Capstone Project

Machine Learning Engineer Nanodegree

Armin Sadighi

December 26, 2016

Robot Motion Planning Capstone Project

Plot and Navigate a Virtual Maze

Definition

Project Overview

This project is a robot navigation project, in which the goal is to train a robot in a maze. The goal is that the robot learns how to reach a goal in this maze as fast as possible. This project is based on a real competition called the Micromouse competition where a physical robot tries to do the same in a real maze build on the ground.

Problem Statement

At the beginning the robot is located at the lower left corner of a maze, the robot has no information about the map other than some sensor inputs. In the project the robot has to explore the maze once and try to find a goal, which is in the middle of the maze, and also explore the map. In the second round the robot has to reach the goal as fast as possible.
So the objective of this project is to explore an unknown maze with a robot and learn the paths and walls in the maze, and then explore what the robot has learned to reach the goal as fast as possible.
For this project there are some available components. These components are some test mazes, and also the model of the world along with a tester that feeds the information of the world to the robot. The objective here is to implement an agent that based on this given information and the testers input can, decides where to go next. In the following parts of this document the implemented algorithm will be explained in detail.

Metrics

To be able to evaluate the efficiency and effectiveness of the algorithm, a quantitative measure is required. For this project the unit of measurement will be time steps. In each time step the robot can rotate and move up to 3 fields.
The robot needs to explore the map and hence needs to rotate and move, with this the robot requires some time steps to reach the goal and explore the map in it's 2 runs. The robot has a

total of 1000 time steps for both of it runs to reach the goal. Also the robot can not finish the first run and request the second run if has has not reached the goal.
The score of the robot is the number of time steps in the second run, which should be as small as possible, plus one thirtieth of the first run, that is ment for the robot to explore the map. So the optimal case is that the robot reaches the goal extremely fast in the second run and also explores the map in a good way, to exploit this tradeoff as best as possible.

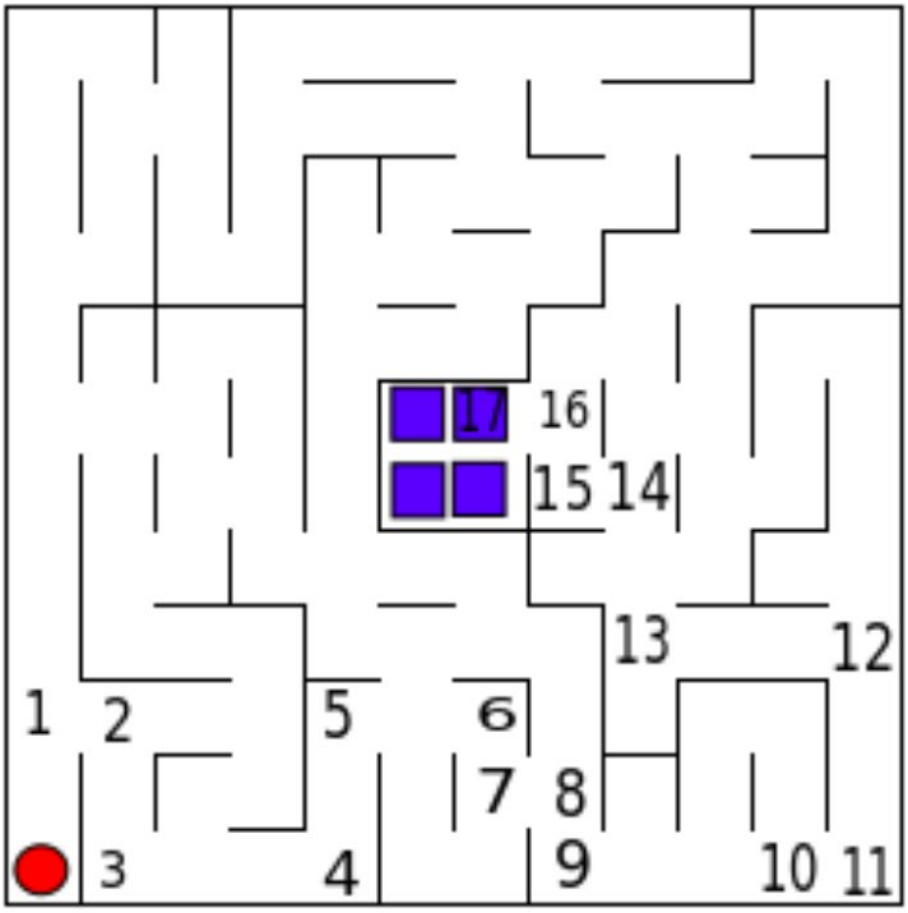
Analysis

Data Exploration and Exploratory Visualization

The robot is always placed in the corner bottom left square of the maze facing upward. This square has the unique property that the left and right of it are always closed, surrounded by walls, so the robot can only move upward, direct move toward it's heading. Also as this is a model and everything is considered perfect in this model the robot is always placed at the middle of the square it resides.
The robot needs some input to understand its surroundings and be able to plan for the next move. For this input the robot has three sensors that can recognize obstacles. These 3 sensors are connected to the robot's front, left and right side. What these sensors do is that they return the number of open squares on the three side until there is a wall. For example if the robot can see 4 open squares on its right side and then a wall the right hand sensor returns 4. Sensor input are given to the robot as a tuple in the form of (left, forward, right).
A time unit called time step is defined in the environment. In any time step the robot can rotate and decide on its movement with the following rules:

- Number of steps that the robot can take in each time step is an integer in the interval [-3,3]. Meaning that the robot can move up to 3 squares forward, in its heading direction, or up to 3 squares backward, hence the negative sign, in the direction of its heading.
- The robot can either have no rotation, rotate 90 degrees clockwise or 90 degrees counter clockwise. So the robot decides this rotation by returning the numbers 0, 90, -90 respectively for no rotation for clockwise rotation, and for counterclockwise rotation.

To sum it up till here the "next_move" function of the robot receives the sensor input data and returns the pair of rotation and movement numbers that tell the robot to rotate first, based on the output and then move also accordingly to the returned number for movements.
The rotation and movement of the robot are considered perfect. Meaning that it will happen just like outputted. If the robot hits the wall during an action the robot stays where it is.
The mazes used in the model for the robot environment have some fixed characteristics. These mazes are always square mazes with, n x n mazes, with n being one of the numbers 12, 14, or 16. So this means that the maze has always equal width and length and also it's always an even number. The goal for the robot is always located in the middle of the maze.
As an example the test maze 1 is provided by the problem that can be seen in the below picture:



In this maze the start point of the robot which is the corner on the bottom left side is shown with a red circle. The goal is in the middle of the map, so the 4 squares exactly in the middle of the maze are shown with blue squares.
Because the robot can take up to 3 moves in a time step the optimal solution for the robot in this maze looks as above in the picture, labeled by numbers. When the robot is in its initial place the sensor array gives the robot the input (0, 11, 0) which means that 11 fields in the forward direction of the robot are open and there is no wall till this number. But the left and right side of the robot are walls so the sensors represent the number 0 here.

Algorithms and Techniques

Solving a maze problem is a relatively old problem and a lot of efforts and endeavours are put in to solve this kind of problem. Based on the view of the robot there are many different techniques and algorithms that can be used to solve a maze. If the robot can see the whole maze, has the complete view of the maze, for example an A-star algorithm can solve the problem. Also other known algorithms can be used in this situation like dead-end filling algorithm.
But in the problem at hand the robot doesn't have the view of the complete maze. For this problem there are also some known algorithms that can solve the problem. These algorithm

have in common that they somehow based on some rules try to explore as much as possible of the map and find a good, but possibly not the best solution, to the problem. For our case we have a specific kind of scoring that should be considered while using these algorithms that the exploration phase shouldn't also take a long time.
One of the most famous algorithms for this problem, also used in the actual micromouse maze competition is the flood-fill algorithm.
In this problem I will use a variation of this algorithm that starts from the beginning and explores the maze based on a greedy algorithm that tries to get closer and closer to the goal in each step, but finds the path with the flood-filling algorithm. This solution will be discussed in detail in the Implementation part of the report.

Benchmark

For the benchmark what can be considered is that, a perfect robot exist that has the full map of the maze in it's memory. So the perfect robot can find the best and optimal solution based on known algorithms like A-star algorithm. After knowing the best solution based on the A-star algorithm the perfect robot adjusts this path with the fact that the robot can take more than 1 step at a time and so creates a path that is optimum in number of movements. An example for a perfect solution is represented in the data visualization part in this document for the test maze number 1 that could have been reached in 17 steps. In this problem the robot has also to explore the map, again let's consider the perfect robot that can see a square of the maze in 1 time step, which is highly exaggerated because sometimes in the dead ends you need to come back to the branching place, so the perfect robot needs n x n time steps to explore the complete map. For a 12 x 12 map it will be 149 time steps. It is known that the robot only has 1000 steps combined for the 2 runs. So the score based on the metics part will be: 149 / 30 + 17 = 21,8.
This benchmark will also be used on the 14 x 14 and 16 x 16 mazes, which in those cases it's even more underrated, because the robot would need 196 and 256 if there are no walls at all in the maze, but with walls and dead ends it will definitely take the robot more time steps to reach the goal.
This benchmark will be used to evaluate the implemented algorithm later in the report.

Methodology

Data Preprocessing

In this project the model of the environment, the robot with its sensor information and also the test mazes are given and there is no error or deviation in them. So for this project there is no need to preprocess any data. The data provided by the robot sensor in any stage of the runs in the maze can be directly used.

Implementation

In this project for the exploration phase, the first run of the robot in the maze, an algorithm based on the flood-filling algorithm is used in a combination with a heuristic that chooses the next square the robot moves closer to the goal.
The basic flood-filling algorithm, which is used by many contenders in the micromouse competition, works in a manner that the robot associates numbers to the different fields it can see from the current standing place. If the robot for example can reach a square with one move from where it's standing it associates a one to that squares and goes on with this numbering until it reaches the goal. After reaching the goal, all of the previous squares have a number associated to them that show how many steps it took the robot to reach there. So with backtracking the footsteps from the goal to the start the robot can find the best path it has seen.
In the project because the robot has more than one option to move toward, I have chosen a heuristic to decide which of the options the robot should pursue. This heuristic is that the move should bring the robot closer to the goal. For this I implemented a function that calculates the distance to the goal from that square, as an average to the 4 goal squares and returns the value.
To be able to use the flood filling algorithm, the robot has a dictionary called mapDict. This dictionary has as its key the number of movement that the robot is making and the value is a list of possible squares accessible from that point. Consider that the robot is in the starting point of the maze1 test maze that depicted in the Analysis part of this report. So the robot can go to the points [0,1], [0,2] and [0,3] at the beginning. So there will be be an entry created in the mapDict that is as follows: mapDict = {0: [[0,1], [0,2], [0,3]]}. Basically this dictionary is a good way to understand in which move we could have reached where. This feature is used in the backtracking phase from the goal to create the optimal path of the algorithm. After this the robot moves to the best net point. Also every move that the robot makes is saved in a list called visited, so that the robot knows which squares it has visited, in order to restrict the same moves by the robot, which lead to deadlocks. This was one of the problems I encountered first because if there is a loop in the map around the goal the robot will circle in this loop and never leave cause it wants to stay as close to the goal as possible.
The best next move that the robot moves to is decided by the heuristic discussed, which takes the robot closer to the goal. In the code there is a function implemented that calculates all the possible points that the robot can move and picks the one that is better than the others.
So with every move the robot tries to get closer to the goal until in some point it reaches the destination and then the exploration phase finishes and the robot returns (‘reset’, ‘reset) values that signal the start of the second fast run.
When the robot has reached the goal, the backtracking phase starts. In this phase the robot looks at the last visited point in its visited list. The first place in the dictionary that this point could have been reached is found and used for further references. Then the previous visited point is checked. If this point could have been reached before the reference that was just created this point will be included in the best solution, if not we move to the previous visiting point. So we go down the keys in the dictionary until we reach start.
The implementation described above is basically a combination of the flood filling algorithm with the heuristic goal oriented exploring. In the second run the robot then just follows the path found while backtracking.

So to summarize also the code structure, the code has 2 main parts. When the robot is learning and exploring the map, so the exploration phase of the first round, and the fast second run that the learning is false in the agent. When the learning is true the robot learns based on the discussed algorithm above. When the robot reaches the goal, the learning phase is ended and the optimal path is calculated and the robot stores the best moes it can take. Then in the 2nd run the robot uses the best moves toward the goal based on the backtracking. So the robot learns and uses the learned path to reach the destination.

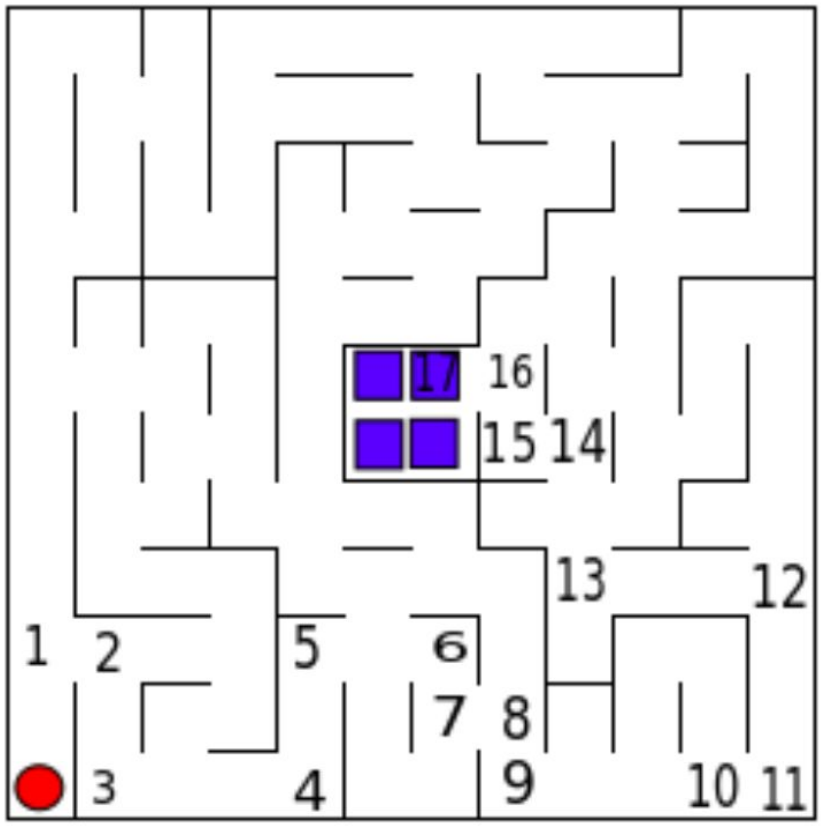
Refinement

The implementation process of the project started with familiarization with the environment. First the map structure of the maze and then the sensor inputs were studied. The relation between the functions learned and tried to play around with the constellation of the provided starter code. As in the case of the smartcab project in the nanodegree I started to make random moves with the robot so that the robot receives the sensor inputs and makes a random move and see what happens. Clearly the random movement in the maze was not a good idea and didn't lead to good results.
After that I started to explore the map with some heuristic that the agent should get closer to the goal in each step. So the movement wouldn't be random anymore, but with some goal in mind. But here the problem arouse that if there were loops in the maze the agent wouldn't be able to come out of the loops and deadlocks would happen in the maze. So a list of visited points was added to be able to avoid squares that had already been visited.
Also another important step that I took was that I tried to exploit the possibility of movements bigger than one step to exploit the map, that turned out to be not that effective as thought. The reason is that there are lots of uncertainties and you might jump a branch that could lead to a goal and also the explored step won't have that much order and the backtracking will be much more complex.

Results

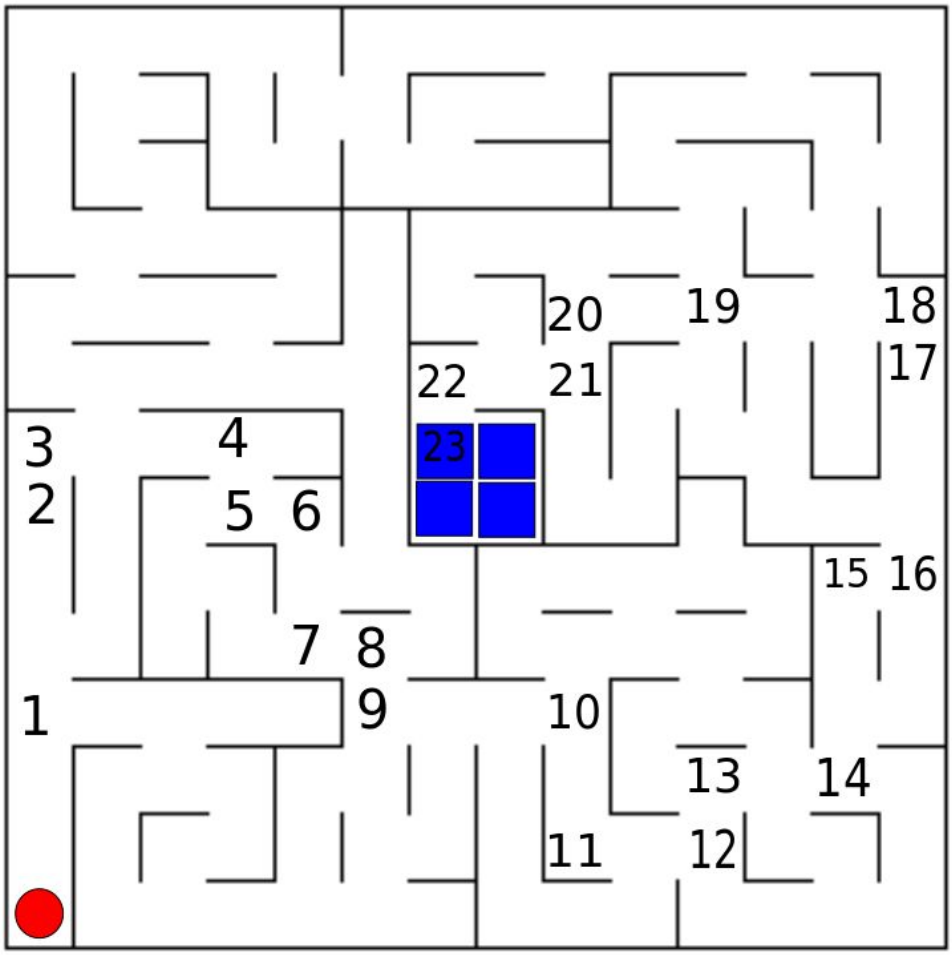
Model Evaluation and Validation

For this project, three available mazes accompanied the data given to the programmer. For this 3 mazes based on the benchmark that was discussed previously, the optimal solution has to be found. This was done by manual exploration of the maze by hand. The result for the optimal solution for the three mazes are as given below:
For the first maze the optimal solution is as follows:



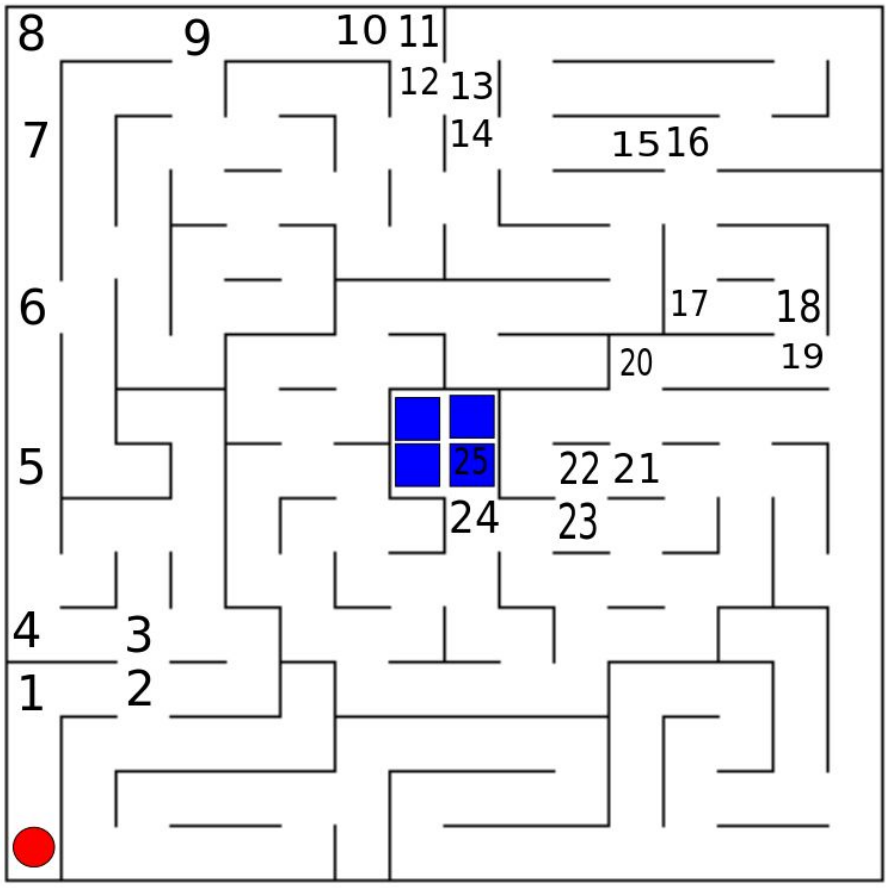
Where the red circle is the start, the 4 blue squares are the goal and the numbers in the maze represent the steps the agent has taken, for example number 1 means that in step 1 the robot has reached the corresponding square. So it can been seen that the optimal number of moves is 17.

The second mazes best solution is as follows:



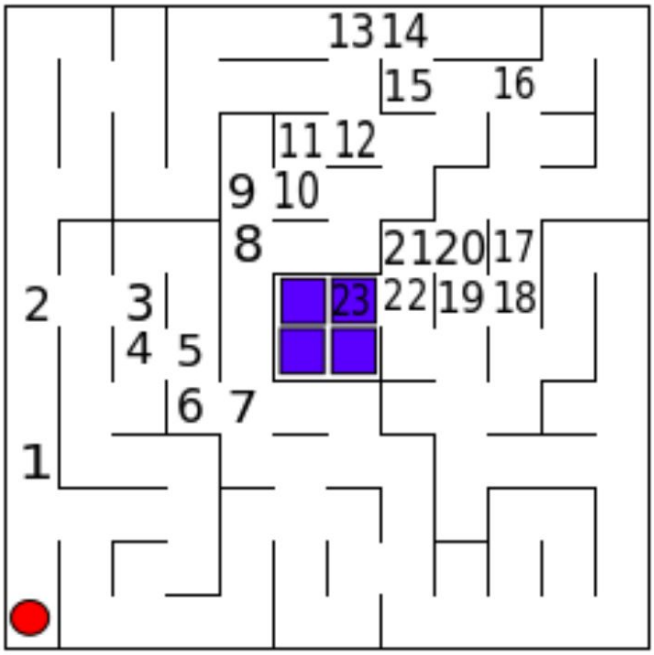
So the optimal number of moves in the second maze is 23.

Last but not least the third maze with its optimal solution looks as follows:

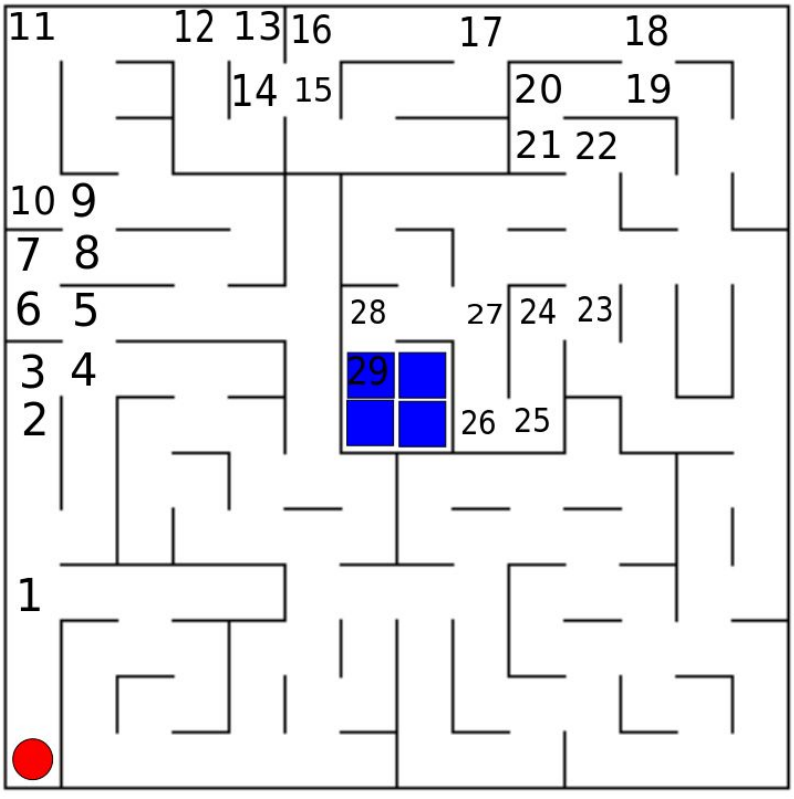


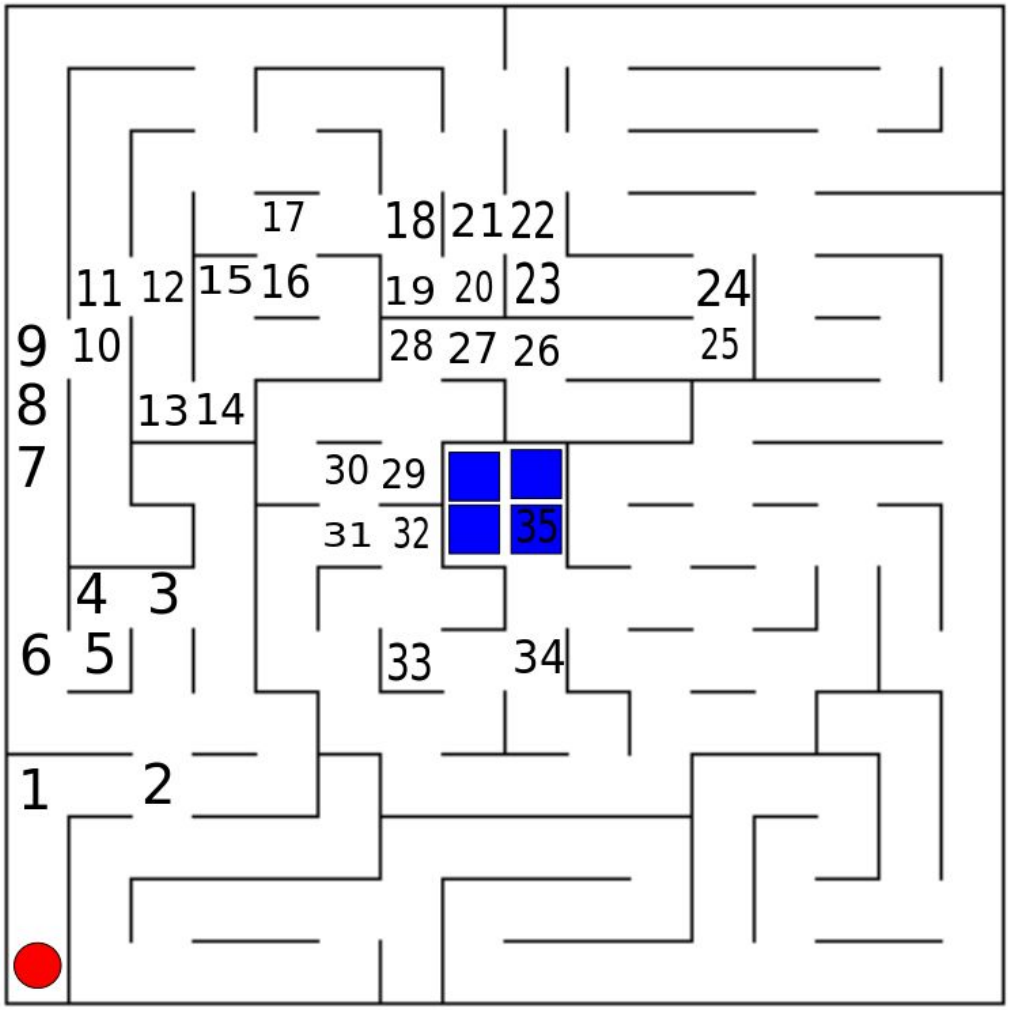
So the best solution has 25 steps to reach the goal in the middle of the maze.

After we know the best solutions for the three mazes, the implemented algorithm should also solve the problem for the same mazes. The results will be compared and checked with the benchmark in the Justification part.
So now it's time that the algorithm is run on the three mazes. The result is as shown as follows for maze1:



For mazes 2 and 3 also the solution looks as follows respectively in the pictures.





In the next part these results will be discussed in much more details and comparisons are provided to decide about the effectiveness of the provided algorithm.

Justification

Based on the result provided in the previous part, for the benchmark the following table can be concluded that shows the number of steps for the first run, the number of steps for the second run and score for that test maze.
The number of first run steps are basically the area of the maze, because as discussed in the benchmark part it is the least number of steps a robot needs to explore all the maze, if there is no dead end at all.

Test maze number	Number of first round steps	Number of second round steps	Score
1	144	17	21.8
2	196	23	29.53
3	256	25	33.53

To summarize what the algorithm did, we output the number of steps in the first run and the second round, but also the percentage of the maze coverage. Meaning that with the number of steps that the agent took how much of the maze became visible. The maze coverage percentage will basically be the explored squares divided by the total number of squares in the maze.
This metric will give us a rough idea that how many steps were really required for the agent to get full visibility of the map, to be able to have a better comparison with the benchmark case.
The results of the algorithm for the mazes are summarized in the following table:

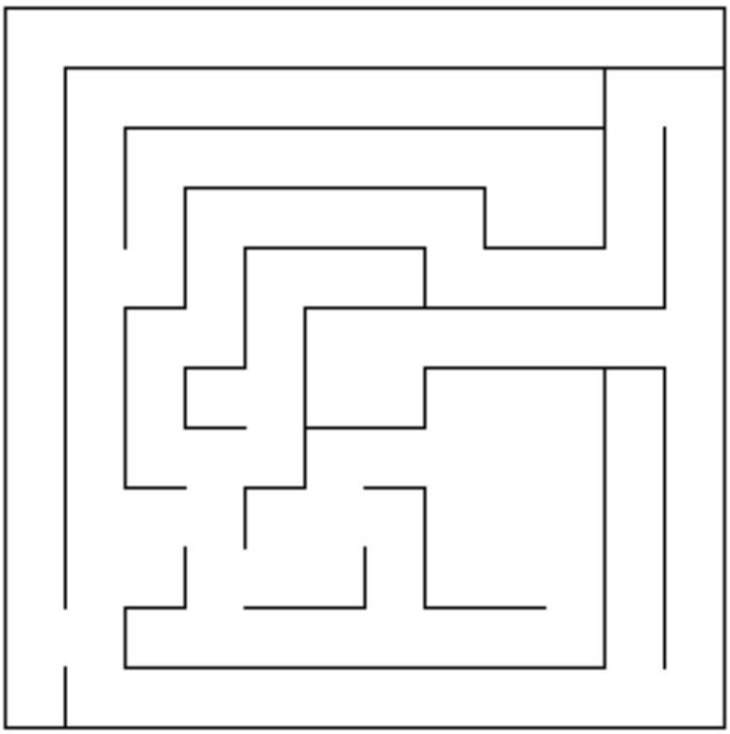
Test maze number	Number of first round steps	Maze coverage percentage	Number of second round steps	Score
1	95	45.833333333	23	26.167
2	301	88.265306122	29	39.03
3	101	29.296875	35	38.367

Based on the results from the table above we can see that the score are all worse than the perfect score reported based on the benchmark. But it can also been seen that the first impression about the perfect robot, that it can make all the maze visible after n x n steps, n being the dimension of the maze, is extremely farfetched. As can been seen in the first maze 95 steps are required to explore approximately 46 percent of the maze. Also for the other two mazes the number of explorations show that the score in the benchmark is lower than the real case would be.
So based on these results and the coverage percentage, and the fact that the benchmark is too idealistic, I believe that the algorithm is doing a decent job in scoring and is comparable to the benchmark that was defined.

Conclusion

Free-Form Visualization

In this section I am going to tackle one of the major disadvantages of my algorithm in my viewpoint. The problem with my algorithm is that it uses a goal greedy exploration movement, so it prefers any move that takes the agent closer to the goal over other movements. The problem is that if there is a path in the maze that at begining gets farther apart from the goal and then leads directly to the goal, and there is also another path that goes toward the goal and bends many times and in a much higher number of moves reaches the goal, the algorithm will find the latter one, not exploring the first path. So there will be high cost for the exploration and also a high cost in number of moves to reach the goal. The maze below, which I came up with, shows this problem in some degree.



The best case in this maze would be if the agent moves one square up and then right and then down and after that just follows the path to the goal. But the algorithm goes all the way up to the goal, ends in the deadend, gets back and goes up till reaching the goal. Of course this scenario can be even worsened. The result of the algorithm is summarized in the following table:

Maze number	Number of steps in first round	Maze percentage coverage	Number of steps in second round	Score
4	150	62.5	19	24

Reflection

In this project, a maze solving robot is coded. The robot has one run to explore the maze and one run to reach the goal as fast as possible. An appropriate metric and benchmark was also created to try to quantify the numbers, such that the effectiveness of the algorithm from the solution could be assessed.
There were many interesting and challenging aspects for me to this project. The first challenging part for me was to try to think without any information to try to simulate the robots behavior, in order to be able to turn the idea into a code. The problem here was that in the problem 3 example mazes are provided and when trying to find a good algorithm to solve this problem, you limit yourself to these cases, whereas you need to have a general approach. There are a lot of cases where differ between different mazes and sometimes your solution can work extremely well in one maze, while in the next maze your algorithm does a poorer job, which was shown in the freeform visualization maze4. This is in my opinion one of the most interesting and challenging things that I encountered in the maze project. Trying to think in a way that can give you the best solution in most of the cases and a great average answer on many many mazes that are possible.
Another great challenge I had was in the exploring phase, to try to take advantage of more than one movement. In the end, as explained in the Refinement section, I decided to explore the map one move at a time. The uncertainty that was in moving more than one step was extremely high and the risk of missing branches and not being able to get back in some places where the biggest challenges I faced, because I had to debug and recheck a great deal of data to understand what is going wrong.
I had some exposure to mazes where you had the complete view of the map, but solving a map with limited visibility was completely new for me and I got familiar with some methods and algorithms that can tackle this problem, but most important, I had to think a lot about this problem and I dare to say that I learned a great deal by trying to solve this problem.

Improvement

In the micromouse competition, physical robots are used in real world, rather than using a simplified model of the world. This automatically adds two different aspects. First a model is always a representation of the real world that omits some details and makes it easier to use. In this case the model implements everything in discrete domain. Time steps are discrete, the movement of robot is discrete, one to three boxes in directions, the rotation of the robot is discrete, either +90, or -90, or no rotation at all. In real world this is not the case at all. A physical robot has to move and control its speed in centimeters and centimeter per second for example which are in a continuous domain. The robot can also rotate in a continuous domain, for example 30 degrees. Also based on the youtube videos available from this competition a lot of robots move diagonally to save the rotation times which in the model is not allowed.
The other aspect is that in real world we have always some degree of error. This error can happen in the sensor input, the actuator output, some rigidity on the ground and etc. So the

input to the robot is not always reliable and the output of the robot movement doesn't always do exactly what it's supposed to do. This adds a lot of complexity to the problem.
Another thing that can be mentioned is that in physical world we have different vendors for the parts that constitute the robot. The motor, wheels, sensors can have different vendors with different specifications and errors, hence different price ranges, that should be chosen based on the results we want to see.