Machine Learning Engineering Nanodegree

Capstone Project

Lívio Tonini Gouveia e Silva

Plot and Navigate a Virtual Maze

December 26th, 2017

Project Overview

This project is inspired on the Micromouse competition, a competition that began in the late 1970s where students have an opportunity to learn and practice important skills such as circuit design, programming, and many others. The objective is for a robot to find its way in a maze to reach a goal location as fast as possible in that maze with no previous knowledge of it. This will be a virtual simulation of this task, where the virtual robot will try to find the goal location of a maze, considering only the decision making process of the robot and any hardware modeling will not be included.

Problem Statement

In this competition, the robot must reach the center region of the maze as quickly as possible. For that, the competition is divided in two steps, the first is the exploration step, in which the robots attempt to map the maze and reach the center, after that, the robot may keep exploring the maze or continue to the second step. In the next step, the robot must reach the goal region as quickly as it can by using the knowledge obtained on the previous step.

As this is a virtual simulation of the competition, instead of considering time as a measurement of efficiency of the robot, it is used a time step, where in each time step the robot will decide its next action. For this project, a maximum of 1000 time steps can be used on both steps.

Metrics

To evaluate the performance of the robot in a maze, a scoring function is used that is the number of time steps that takes the robot to reach the goal location in the second run plus one thirtieth of the time steps used for the robot in the first run.

Data Exploration

For this problem, the robot has to explore the maze to find a path to the goal so instead of using a dataset, will be available for the robot three virtual sensors, which will detect the distance to an obstacle in each of its sides and in its front. As this is a virtual simulation of a robot, it will be assumed that all sensors will work perfectly and so will the robots movements.

In each time step of the simulation the robot may choose to rotate ninety degrees clockwise or counterclockwise and to move forwards or backwards up to three square units. For any given time step, the robot must return the rotation desired and the movement desired. The rotation should be an integer of one of the following values: -90 for ninety degrees counterclockwise rotation, 90 for a ninety degrees clockwise rotation or 0 for no rotation. The movement should also be an integer ranging from [-3, 3] inclusive. When the robot reaches the goal and it desires to move on to the second run, it should return a value on both variables of "Reset" to inform the simulator to start the next run. Any other values besides those the simulator will ignore and a default rotation of zero and a default movement of zero will occur.

If the robot attempts to move to a wall, the wall will stop its movement. As the robot has no location sensor nor the simulator will tell the robot where it is or which direction it is facing, an internal location system must be coded in the robot's algorithm to keep track where it is and even know when it has reached the goal so that the second run may start.

In every maze, the start location of the robot will always be the bottom left corner, facing North, and in that square, there should always be a wall on each of the robot's sides and in its back. The goal location will be the center region of the maze, a 2x2 square units region.

For the mazes, this project will use the three mazes provided by Udacity, with sizes 12x12 squares unit, 14x14 squares unit and 16x16 squares unit, in conjunction with the starter code provided, which includes a file to test the performance of the robot in a maze, a maze simulation file that simulates the environment of the maze and validates the robot's movements and a file that generates an image of a maze.

The maze files are encoded in text files where the first line will have a number describing the dimension "n" of a *n x n* maze. The following "n" lines will have a number per "n" square of the maze, describing which ways are open to movement. Each of those numbers is a four-bit number and each bit will have a value of zero if an edge has a wall, or one if is an open space. The 1s register corresponds to the North-facing side, the 2s register corresponds to the East-facing side, the 4s register to the South-facing side and the 8s register to the West-facing side. As an example, the number 12 will be (0*1 + 0*2 + 1*4 + 1*8 = 12) meaning that the South and West of that square has an open space and the North and East side has a wall.

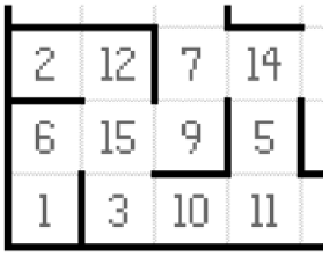


Image 1 – Illustration of the maze's encoding system.

As an illustration, the maze with 16x16 square units dimension is displayed below in image 2.

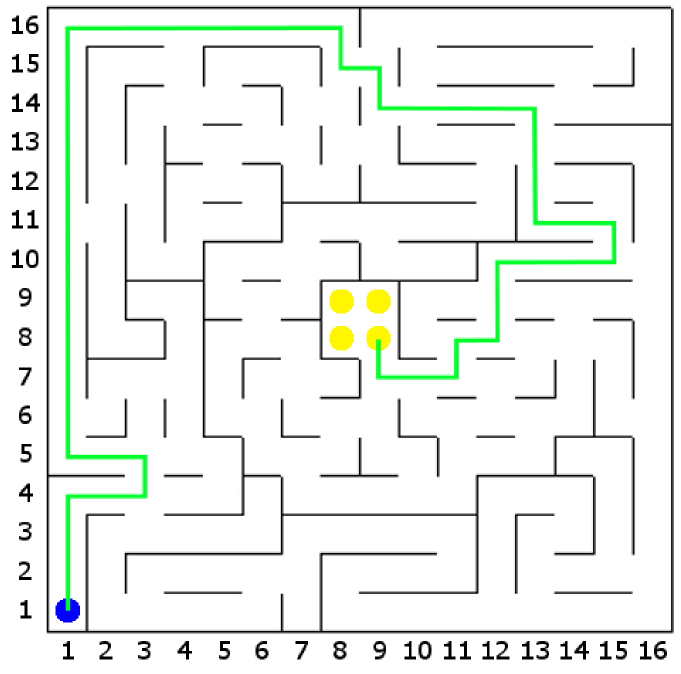


Image 2 – Maze 3 best path.

In the picture above, the start position is marked with a blue dot. The object of the agent will be to reach the center of the maze, marked with yellow dots. There are multiple paths that will lead to the center location but the optimal path is showed with a green line.

The optimal path takes a total of 50 squares (including the start location and the end location), but considering that the robot may move up to 3 squares per action, the optimal actions for the robot would be to arrive in the goal with 25 Actions.

Analyzing this maze, there are some aspects that must be considered in order to properly navigate in it. Examining the maze, we can see some paths that will lead to dead-ends and so, such paths should be avoided. As the robot can only rotate a maximum of 90º and so it cannot completely turn around, dead-ends offers some challenge to the agent and so it must learn how to handle them to be able to leave those paths not wasting many actions doing so. Another possible problem a robot may face when exploring this maze, is to be caught in a loop region. In such regions, it is possible for the robot to run in loops or at least to go pass through such regions multiple times, wasting actions that would be better spent trying to go elsewhere. To avoid such scenarios, the robot should memorize where it has been to prefer to explore unvisited locations instead of going to the same place multiple times. In the image below, the dead-ends are marked with a red dot while some of the loop regions are marked with in orange.

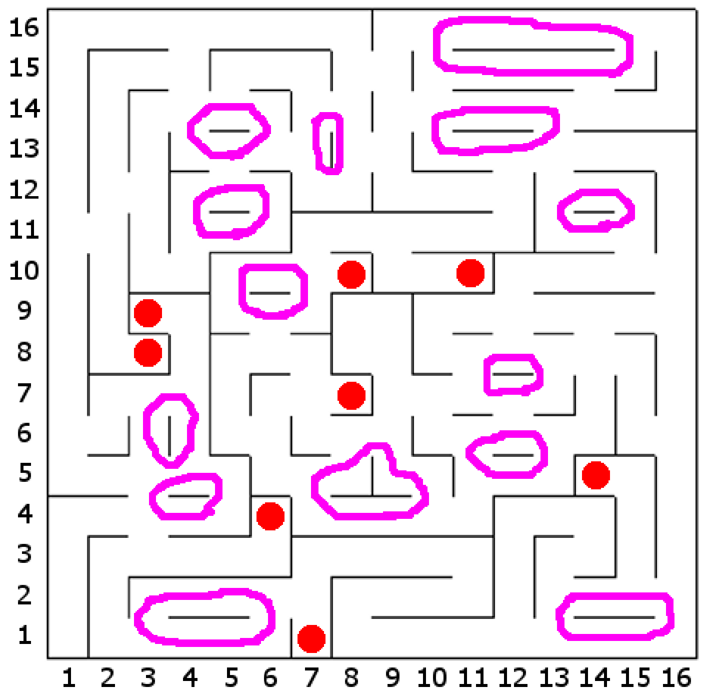


Image 3 – Maze 3 possible problems.

Algorithms and Technique

This project can be divided in two different tasks. The first is for a robot to find an optimal path to a location given the possibilities that it knows, and this is what the agent must do in the second run. The first run on the other hand has a different problem, the robot must explore the maze not only to reach the goal but preferentially also explore enough of the maze to be able to find the optimal path in the second run. On both runs, time is a factor, or more precisely, time steps are a factor, and so both tasks should be executed as fast as possible to achieve a good result.

To the exploration run, first the agent will need to make use of its sensors to learn to navigate in the maze without colliding into walls. For this, in each time step the robot will receive information from the sensors regarding the distances to the nearest walls in each of its sides and in its front and this will let the agent know the possible actions that it can take. Although it is allowed for the robot to move three square units at a time, in the exploration run the robot will only move one square at a time so that it can properly map out the maze and not skipping locations that are junction points that could lead to a faster path to the goal.

For a good exploration, the robot must take some considerations not to waste time. In a maze, there will be paths that lead to dead-ends, so the robot must learn to avoid them. In the exploration algorithm, was included a way to identify dead-ends and mark the paths that lead to them in order to make the agent avoid going to such areas more than once. Another likely problem is for the robot to go to the same location multiple times or even get stuck in some possible loop region while attempting to reach the goal. To avoid this issue, a solution inspired on Tremáux's algorithm was applied. This algorithm add marks to the path that the agent takes so in each junction point the agent will prefer an unvisited location. If in a junction point every possible path is already marked, the agent will choose to go back. At the end, each location should have a maximum of two marks and a solution of the maze would be a path with least marks. This will allow the robot to avoid going to the same spot multiple times and so having a better overall exploration of the maze.

Finally, the robot have no prior knowledge of the maze except for the fact that the goal location of a maze is located at the center of the maze. With this information and the maze dimensions that are also available for the agent, a heuristic will be added to the decision making process of the robot so that, if possible, It will choose a next location that will reduce the distance to the center of the maze and thus, get closer to the objective.

Combining the aspects mentioned above, the exploration algorithm of the robot should make it reach the goal location in considerable less time steps than the allowed 1000 time steps while avoiding passing through the same locations and thus allowing the robot to explore the maze so that it can find at least a sub-optimal path in the second run. However, even with all the above aspects implemented, in a junction point there may be more than one possible action that will lead to an unvisited location that will reduce the distance to the goal and so reaching a tie. When ties happens, the robot will take a random decision amongst the best decisions and this will make the exploration run have a randomness factor. This randomness will make the robot achieve different results each time that it is tested and for a better evaluation of its performance, each maze will be tested one hundred times so that the results can be averaged to have a better measurement of the robot's performance.

Once the robot has explored and mapped the maze, it will be ready for the second run. The robot will be returned to the start location and calculate the quickest path among the explored locations to reach the goal. For this calculation, the A* pathfinding algorithm is implemented that uses a distance heuristics to calculate the best path. This algorithm calculates for each node, or in this case, each square unit of the maze, a distance of the current location for the goal location, and uses a priority queue to test possible paths that has an overall smaller cost to reach the goal. For the distance heuristics, different possibilities may be used, but as for this project the robot will not be allowed to move in diagonals, a Manhattan distance heuristics, also known as taxi-cab distance, will be used for the distance heuristics in A*.

For both runs, it is important that the robot keep track of where it is and where it is facing to be able to navigate in the maze. As the only inputs given to the robot in each time step is the obstacle sensors inputs, the robot has no location system embedded and so it must update its current location and the heading direction internally, by considering the start location, that is always the same, and the actions taken in each time step, that will generate a given variation in direction and location that will be used to update its internal navigation system.

For a better understanding of the robots actions in each step, and to help debugging, a simulation log will be created where it will keep track of the inputs and decisions of the robot in each time step. For a better visualization it will also be implemented a rendering function that will allow a visualization of the robot’s movements in the tested maze.

Benchmark

As a benchmark model for the robot in the maze it will be used the original Trémaux’s algorithm to find the goal location in the exploration run. Additionally, the robot’s performance will also be compared to the optimum paths for each maze and also the optimal amount of movements that the robot can take to follow that path as fast as it can. As the robot will have no prior knowledge of the maze, its decision making process will have some degree of randomness and so different results may be achieved each time it is run, so for this the robot will be tested multiple times in each maze and have its results averaged for a better comparison.

Data Preprocessing

For this project, as there is no hardware modeling and considering that the sensor specifications and environment designs were provided as a start code from Udacity and it is assumed that it all works perfectly, there is no need for data preprocessing. In addition, no datasets will be used, it is a task for the robot to acquire information about the maze it is exploring to map its environment, so for this, no preprocessing will be used either.

Implementation

The first thing that the robot should be able to do is to be able to navigate the maze without colliding into walls. To find the possible paths in each square location, the sensors values are used. As in the exploration run, the robot will only move one square unit at a time, the robot must only know if the sensor value for that direction is greater than zero.

The robot will preferentially avoid moving backwards as it has no obstacle sensor in its back and to avoid passing through the same locations, however, there will be times when such action is needed so the agent must know if it has obstacles behind it or not. To compensate for the lack of sensor in the back, a “virtual” sensor that will keep track of the available space in the back of the robot will be created. As the robot, in the start location, has a wall behind it, the available space in its back is zero, but whenever it moves forward, the movement value is added to that sensor. When a robot makes a turn, the space that was detected on the opposite side of where the robot turned will be the new value for the backwards sensor.

Once the robot know where it can go in each time, it is important to add an internal location system to it, so that it knows where it is and where it is facing. For this task, the variation of movement and rotation in each time step is used to adjust the new location and direction of the robot, using as reference the starter location of the robot, and the initial heading direction Once the robot makes a 90º turn when it was previous facing North, it is calculated that the new heading direction is now West and if it was moved 1 square unit, the X location coordinate is subtracted by 1 and the new location is found.

Whenever the agent encountered a dead-end, it must know how to act to move out of it. First, the agent detects a dead-end by checking if the three obstacle sensors returns the value of zero, which means that there is nowhere to go, except backwards. However, only moving backwards once may not be enough as dead-ends may have a length greater than one. To overcome such problem, whenever a dead-end was detected, the robot entered a “going backwards” mode, where it starts to move back until it reaches a junction point, knowing that there will be other possibilities of actions that will not lead to the dead-end.

When at a junction point, the robot will prefer an unvisited location that leads it closer to the goal location. For that, the robot must know where the goal location is. As the maze dimensions are an input to the robot, it calculates the 2x2 goal locations of maze. As there are four goal locations, the robot will choose one to calculate the heuristics. For this, in each time step it is used the current robot location to see which is the closest goal location and that is the one used to help to decide the next best action. In the image 4 below two examples are illustrated, where the robot is represented in blue, the closest goal location in yellow.

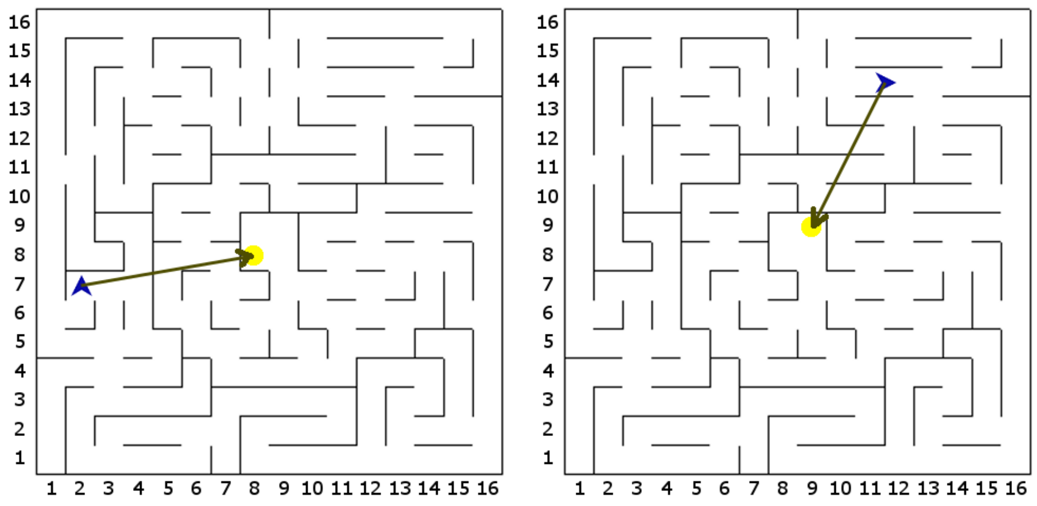


Image 4 – Robot’s closest goal location.

To complete the exploration algorithm, a “memory” must be added to the robot’s decision making process. A variable is used to store each location visited by the robot and how many times was visited, e.g. if the location (2, 4) was visited twice, the variable for that location will be: (2, 4) = 2

There will be times where even when considering unvisited locations and new locations that will be closer to the goal, that it will have more than one next possible action. When such scenarios happen, the robot will randomly pick a next action, among the best ones.

Combining the techniques above, the robot was tested in the exploration run until the goal was found, the results displayed below are of an average of 100 trials on each maze.

Maze	Reached Goal	Time Steps Used	Explorated Area
Maze 1	Yes	135	59,04%
Maze 2	Yes	227	61,60%
Maze 3	Yes	129	38,50%

Table 1 – Exploration run in the tested mazes.

Once the robot was able to reach the goal in the first run without using too many time steps, it must now calculates the best actions to reach the goal.

To implement the A* algorithm, it must be known where the robot was and for each of those locations, it must also be known where can the robot go from there, the "neighbors" from that location. To save such information, an additional variable is created that in each time step it is tested whether such location already exists in such variable or not and if not, it is added with the location of the square, and the neighbor squares. When saving the neighbors of a location however, there may be cases where a neighbor wasn't visited by the robot so that neighbor can't be considered in the A* algorithm as it would not have enough information of that square to calculate the best path, so the A* will only consider neighbors that was visited by the robot at some point.

Once the first run is ended, each location visited with the respective neighbors are sent to the A* algorithm so that it can calculate the best path. This algorithm when searching for the best path considers two values: the cost to reach that new location, conventionally named as the "G" value and a heuristic value, conventionally named as the "H" value, that will be the estimated distance of a given square location to the goal location.

For the heuristic value, the Manhattan distance, commonly known as the Taxicab distance, is calculated. That value is calculated by summing the absolute value of the differences between the "x" and "y" coordinates of the goal location to the current location. The image below demonstrates the Taxicab distance in a grid and compares it to the Euclidian distance. The red, blue and yellow lines have the same Manhattan distance (or Taxicab distance), while the green line represents the Euclidian distance.

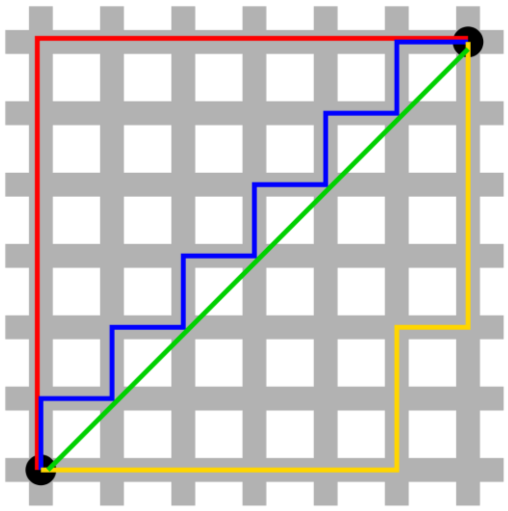


Image 5 – Illustration of the Taxicab distances.

Once the best path is found, the robot now know where to go but it must know how to go to those locations as it must output to the simulator a rotation and a movement. For this reason, the locations of the best path found are converted to actions. Results of the A* algorithm are shown below.

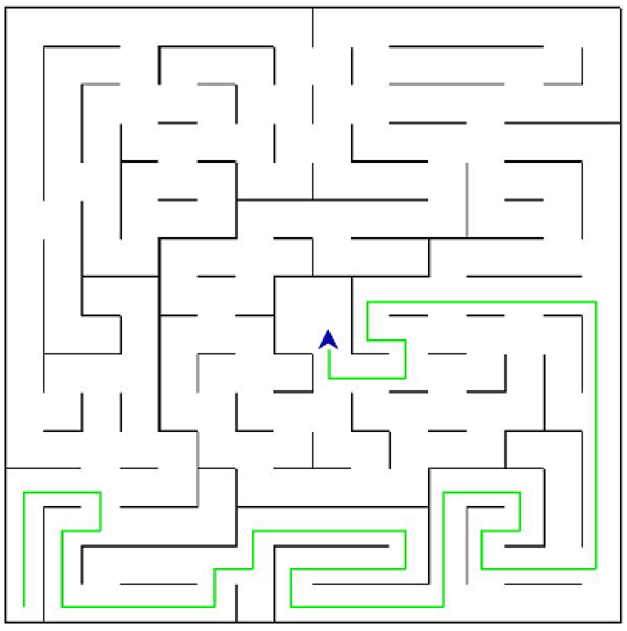
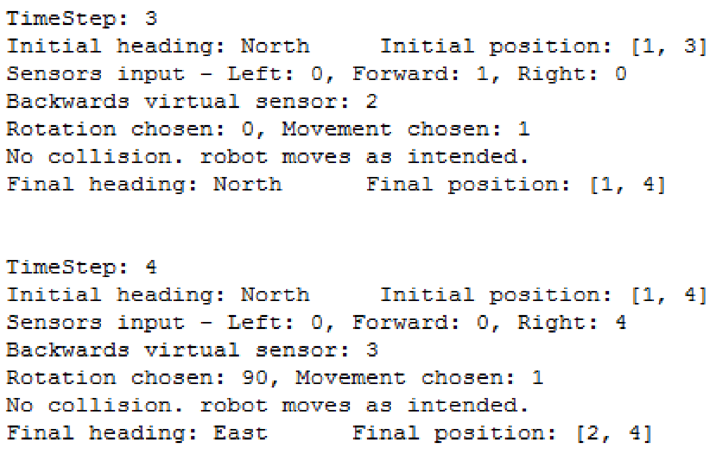


Image 6 – Example of best path found using A* algorithm.

To a better understanding of what is happening in each time step and to a better evaluation and debugging of the algorithm, a function that creates a log file of the simulation is implemented. At every time step it is written the initial location and direction of the robot, the sensors values, the action chosen by the agent, whether or not a collision happened and finally, the final location and heading direction of the robot after taking that action using the internal location system created.



```
TimeStep: 3
Initial heading: North     Initial position: [1, 3]
Sensors input - Left: 0, Forward: 1, Right: 0
Backwards virtual sensor: 2
Rotation chosen: 0, Movement chosen: 1
No collision. robot moves as intended.
Final heading: North      Final position: [1, 4]


TimeStep: 4
Initial heading: North     Initial position: [1, 4]
Sensors input - Left: 0, Forward: 0, Right: 4
Backwards virtual sensor: 3
Rotation chosen: 90, Movement chosen: 1
No collision. robot moves as intended.
Final heading: East      Final position: [2, 4]
```

Image 7 – Example of the simulation log file.

Finally, to better visualize the robot in a maze and the path that it takes, a function that displays the robot in the maze is created to render the simulation. This allows for an easier understanding of the simulation and a good way to detect possible improvements for the algorithm. An example image of the rendered simulation is provided below.

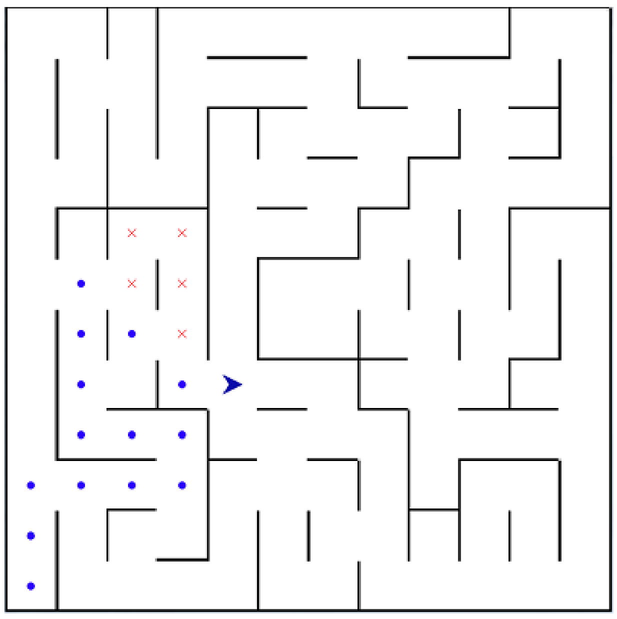


Image 8 – Example of the rendered simulation.

Below an example of the exploration run and the second run of the maze 3 is displayed, with the images generated by the algorithm. In the first image, the blue dots represent locations visited once while a red x represent locations visited twice. In the second image the green line represent the best path calculated using the A* technique.

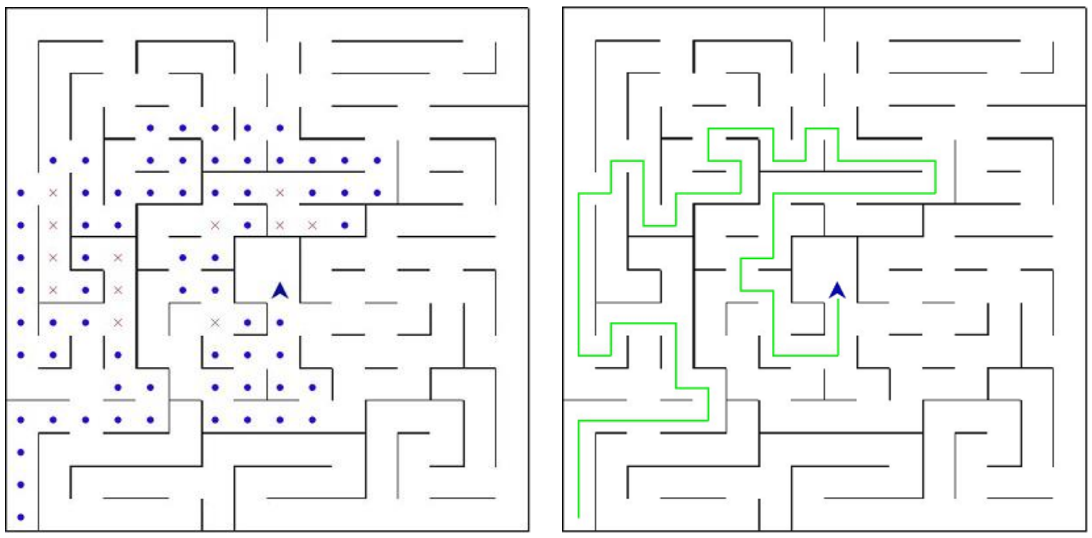


Image 9 – Example of the exploration run and the second run on maze 3.

Refinement

During the development of this project, especially after the rendering function was created, there were some aspects that were analyzed that could improve the overall performance of the algorithm.

In order to reduce the amount of time steps used when exploring, whenever a dead-end was encountered by the agent, it now calculates the distance of the last junction point that the robot passed through and go back to that junction point if possible. If not, it goes backs as much as it can per time step, until a junction point is found. To know where the last junction point was found, an additional variable that saves the location of the last junction point is created, checking to see if at least three of the four sensors (including the virtual backwards sensor) indicates that there is room to move to. This modification makes leaving dead-ends faster, and so the exploration run uses less time steps, which can improve the overall score of the robot.

Another modification was how the dead-ends are marked. If marked only once, there is a possibility that the agent may return to the dead-end again, so to avoid this, when the robot are in such locations, it adds 5 marks to that location and the following locations until a junction point is found, normalizing the marks then. Such change in the code will make dead-ends a place not to be visited again. In addition, as the starter location of the maze is a dead-end, the robot starts the maze by marking the path as if it were in a dead-end. The image below illustrates this modification, where paths visited once are marked with a blue dot, and dead-end locations are marked with an exclamation inside a red circle.

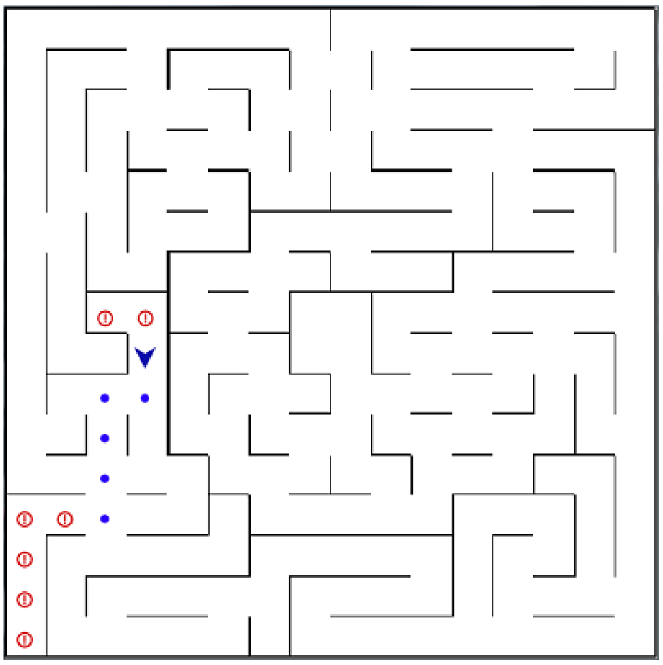


Image 10 – Illustration of the new dead-end marks.

After analyzing the robot’s decision making so far, it was observed another possibility of improvement. The exploration heuristics uses the goal’s nearest location to the agent to help to decide, in a junction point, which path to take. However, it often presents situation where two possibilities would equally bring the robot closer to the goal’s nearest location, so to break such ties, the robot also verify which of the possible actions will get the robot closer to the goal’s center point, that is, the center point of the 2x2 central region area of the maze. This improvement makes the robot pursue the goal more efficiently. As an example, the image 11 below is from the 14x14 maze provided by Udacity. The robot, illustrated in blue, in the current location can choose either to go North or to go West, the movements are represented by brown arrows. Its priority per heuristic would be to get as closer to the nearest goal location, represented by the yellow dot, as it can, but in that location it is unable to move South. Either going West or North would make it more distant from the closest goal location so with only that heuristic it would choose either option. Adding this new heuristic however, in such situations it now searches for the new location that will bring it closer to the central goal region, represented by the red dot below and so it chooses to go West, which has a higher chance to reach the goal faster.

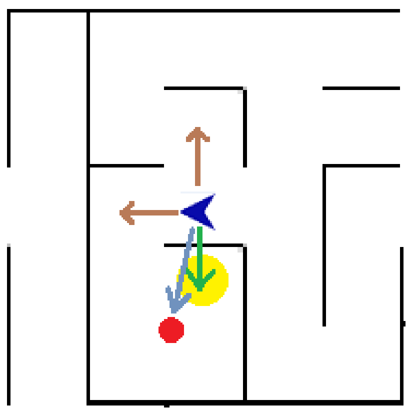


Image 11 – Illustration of the new heuristic implemented.

Below the performance results of the algorithm is displayed, comparing the before and after of having implemented the refinements, by testing each maze and averaging the results one hundred times.

Maze	Initial Model				Refined Model			
	First Run		Second Run		First Run		Second Run	
	Time Steps	Explorated Area	Number of Actions	Score	Time Steps	Explorated Area	Number of Actions	Score
1	136	59,04%	21	25,53	146	62,48%	20	24,87
2	228	61,60%	29	36,60	120	44,10%	31	35,00
3	129	38,50%	34	38,30	105	33,64%	34	37,50

Table 2 – Comparison between initial model and refined model.

Analyzing the table above, although the average scores did improve, the changes were small, but considering that this is a simulation of a competition where the slightest difference in results could make the difference, the refined model is still a considerable improvement.

All of the improvements were made on the exploration algorithm, to spend less times on dead-ends and to have a more direct search for the goal locations by applying the second heuristic, and the results above shows that the most significant changes were on the first run.

Model Evaluation and Validation

Each maze has an optimal path and an optimal amount of moves in the second run to reach the goal, considering that the robot may move up to three squares at once. Below the optimal path for each maze is compared to the refined model's results after averaging hundred trials.

Maze	Optimal Path		Refined Model		
	Path Length	Number of Moves	Path Length	Number of Moves	Frequency that the best path is found
1	30	17	36	20	21%
2	43	23	49	31	4%
3	49	25	57	34	1%

Table 3 – Comparison of the refined model's performance with the optimal path.

The algorithm developed does not find the optimal path with a high frequency due to the fact that the exploration run does not cover enough area to find that path. The exploration run was optimized so that it finds the goal as fast as it can so, given the rules of this project, the score for each run was improved by a faster exploration and this was demonstrated on the table X that compares the results before and after the refinements were made.

If the robot however explores the area that covers the optimal path, the A* algorithm succeeds in finding it and in the second run the robot reach the goal in the optimal number of moves. Below are examples of trials in each maze where the optimal path was found.

Maze	First Run		Second Run		Score
	Explorated Area	Time Steps	Path Length	Number of moves	
1	68,05%	165	30	17	22,50
2	77,55%	256	43	23	31,53
3	78,12%	287	51	25	34,57

Table 4 – Example of optimal path found on each maze.

Justification

The final results for each maze are compared with the benchmark model, the original Trémaux's algorithm, in the table 5 below. One hundred trials were tested on each maze and the results averaged for a better comparison.

Maze	Trémaux's Algorithm				Refined Model			
	First Run		Second Run		First Run		Second Run	
	Time Steps	Explorated Area	Number of Actions	Score	Time Steps	Explorated Area	Number of Actions	Score
1	213	73,96%	21	28,10	146	62,48%	20	24,87
2	283	68,88%	29	38,43	120	44,10%	31	35,00
3	331	67,96%	29	40,03	105	33,64%	34	37,50

Table 5 – Comparison between benchmark model and the refined model.

Analyzing the results above, an overall improvement of the score is clear on the refined model when comparing to the Trémaux's algorithm. That improvement is due mainly to the heuristics in the exploration that makes the robot reach the goal generally faster, and this can be seen above in the number of time steps used for the first run in the table above.

Free-Form Visualization

There are many possibilities for mazes, even by fixating the starting location and goal locations. Some maze designs would make exploration techniques more difficult and would make the task of building a generic exploration algorithm more challenging. The maze below was created as a possible scenario …

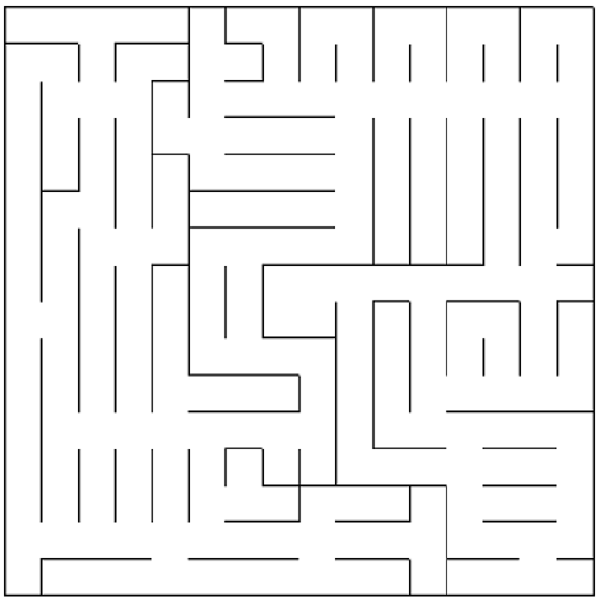


Image 12 – Custom maze created.

This maze has areas with many possibilities to move and multiple dead-ends. This would make a random exploration algorithm have more difficulty when trying to reach the goal. As a test, the original Trémaux's algorithm and the refined model algorithm created in this project were both tested and the results, after 100 trials, displayed below.

Maze	Trémaux's Algorithm				Refined Model			
	First Run		Second Run		First Run		Second Run	
	Time Steps	Explorated Area	Number of Actions	Score	Time Steps	Explorated Area	Number of Actions	Score
4	341	68,50%	22	33,37	218	48,76%	20	27,27

Table 6 – Benchmark model and refined model performance in the custom maze.

Reflection

This problem was divided in two parts, the exploration first and then finding the best path. For the exploration task, the problem was solved by taking small steps to achieve the final results. First the robot needed to be able to run the maze without colliding and knowing where it was, then the decision making process started to get more complex until the heuristic were implemented.

After creating the rendering function to see the robot's path in the maze, it was clear that were some aspects that could be improved, and so the algorithm was refined to handle some scenarios and promote a better overall performance. The final exploration algorithm had still some random degree and so its performance will vary each time it is tested, but it follows a logic to reach the goal and completes that task every time without wasting too many time steps.

In the second run the A* pathfinding algorithm was used to find the best path among the ones explored. Due to the randomness of the exploration run and the fact that the robot does not tend to explore the full maze, the maze's optimal path is not often found, but it does find the best path given.

In the end, a custom maze was design to offer some challenges to the robot to test its performance and to compare it with the benchmark model. In every maze, this project's algorithm proved to have a better performance than the benchmark model after 100 trials were tested and its results average.

For such problem, one decision that was difficult to make is how much would the robot invest in exploring in the first run before ending it and starting the second run. As the first run counts to one thirtieth of the final score, if the robot uses 29 more time steps exploring but, by doing so, finds a better path, it would be still result in a better score for the robot, so reaching the goal too fast is not ideal. However, investing too much time exploring will use more time steps and it is not guaranteed to find a better path to the goal, so we cannot be sure if the robot should keep exploring or not. Finding a balance to when stop explore was challenging.

Improvement

As initially stated, this project only approached the decision making process of a robot in a discrete domain to solve a maze. In the real Micromouse competition however, this is only one of the many tasks of building a robot and even so, it would all need to be adjusted to work in a continuous domain, there would be no time steps anymore and everything would have to be executed as fast as possible, including the run time of the algorithm.

To build such a robot, it would first need to make a project of its hardware, all the sensors reading used would have to be configured so that the robot would be able to understand and use that data. It would all add considerable more complexity to the robot, sensors could present errors, the movements wouldn't be perfect and adjustments in motion would have to be executed so that the robot doesn't stray from its path, acceleration would have to be taken into account, how would the robot make turns to go as fast as it can and much more.

Another consideration is that for this project, no diagonal movement was allowed, but in the real competition, a robot would be better of moving diagonally when it was possible, to have a more direct path and so a faster path to the goal.

REFERENCES

- **Trémaux's algorithm -** http://blog.jamisbuck.org/2014/05/12/tremauxs-algorithm.html
- **A* pathfinidng algorithm -** https://en.wikipedia.org/wiki/A*_search_algorithm
- David M. Willardson, "Analysis of Micromouse Maze Solving Algorithm" in Learning from Data, spring, 2001.
- Singh, Amanpreet & Sekhon, Gianetan. (2011). "A New Shortest Path Finding Algorithm For a Maze Solving Robot With Simulator".