UDACITY

Machine Learning Nanodegree
Plot and Navigate a Virtual Maze
Capstone Project Report

Prepared by: Clive R. Cadogan

Sunday, October 29, 2017

I. Definition

Project Overview

This project is based on Artificial Intelligence Pathfinding. Pathfinding or pathing is the plotting, by a computer application, of the shortest route between two points[1]. Robot navigation is a generalization of the route-finding problem This project takes inspiration from Micromouse competitions, wherein a robot mouse is tasked with plotting a path from a corner of the maze to its center. The robot mouse may make multiple runs in a given maze. In the first run, the robot mouse tries to map out the maze to not only find the center, but also figure out the best paths to the center. In subsequent runs, the robot mouse attempts to reach the center in the fastest time possible, using what it has previously learned.

The approach may also be applicable to path planning for self driving cars that need to plan the optimum route to the desired destination and other planning challenges.

This project will attempt to utilize A* and Breath First search algorithms to find the optimal path through a maze.

The A* search algorithm is widely used in pathfinding and graph traversal, the process of plotting an efficiently directed path between multiple points, called nodes. In 1968, AI researcher Nils Nilsson was trying to improve the path planning done by Shakey the Robot, a prototype robot that could navigate through a room containing obstacles. It is an extension of Edsger Dijkstra's 1959 algorithm. A* achieves better performance by using heuristics to guide its search[2].

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbor nodes first, before moving to the next level neighbours. BFS and its application in finding connected components of graphs were invented in 1945 by Konrad Zuse. It was reinvented in 1959 by E. F. Moore, who used it to find the shortest path out of a maze,[3][4] and discovered independently by C. Y. Lee as a wire routing algorithm[3].

NB. The test mazes for this project were provided by udacity as part of the started package for this project.

[1] "Pathfinding." Wikipedia, Wikimedia Foundation, 6 Nov. 2017, en.wikipedia.org/wiki/Pathfinding.

[2] "A* search algorithm." Wikipedia, Wikimedia Foundation, 5 Nov. 2017, en.wikipedia.org/wiki/A*_search_algorithm.

[3] "Breadth-First search." Wikipedia, Wikimedia Foundation, 3 Nov. 2017, en.wikipedia.org/wiki/Breadth-first_search.

Problem Statement

To search and determine the best path towards a desired location. This project will utilize search algorithms to help a robot navigate the fastest way through a maze. This may be useful solving path planning problems such as in self driving cars and other real world robots that need to navigate the real world to accomplish a goal.

The robot will first explore the maze tho collect data that may be used to compute the optimum path. Once the optimum path is determined by the algorithm the robot must be able achieve the goal within the required time limit.

This project will attempt to utilize A* and Breath First search to find the optimal path through a maze. These are all approaches that have been successfully applied to pathfinding problems. Breadth-first and A* search algorithms were chosen because they are commonly used on pathfinding problems[4].

Metrics

The score received for successfully completing the game based on number of time steps required to execute the second run, plus one thirtieth the number of time steps required to execute the first run. In this case a lower score is better. This metric was chosen to serve as a weighted measure of how effective the algorithms is at exploring the maze and finding an optimal path. NB. A weight is given to finding an optimal path.

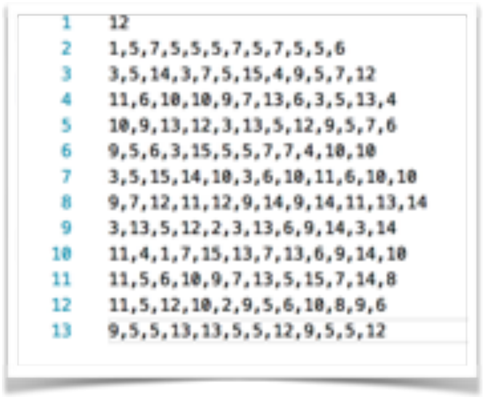
[4] "Pathfinding." Wikipedia, Wikimedia Foundation, 6 Nov. 2017, en.wikipedia.org/wiki/Pathfinding.

II. Analysis

Data Exploration and Visualization

All the mazes are squares and the start location for the agent is always at the bottom left corners (0, 0). In the center of the grid is the goal room consisting of a 2 x 2 square.

Mazes are provided to the system via text file. On the first line of the text file is a number describing the number of squares on each dimension of the maze n. On the following n lines, there will be n comma-delimited numbers describing which edges of the square are open to movement. 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 corresponds 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 10 means that a square is open on the left and right, with walls on top and bottom (0*1 + 1*2 + 0*4 + 1*8 = 10). Note that, due to array indexing, the first data row in the text file corresponds with the leftmost column in the maze, its first element being the starting square (bottom-left) corner of the maze. Kindly see sample of 12 X 12 Maze Raw Data in the figure below.



```
12
1,5,7,5,5,5,7,5,7,5,5,6
3,5,14,3,7,5,15,4,9,5,7,12
11,6,10,10,9,7,13,6,3,5,13,4
10,9,13,12,3,13,5,12,9,5,7,6
9,5,6,3,15,5,5,7,7,4,10,10
3,5,15,14,10,3,6,10,11,6,10,10
9,7,12,11,12,9,14,9,14,11,13,14
3,13,5,12,2,3,13,6,9,14,3,14
11,4,1,7,15,13,7,13,6,9,14,10
11,5,6,10,9,7,13,5,15,7,14,8
11,5,12,10,2,9,5,6,10,8,9,6
9,5,5,13,13,5,5,12,9,5,5,12
```

Figure 1. :12 X 12 Maze Raw Data

The robot has three obstacle sensors, mounted on the front of the robot, its right side, and its left side. Obstacle sensors detect the number of open squares in the direction of the sensor.

It was decided to limit the robots movement to one step if possible for each time step. During each time step the robot may choose to rotate +/-90 degrees then move forward one step.

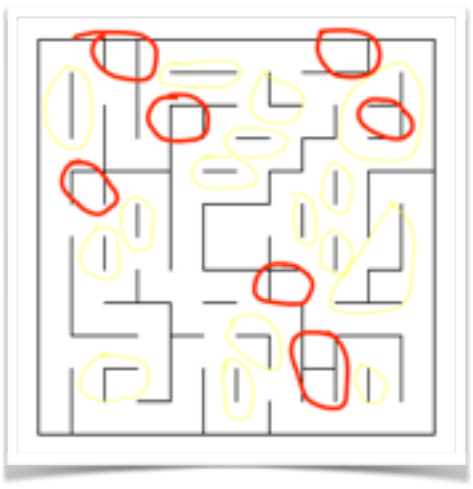


Figure 2. : 12 X 12 maze's Dead-end cells marked in read and possible loops in yellow to be avoided

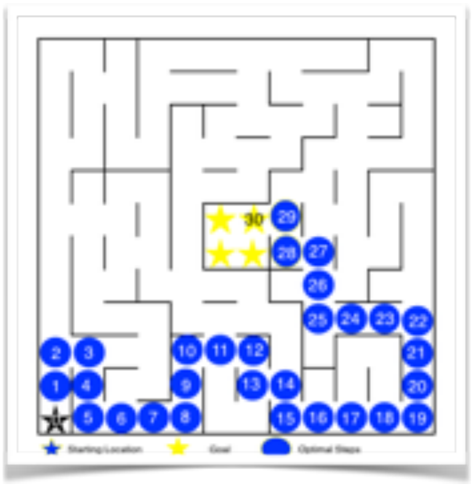


Figure 3. : 12 X 12 maze's Ideal Optimal Path.

It is assumed that the sensors have 100% accuracy however, if the step tries to take the robot into a wall the robot stays there.

The shortest path to the goal for maze one is 30 single steps. Despite the robot being able to take a maximum of three steps it was decided to utilize only steps of one in the desired direction once permitted by absence of a blocking wall in order to more thoroughly explore the maze.

The robot will need to find a path that avoids loops and dead-ends. Hence these locations will be avoided during the second run by utilizing the optimal path found to navigate through the maze to the goal state.

Algorithms and Techniques

The problem the project addresses is commonly solved with Artificial Intelligence Pathfinding approaches. Pathfinding is the plotting, by a computer application, of the shortest route between two points[5]. This project will attempt to utilize A* and Breath First search algorithms to find the optimal path through a maze.

The A* search algorithm is widely used in pathfinding and graph traversal, the process of plotting an efficiently directed path between multiple points, called nodes. Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute (now SRI International) first described the algorithm in 1968. It is an extension of Edsger Dijkstra's 1959 algorithm. A* achieves better performance by using heuristics to guide its search[6]. A* evaluate nodes by combining g(n), the cost to reach the node n, and h(n), the estimated cost the cost to get from the node to the goal: $f(n) = g(n) + h(n)$[7].

Estimating cost from the node to the goal: The heuristic function h(n) tells A* an estimate of the minimum cost from any vertex n to the goal[8]. It's important to choose a good heuristic function. The heuristic is used to control A*'s behavior for example if h(n) is sometimes greater than the the cost of moving from the node n to the goal, then A* is not guaranteed to find a shortest path, but it can run faster. In this project due to the time constraints exploration will need to be fast enough in order to and as a result will need to compromise on finding optimal path particularly with the larger mazes. The heuristic on square grid grid where you can move in 4 directions should be D times the manhattan distance: D * (abs(node.x - goal.x) + abs(node.y - goal.y)). Where the hyper-parameter D can be used scale the estimated cost up to speed up exploration in order to comply with time constraints. Computing path costs to reach the node: Because paths with many turns are more likely to be longer therefore steps with turns will be penalized by applying a weight factor (hyper-parameter) that will cause the path incur additional costs for turns.

NB. The A* search algorithm will only find the optimal path if the heuristic used to estimate the distance the current node and the goal is admissible, meaning that it never overestimates the actual cost to get to the nearest goal node[9]. In this project for some mazes in order to comply with the time constraints the estimate to the goal may need to be scaled to speed up exploration which may compromise finding the optimal path.

[5] "Pathfinding." Wikipedia, Wikimedia Foundation, 6 Nov. 2017, en.wikipedia.org/wiki/Pathfinding.

[6] "A* search algorithm." Wikipedia, Wikimedia Foundation, 5 Nov. 2017, en.wikipedia.org/wiki/A*_search_algorithm.

[7] Russell, Stuart J., and Peter Norvig. Artificial intelligence: a modern approach. Pearson, 2016.

[8] "Introduction to A*   From Amit's Thoughts on Pathfinding." Introduction to A*, theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html.

[9] A* search algorithm." Wikipedia, Wikimedia Foundation, 5 Nov. 2017, en.wikipedia.org/wiki/A*_search_algorithm.

BFS and its application in finding connected components of graphs were invented in 1945 by Konrad Zuse. It was reinvented in 1959 by E. F. Moore, who used it to find the shortest path out of a maze,[3][4] and discovered independently by C. Y. Lee as a wire routing algorithm[10]. Breadth First Search is an considered an uninformed search strategy[11]. It is used for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbor nodes first, before moving to the next level neighbours. In general, all nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded. Breadth-first search chooses the shallowest unexpanded node to expand first. It also discards any new path to a state already in the frontier or explored set since such a path must be attest as deeps any already found. Thus, it always has the shallowest path to every node on the frontier.

Breadth-first and A* search algorithms were chosen because they are commonly used on pathfinding problems[12].

Benchmark

The results from the Breath First Search Algorithm will be utilized as benchmark for the proposed A* algorithm which should be able find an optimal solution. The results of the two algorithms will be compared expecting improvements from the more advanced A* technique.

[10] “Breadth-First search.” Wikipedia, Wikimedia Foundation, 3 Nov. 2017, en.wikipedia.org/wiki/Breadth-first_search.

[11] Russell, Stuart J., and Peter Norvig. Artificial intelligence: a modern approach. Pearson, 2016.

[12] “Pathfinding.” Wikipedia, Wikimedia Foundation, 6 Nov. 2017, en.wikipedia.org/wiki/Pathfinding.

III. Methodology

Data Preprocessing

This project did not require any data preprocessing since the sensor specifications and environment designs were provided as part of the starter package.

Implementation

- The state space is essentially represented by the maze coordinates of the locations the robot explores. This was selected to accurate as possible identify the unique states without make the space too large to effectively explore within the allotted time. Hence the possible values of the space for each maze would be equivalent to N x N, where N is the dimension of one size e.g. 12 for maze 1.
- The cost to the goal was computed using the average of the manhattan distance to each of the four goal states.
- The process used: During the first run before proceeding to determine the next move the robot test if its in the goal state. If it is the robot stores the path that got it there in preparation for the second run and it re-initializes. If the robot has determined that the current location is not within in goal bounds it continues its process to determine the next step.
- The robot selects a path based on the search algorithm in use. It then compares its current location to the state on that path prior to the frontier state to determine if it can proceed from its current location or it needs to reverse along the previous path to a common node and then proceed to the next state to be explored. Based on the results it either reverses or move forward ignorer to get to the next state to explore it.
- If the robot needs to reverse the path to the next state's predecessor is computed and used to navigate the robot there. Once the robot reaches the next state predecessor it moves forward to explore the new state. If no reversing is required the robot can move from its current location directly to the next state for its exploration.
- Each action required to navigate consists of a rotation value and movement value that is computed based predecessor and successor pair states. The robots location is constant updated with every action taken.
- Both the BFS and A* the set of explored states and list of paths with unexplored possible next (frontier) states are updated before determining the next path with a frontier state to be explored.
- Breadth First Search determines the next path with a state on the frontier to be explored by sort paths in the frontier set and selecting the shallowest path to explored next. the frontiers stated is extracted and used to determine and excuse action(s) required to get the robot to explore the state
- A* Search determines the next path with a state on the frontier to be explored by sort paths in the frontier set and selecting the path with the lowest combined cost of the path and estimated cost to the goal to explored next. The frontiers stated is extracted and used to determine and excuse action(s) required to get the robot to

explore the state. The length of the path is used as the path cost and the estimated cost to the goal is the average of the manhattan distance to each of the four goal states from the selected frontier state.

- Due to the initial decision to have each state represented by: (location, heading, action) and the algorithm to quite a long time to search the space and was sequently reduced to where each state was effectively represent by the location coordinates alone. This significantly improved the expiration time required and gave even better results for the second run.
- In changing the way each state was represent i had figure out a way to compute action based predecessor, successor states and the robots current or proposed location and heading.
- Initially the A* algorithm could search even the significant reduced state space within the a lot time constraints. The constraints was temporarily increase to allow trouble shooting, which allowed me to realize that the need to tune the weights applied to the estimated cost to the goal. The value was initially one tuning it allowed the exploration time to reduce significantly

Refinements

- The initial state space was quite large with each state represented by: (location, heading, action) and the algorithm to quite a long time to search the space and was sequently reduced to where each state was effectively represent by the location coordinates alone. This significantly improved the exploration time required and gave even better results for the second run. Initial the A* model failed to complete the exploration with 100 time steps, for instance it just under 5000 steps to explore maze 1 and after reducing the state space it took just over 1200 time steps.
- Actions since no longer being stored in the state object had to be computed pure based on successor and predecessor coordinates. This was necessary to support the above refinement.
- Initially the A* algorithm could search even the significant reduced state space within the time constraints. The constraints was temporarily increase to allow trouble shooting, which allowed me to realize that the need to tune the weights applied to the estimated cost to the goal. The value was initially one tuning it allowed the exploration time to reduce significantly. Initially it took over 1200 time steps to explore for instance maze 1 and after this refinement exploration time was reduced to 749 but did find an optimal path for maze 1.
- Initially the A* path cost was strictly based on path length resulted in the algorithm find a slightly sub optimal path. The computation was subsequently changed to penalize paths with more turns and this allow the algorithm to find a optimal path. Initially a path of length 32 was found for maze 1 while the length of the optimal path was 30 after this refinement the path found maze 1 was reduced to 30, the optimal path. However the model was unable to find the optimal path for the larger mazes due to the imposed time constraint.

IV. Results

Model Evaluation and Validation

MAZE 1

- The benchmark model's (Breadth First search: BFS) exploration was much faster than A* but the path it found to the goal was significantly much less optimal.
- BFS's took only 63 time steps to explore while A* took 473 time steps. Which indicates that that while BFS only explores a small part of the maze while A* explores much more and is thus able find a much more optimum path.
- The optimal path to the goal for Maze one (01) is 30 time steps, BFS found path to the goal of 40 time steps and A* 30 Time steps.
- A* was able find an optimal path after over 1200 time steps of exploration with the weighting of the manhattan estimate of cost to the goal reduced to 1.
- The A* final weight for the estimated cost was 22, this value was tuned to minimize exploration time and overall score. A* used a factor of ten (10) time costs for changes in direction, thus paths with more turn are avoid.

Time	BFS	A*
Run 1	63	749
Run 2	38	30
Score = R2 + R1/ 30	40.1	54.967

A* PATH: ((0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (1, 6), (2, 6), (2, 5), (3, 5), (3, 4), (4, 4), (4, 5), (4, 6), (4, 7), (5, 7), (6, 7), (6, 8), (7, 8), (7, 9), (8, 9), (8, 10), (9, 10), (9, 9), (9, 8), (8, 8), (8, 7), (7, 7), (7, 6), (6, 6))

BFS PATH: ((0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (3, 1), (2, 1), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2), (5, 2), (6, 2), (6, 1), (7, 1), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (11, 1), (11, 2), (11, 3), (10, 3), (9, 3), (8, 3), (8, 4), (9, 4), (9, 5), (9, 6), (9, 7), (9, 8), (8, 8), (8, 7), (7, 7), (7, 6), (6, 6))

Figure

4. : Maze 1 Results: Run times, scores, paths of BFS and A*

MAZE 2

- The benchmark model's (Breadth First search: BFS) exploration was much faster than A* but the path it found to the goal was significantly much less optimal.
- BFS's took only 144 time steps to explore while A* took 820 time steps. Which indicates that that while BFS only explores a small part of the maze while A* explores much more and is thus able find a much more optimum path.
- The optimal path to the goal for Maze two (02) is 43 time steps, BFS found path to the goal of 51 time steps and A* 47 Time steps.
- The A* final weight for the estimated cost was 60, this value was tuned to minimize exploration time and overall score. A* used a factor of one (1) time costs for changes in direction, thus paths with more turn are not avoid. Higher values led increase exploration time, allowing the robot to discover a more optimal path and exceeding the time limits.

Time	BFS	A*
Run 1	144	820
Run 2	51	47
Score = R2 + R1/ 30	55.8	74.333

A* PATH: ((0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (1, 7), (2, 7), (3, 7), (3, 6), (4, 6), (4, 5), (5, 5), (6, 5), (6, 4), (5, 4), (5, 3), (6, 3), (7, 3), (8, 3), (8, 4), (9, 4), (10, 4), (10, 3), (11, 3), (11, 2), (12, 2), (12, 3), (12, 4), (12, 5), (13, 5), (13, 6), (12, 6), (11, 6), (11, 7), (10, 7), (10, 8), (9, 8), (9, 7), (9, 6), (8, 6), (8, 7), (8, 8), (7, 8), (6, 8), (6, 7))

BFS PATH: ((0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (2, 2), (3, 2), (3, 1), (2, 1), (2, 0), (3, 0), (4, 0), (5, 0), (5, 1), (6, 1), (6, 2), (6, 3), (7, 3), (7, 2), (7, 1), (7, 0), (8, 0), (9, 0), (9, 1), (10, 1), (10, 2), (11, 2), (12, 2), (12, 3), (13, 3), (13, 4), (13, 5), (13, 6), (13, 7), (13, 8), (13, 9), (12, 9), (11, 9), (11, 8), (11, 7), (10, 7), (10, 8), (9, 8), (9, 7), (9, 6), (8, 6), (8, 7), (8, 8), (7, 8), (6, 8), (6, 7))

Figure 5. : Maze 2 Results: Run times, scores, paths of BFS and A*

Maze 3

- The benchmark model's (Breadth First search: BFS) exploration was not as fast as A* and they both found the same path.
- BFS's took only 408 time steps to explore while A* took 250 time steps. Which indicates that that while BFS may have explore a larger part of the maze while A* may have explored less both found the same path.
- The optimal path to the goal for Maze two (02) is 47 time steps, BFS found path to the goal of 59 time steps and A* 59 Time steps.
- The A* final weight for the estimated cost was 55, this value was tuned to minimize exploration time and overall score. A* used a factor of one (10) time costs for changes in direction, thus paths with more turns are avoided. Higher values led increase exploration time, allowing the robot to discover a more optimal path and exceeding the time limits.

Time	BFS	A*
Run 1	408	250
Run 2	59	59
Score = R2 + R1/ 30	72.6	67.33

A* PATH: ((0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (3, 3), (4, 3), (4, 4), (3, 4), (3, 5), (3, 6), (2, 6), (1, 6), (1, 5), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 10), (1, 10), (1, 11), (2, 11), (2, 10), (2, 9), (3, 9), (3, 10), (4, 10), (5, 10), (5, 11), (4, 11), (4, 12), (5, 12), (6, 12), (6, 11), (7, 11), (7, 12), (8, 12), (8, 11), (9, 11), (10, 11), (11, 11), (11, 10), (10, 10), (9, 10), (8, 10), (7, 10), (6, 10), (6, 9), (6, 8), (5, 8), (5, 7), (6, 7), (6, 6), (6, 5), (7, 5), (8, 5), (8, 6), (8, 7))

BFS PATH: ((0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (2, 2), (1, 2), (1, 1), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (5, 1), (6, 1), (6, 2), (7, 2), (8, 2), (9, 2), (10, 2), (10, 1), (9, 1), (8, 1), (7, 1), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (11, 1), (11, 2), (11, 3), (12, 3), (13, 3), (13, 2), (12, 2), (12, 1), (13, 1), (14, 1), (15, 1), (15, 2), (15, 3),(15, 4), (15, 5), (15, 6), (15, 7), (15, 8), (14, 8), (13, 8), (12, 8), (11, 8), (10, 8), (9, 8), (9, 7), (10, 7), (10, 6), (9, 6), (8, 6), (8, 7))

Figure 6. : Maze 3 Results: Run times, scores, paths of BFS and A*

Justification

The A* model's results for the three mazes were an improvement on the benchmark model interns of it is able to constantly find a more optimal path. The A* model was able to find the optimal path within the time limit for the smaller maze one (01). However despite beating the benchmark for mazes two and three the A* model was unable to find the optimal paths during the time limit. Based on this i now believed that the A* approach is sufficient given the parameters of this problem.

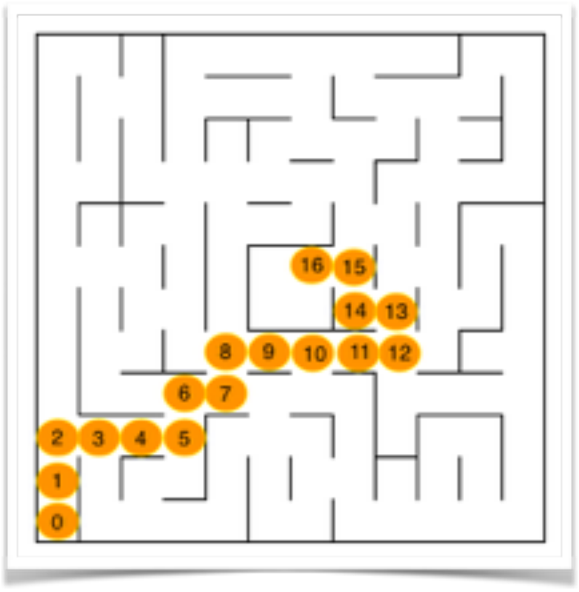
	Maze 1		Maze 2		Maze 3	
Time	**BFS**	**A***	**BFS**	**A***	**BFS**	**A***
Run 1	63	749	144	820	408	250
Run 2	38	30	51	47	59	59
Score = R2 + R1/ 30	40.1	54.967	55.8	74.333	72.6	67.33

Figure 7. : Summary of Maze Results

V. Conclusion

Free-Form Visualization

This maze was designed with more opportunity to access the goal. This should emphasize that the benchmark model's performance will suffer with more possible paths to search due to the lack of directed effort towards the goal. Thus proving the A* model to also be more robust to increases in possible paths towards the goal. This should also emphasize that with sufficient time the A* model will be able to identify the optimum path to the goal.



A* Path: ((0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (3, 3), (4, 3), (4, 4), (5, 4), (6, 4), (7, 4), (8, 4), (8, 5), (7, 5), (7, 6), (6, 6))

BFS Path: (0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (3, 1), (2, 1), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2), (5, 2), (6, 2), (6, 1), (7, 1), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (11, 1), (11, 2), (11, 3), (10, 3), (9, 3), (8, 3), (8, 4), (7, 4), (6, 4), (5, 4), (4, 4), (3, 4), (3, 5), (2, 5), (2, 6), (2, 7), (3, 7), (3, 8), (3, 9), (3, 10), (4, 10), (5, 10),(6, 10), (6, 11), (7, 11), (7, 10), (8, 10), (9, 10), (9, 9), (9, 8), (8, 8), (8, 7), (7, 7), (7, 6), (6, 6))

Figure 8. :12 X 12 Own Design Maze

Maze 4 - 12 X 12 additional maze

- The A* model's exploration was just as fast as BFS and the path it found to the goal was the optimal path.
- BFS's took only 107 time steps to explore while A* took 95 time steps. Which indicates that that while BFS only explores a small part of the maze while A* explores much more and is thus able find a much more optimum path.
- The optimal path to the goal for Maze one (01) is 16 time steps, BFS found path to the goal of 56 time steps and A* 16 Time steps.
- The A* final weight for the estimated cost was 22, this value was tuned to minimize exploration time and overall score. A* used a factor of one (1) time costs for changes in direction, thus paths with more turn are avoid.
- Maze 4 is a 12 X 12 maze like maze one but with more possible path designed to explore the weakness of the benchmark model. As result benchmark model result were worst and the A* model's performance was much better. this displayed it robustness to increase in possible paths due to its ability to select paths that may get to the goal faster.

	Maze 1		Maze 2		Maze 3		Maze 4	
Time	**BFS**	**A***	**BFS**	**A***	**BFS**	**A***	**BFS**	**A***
Run 1	63	749	144	820	408	250	107	95
Run 2	38	30	51	47	59	59	56	16
Score = R2 + R1/ 30	40.1	54.967	55.8	74.333	72.6	67.33	59.567	19.167

Figure 9. : Summary Results Including Own Design

REFLECTION

- It was decided to use Search based classical approaches to pathfinding in AI and Breadth First and A* Search algorithms were selected after try out a few the approaches i have successfully used on other projects on similar problems.
- The process used: During the first run before proceeding to determine the next move the robot test if its in the goal state. If it is the robot stores the path that got it there in preparation for the second run and it re-initializes. If the robot has determined that the current location is not within in goal bounds it continues its process to determine the next step.

- The robot selects a path based on the search algorithm in use. It then compares its current location to the state on that path prior to the frontier state to determine if it can proceed from its current location or it needs to reverse along the previous path to a common node and then proceed to the next state to be explored. Based on the results it either reverses or move forward ignorer to get to the next state to explore it.
- If the robot needs to reverse the path to the next state's predecessor is computed and used to navigate the robot there. Once the robot reaches the next state predecessor it moves forward to explore the new state. If no reversing is required the robot can move from its current location directly to the next state for its exploration.
- Each action required to navigate consists of a rotation value and movement value that is computed based predecessor and successor pair states. The robots location is constant updated with every action taken.
- Both the BFS and A* the set of explored states and list of paths with unexplored possible next (frontier) states are updated before determining the next path with a frontier state to be explored.
- Breadth First Search determines the next path with a state on the frontier to be explored by sort paths in the frontier set and selecting the shallowest path to explored next. the frontiers stated is extracted and used to determine and excuse action(s) required to get the robot to explore the state
- A* Search determines the next path with a state on the frontier to be explored by sort paths in the frontier set and selecting the path with the lowest combined cost of the path and estimated cost to the goal to explored next. The frontiers stated is extracted and used to determine and excuse action(s) required to get the robot to explore the state. The length of the path is used as the path cost and the estimated cost to the goal is the average of the manhattan distance to each of the four goal states from the selected frontier state.
- Due to the initial decision to have each state represented by: (location, heading, action) and the algorithm to quite a long time to search the space and was sequently reduced to where each state was effectively represent by the location coordinates alone. This significantly improved the expiration time required and gave even better results for the second run.
- In changing the way each state was represent i had figure out a way to compute action based predecessor, successor states and the robots current or proposed location and heading.
- Initially the A* algorithm could search even the significant reduced state space within the a lot time constraints. The constraints was temporarily increase to allow trouble shooting, which allowed me to realize that the need to tune the weights applied to the estimated cost to the goal. The value was initially one tuning it allowed the exploration time to reduce significantly

Improvement

1. To be used in a more real world environment a extensive localization and mapping system would be required to compensate to variations in measurements, errors movements and changes in the environment. Kalyan filters could be implemented to support localization and mapping. This would allow the model to be able to

correct for uncertainties associated with location predictions based on movement and sensor measurements that would be an inherent part of a real world environment. In the current approach we are assuming when robot tries to move a certain distance it actually does when is quite possible i skid some additional distance or some unexpected object block its path.

2. The implementation did not utilize the maximum allowable step range. The could improved by leveraging adaptive step ranges to accelerate exploration and progress towards the goal.