33.8
SHORT_100        100.0  31.018  0.281547  30.3  31.00  31.125  31.505  31.8
SHORT_GOALS      100.0  29.521  1.894559  24.1  29.85  30.700  31.520  36.8
```

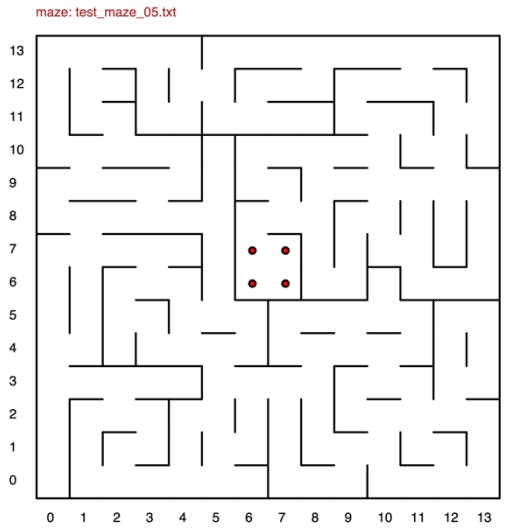
On maze_03 **SHORT_90** produces the best score of 35.4.

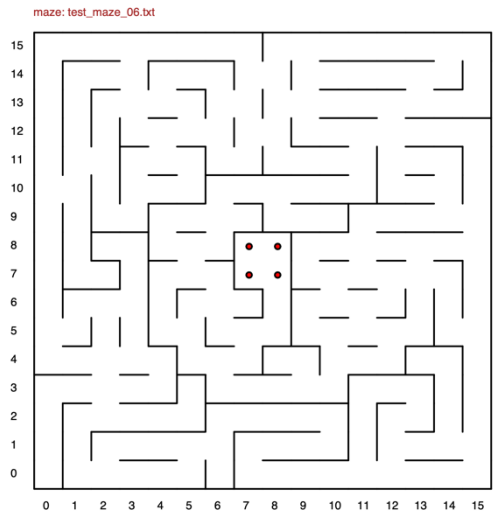
```
Score for maze_03
                 count    mean       std   min    50%     75%     95%   max
Alg
HEURISTIC_070    100.0  33.836  2.123201  31.8  32.50  35.300  38.400  38.6
HEURISTIC_080    100.0  34.467  1.828785  32.6  33.50  36.300  37.410  39.6
HEURISTIC_090    100.0  34.828  0.884785  34.1  34.50  34.900  37.605  38.1
HEURISTIC_100    100.0  36.533  0.287467  35.8  36.50  36.700  37.000  37.1
HEURISTIC_GOALS  100.0  33.459  3.344076  27.6  32.15  34.100  40.100  44.7
SHORT_070        100.0  33.482  1.865496  31.5  32.70  35.000  35.915  42.6
SHORT_080        100.0  33.857  1.304402  32.6  33.40  33.900  36.315  39.4
SHORT_090        100.0  34.661  0.439902  33.8  34.60  35.000  35.400  35.8
SHORT_100        100.0  36.764  0.385997  36.1  36.70  37.025  37.500  37.7
SHORT_GOALS      100.0  32.783  3.010019  27.3  32.40  33.900  38.725  44.3
```

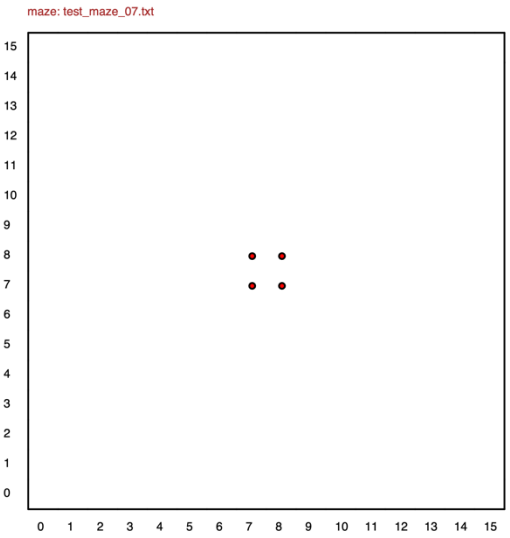
Free-Form Visualization

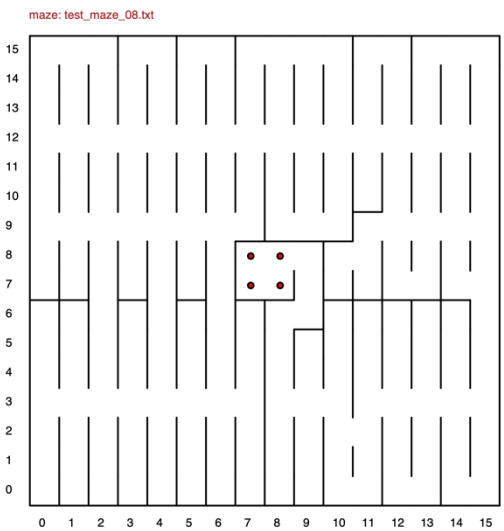
To obtain more representative results I have created additional five mazes as shown below. The maze_04, maze_05 and maze_06 are created based on existing maze_01, maze_02 and maze_03 with slight modifications. The maze_06 should represent a simple maze with no walls at all while maze_06 has many dead-ends and loops.











Justification

Following are the overall results recorded from 8000 random tests (10 algorithms x 8 mazes x 100 attempts). The score for each algorithm and maze is the 95th percentile score which represents the points at which 5% of the score might be worse than the 95th percentile score.

Maze	maze_01	maze_02	maze_03	maze_04	maze_05	maze_06	maze_07	maze_08
Run1 algorithm								
HEURISTIC_070	24.332000	36.901500	38.400000	27.611500	38.675000	40.63200	12.571500	34.232000
HEURISTIC_080	24.001500	33.732000	37.381500	28.103500	38.671500	41.63200	13.570000	34.306500
HEURISTIC_090	24.603500	31.003500	37.578000	25.883000	35.403500	38.41150	14.532000	34.432000
HEURISTIC_100	24.070000	31.470000	37.001500	26.132000	35.801500	37.00000	15.801500	35.478000
HEURISTIC_GOALS	25.086500	36.300000	40.071500	27.803500	38.432000	44.24000	6.230000	34.175000
SHORT_070	23.903500	32.170000	35.942000	27.230000	35.868500	40.70000	12.171500	34.833500
SHORT_080	23.730000	31.156500	36.345000	26.382000	35.615000	41.08800	13.071500	34.510000
SHORT_090	24.171500	30.732000	35.371500	25.637000	34.953500	37.45150	14.100000	34.208500
SHORT_100	23.870000	31.532000	37.501500	26.070000	35.870000	37.47300	15.500000	35.332000
SHORT_GOALS	23.771500	31.548500	38.755000	27.010000	35.523500	41.11850	6.800000	34.608500
Standard Deviation	0.422084	2.242878	1.395055	0.878071	1.482848	2.31811	3.332231	0.465683

Above result shows that each algorithm performs differently on a given maze. On maze_01 and maze_08 all algorithms produce comparable score indicated by the low standard deviation of 0.4 while the performance on maze_07 vary greatly. Since there is not a single algorithm consistently outperforms the others in each maze specification following ranking is created based on cumulative ranking on each maze, see below table.

Maze	maze_01	maze_02	maze_03	maze_04	maze_05	maze_06	maze_07	maze_08	ranking score
Run1 algorithm									
SHORT_090	7.0	1.0	1.0	1.0	1.0	2.0	7.0	2.0	22.0
SHORT_080	1.0	3.0	3.0	5.0	4.0	7.0	5.0	6.0	34.0
HEURISTIC_090	9.0	2.0	7.0	2.0	2.0	4.0	8.0	5.0	39.0
SHORT_070	4.0	7.0	2.0	7.0	6.0	6.0	3.0	8.0	43.0
SHORT_GOALS	2.0	6.0	9.0	6.0	3.0	8.0	2.0	7.0	43.0
HEURISTIC_100	6.0	4.0	4.0	4.0	5.0	1.0	10.0	10.0	44.0
SHORT_100	3.0	5.0	6.0	3.0	7.0	3.0	9.0	9.0	45.0
HEURISTIC_070	8.0	10.0	8.0	8.0	10.0	5.0	4.0	3.0	56.0
HEURISTIC_080	5.0	8.0	5.0	10.0	9.0	9.0	6.0	4.0	56.0
HEURISTIC_GOALS	10.0	9.0	10.0	9.0	8.0	10.0	1.0	1.0	58.0

I nominate **SHORT_90** as the best performing run1 algorithm of all investigated algorithms in this project as **SHORT_90** ranks first in four mazes and second in two mazes. Though it ranks low on maze_01, it may not that significant since other perform in almost the same range. On mazes having few or no walls like maze_08 the performance is indeed low, but we can assume majority of future mazes would have some walls.

As a final conclusion, the solution outperforms the benchmark model of 1000 timesteps, see below results. Moreover, the optimal path in four mazes can be achieved.

	maze_01	maze_02	maze_03	maze_04	maze_05	maze_06	maze_07	maze_08
Run1 timesteps	183	254	311	191	259	345	243	322
Run2 timesteps	19	23	25	20	26	25	6	24
Total timesteps	202	277	336	211	285	370	249	346
A* optimal path	17	22	25	19	26	25	6	23

Sources:

Udacity Project Files Link:

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

Udacity starter code:

https://www.google.com/url?q=https://drive.google.com/open?id%3D0B9Yf01UaIbUgQ2tjRHhKZGlHSzQ&sa=D&source=editors&ust=1633951177995000&usg=AOvVaw20wxDnXySWm08F-g3vMcnc

Example report:

https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-3.pdf

Micromouse Wikipedia:

https://en.wikipedia.org/wiki/Micromouse

A* Search Algorithm:

https://www.geeksforgeeks.org/a-search-algorithm/

https://en.wikipedia.org/wiki/A*_search_algorithm

A* Pathfinding (E01: algorithm explanation) Video:

https://youtu.be/-L-WgKMFuhE