Felix Seckinger

Udacity Machine Learning Nanodegree

27.12.2017

Definition

Autonomous robots are needed for mapping regions. In this project a virtual robot explores different mazes autonomously by using navigation algorithms. The robot needs to identify the obstacles that are present in the maze and navigate through the maze. Therefor the robot needs to create a map by using a mapping algorithm.

The project is inspired by the micromouse competition. It is an event where engineers design and build autonomous robots. The goal of the competition is to solve different mazes. The mouse is allowed to map the maze and locate a shortest path to the goal during an exploration trial. After finishing the mapping of the maze the robot starts an optimization trial where the robot should navigate from start to goal in the shortest time possible. In this project the micromouse will navigate through a virtual maze.

Problem Statement

The main goal of the project is to teach the robot mouse how to get to the center of the maze as fast as possible. The robot is starting at the left bottom corner of the maze. From this starting point the robot uses as less moves as possible to reach the goal in the shortest time possible. The maze consists of an equal number of squares. The height and width of the maze has a certain number of rows and columns. This equal number is 12, 14 or 16. The starting point is defined by walls on the left, right and back sides. Therefor the robot can only move forward in a first step. The goal is defined by a 2x2 square.

In a first run the robot is exploring the maze. Within this trial the robot explores the structure of the maze and finds possible routes to the goal area. A trial is successful if the robot mouse reaches the goal. But after reaching the goal the robot is allowed to continue exploring the maze.

On the second run the robot mouse is using all the collected information and attempts to travel to the goal in the shortest time possible. The performance of the robot mouse is measured by two things. The total number of steps in the first trial is counted and

divided by 30. The total number of steps to reach the goal in the second trial. The mouse is only allowed to make 90 degree turns. In a single movement the mouse is allowed to take three steps. After 1000 steps the trial is abandoned.

The robot mouse is able to detect whether there is an opening to an adjacent cell or if there is a wall blocking the path. The robot can also turn clockwise or counterclockwise in 90° intervals and can decide to move up to 3 consecutive steps forwards or backwards. After each step the robot updates its surrounding.

Metric

Like mentioned before the main metric to quantify the performance of the model is:

$$Score = [Number\ of\ Steps\ in\ optimization\ trial] + [Number\ of\ Steps\ in\ exploration\ trial/30]$$

I will explain the score in a short example. Let's assume the robot took 200 steps in the exploration trial and 20 in the optimization trial. The score would be 20 plus 6.67 which is 20.67.

This metric takes the exploration and optimization trial in account. But the optimization trial has a much higher impact because it is the indicator for the solution. The lower the score the better is the model. Therefor the goal is to minimize the score.

Analysis

Data Exploration

Provided Starter code:

- Robot.py defines the robot class (main script for project where changes are made)
- Tester.py testing the ability of the robot to navigate in maze
- Showmaze.py visual layout of maze
- Test_maze_##.txt sample mazes to test the robot

The test_maze_##.txt file provides information about the layout of the maze. The first line of the row is a number (12,14 or 16) and defines the length of the side of the maze. In the following lines numbers from 1 to 15 are represented. Those describe the

position of the wall of each cell. Each number is a four-bit number and also represents if an edge is closed (walled) or open (open).

```
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 test_maze_01.txt

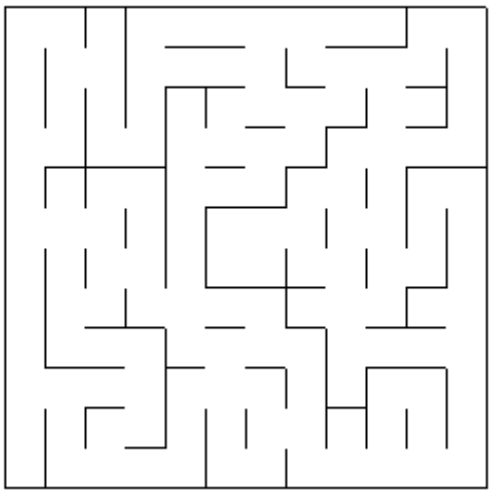

Figure 2 Representation of test_maze_01

The 1s register represent the up-facing side. 2s register describe the right-facing side. 3s register define the left-facing side and 4s register the down-facing side. With this background numbers are calculated which describe each cell of the maze.

Exploratory

A heuristic function is used to help the A* algorithm estimating the distance from the current position to the goal. Each cell is labelled with a number from 0 (goal) to 10. The goal is labelled with a 0. Each step away from the goal increases the label by 1.

10	9	8	7	6	5	5	6	7	8	9	10
9	8	7	6	5	4	4	5	6	7	8	9
8	7	6	5	4	3	3	4	5	6	7	8
7	6	5	4	3	2	2	3	4	5	6	7
6	5	4	3	2	1	1	2	3	4	5	6
5	4	3	2	1	0	0	1	2	3	4	5
5	4	3	2	1	0	0	1	2	3	4	5
6	5	4	3	2	1	1	2	3	4	5	6
7	6	5	4	3	2	2	3	4	5	6	7
8	7	6	5	4	3	3	4	5	6	7	8
9	8	7	6	5	4	4	5	6	7	8	9
10	9	8	7	6	5	5	6	7	8	9	10

Figure 3 Representation of Heuristic grid for test_maze_01

The robot mouse discoveres the maze in the exploration trial. The mouse is not necessarily ending its exploration trial after finding the goal. In order to discover a significant amount of the maze the mouse is continuing the trial until it visited a certain amount.

Algorithms

There are several techniques and methods to solve the maze. I already described several techniques that are feasible to apply for this problem. Some are more appropriate to use for exploration and some for optimization trial. I decided to use the A*-algorithm.

A*: Is recognized as best first search algorithm which combines the advantages of Dijkstra and Greedy best first search algorithm. It is detecting all possible solutions and chooses the shortest path to the goal. A* favores vertices that are close to the starting point (Dijkstra) and favores vertices that are close to the goal (Greedy best first search). Typically The algorithm represents the costs for a path from starting point to any vertex and the estimated cost from vertex to goal.

The algorithm will decide at each step which of the possible paths to take and project the total distance to the goal. It will then choose the path with minimum cost. The heuristic usually underestimates the costs of the optimal path to the goal and is used to focus the search. I decided to use the A* algorithm for the exploration trial.

The optimization trial is implemented with dynamic programming. This is very similar to A*. If the map of the maze and the goal is provided dynamic programming sets an

optimal path from any starting point. The algorithm will define an optimal action at each cell of the maze.

Benchmark

The benchmark model will be scored by the performance score (problem statement). The score is the sum of number of steps in trial 2 taken plus number of steps in trial 1 divided by 30. By limiting the possible steps t 1000 the performance of the method is going to be very basic. During this 1000 steps the robot is making random decisions where to turn and make the next step. If he is able to find a solution in less than 1000 steps a new benchmark is set to the number of steps in trial 1. The robot should then find an optimized solution in trial 2.

The given example mazes have optimal routes that consist of 17, 23 and 25 steps. A robot needs to find the goal in less than 1000 steps for each of the three different mazes. The optimal routes 17, 23 and 25 will be used as a benchmark for the optimization trial.

For the exploration trial the benchmark model has to uncover every cell of the grid. This would need a lot of steps. Especially because the model needs to visit cells multiple times in order to uncover a cell. To calculate an appropriate number of steps for the benchmark model I multiply the number of cells of the maze by 2.5.

Score benchmark maze 1: 17 +(144*2,5)/30 = 29

Score benchmark maze 2: 23 + (196*2,5)/30 = 39.33

Score benchmark maze 3: 25 (256*2,5)/30 = 46.33

Methodology

Implementation

For this project is no data preprocessing needed. Therefor the implementation can be done directly. The implementation of the project will be done by following this procedure:

- Interpreting the surrounding of the robot mouse
- Movement during Exploration trial
- Identifying optimal path to goal

- Starting optimization trial

The robot mouse gains knowledge of the structure of the maze in the exploration trial. The robot stores its knowledge in a 2-dimensional array. This is named maze grid. The knowledge is stored in the wall_locations method. From this list the robot will know where walls and openings are and also which parts of the maze are undiscovered.

Each step the robot senses information that describe the current location and what the robot is facing. This information is stored in a four-bit number (see Data Exploration). The undiscovered parts (0's) of the maze are also stored in an additional two-dimensional array.

During the robot navigates through the maze grid the four-bit number is updated at each step.

Movement during exploration trial

Now is defined how the robot should navigate through the maze during the exploration trial. The robot should prefer undiscovered cells when moving in a general direction toward the goal. I implemented a strategy with a heuristic grid to steer the robot towards the goal area.

Here I created a Heuristic grid that is related to Artificial Intelligence. This grid helps the robot to steer through the maze during the exploration trial maze and the goal.

To define the robots path during the exploration trial the method 'next_move' is used. Within this method is described how the robot is guided to the goal while considering the walls and its current location. The method makes sure that the robot is not stuck. When it is it moves backwards and tries to find a single open path. If there is one the robot rotates to this path. If there are more than one open paths the robot consults the heuristic grid.

In the next step I make sure that the robot finds the goal area during the Exploration trial and how much of the maze should be discovered after the goal was found. Therefor the robot class contains an if statement which asks at the end of every step if the goal area is reached. If the goal area is reached the 'self.discovered_goal' statement is set True and the robot continues discovering the map till a certain area of the map is discovered.

Identifying optimal path to goal

For this reason I implemented the method ‘value’ which defines the optimal path for a given map of the

During the exploration trial the robot finds possible solutions for the maze. One of these solutions is the optimal path which has to be identified in order to make an optimized trial. The method creates a policy grid by considering all the locations of the walls. The policy grid describes the optimal actions for each cell.

In order to make the ‘value’ method work I defined a path value. This path value has the same size as the maze. Every cell in this path value is initially labelled with 99 and describes how far the cell is away from the goal area.

Starting optimization trial

The optimization trial can be started if there is defined when and how to end the exploration trial. In this scenario the robot mouse has to find the goal area and continue exploring the maze until 75% of the maze are discovered. Then the value method is called. ‘Movement’ and ‘rotation’ reset and the optimization trial can start.

Refinement

At first I allowed the robot to end after the goal was found during the exploration run. But then the robot did not explore much of the maze and tended to miss more optimal paths. So I made the robot discover a given percentage of the maze before ending the exploration run and starting the optimization run.

Results

I implemented the A*-algorithm for exploration trial and dynamic programming for optimization trial. I also created a script ‘globalvariables.py’ to store the global dictionaries like dir_sensors, dir_move, ...

To make the robot uncover as many cells of the maze as possible I allowed the robot to continue exploring after the goal was found.

The model I built is able to find the goal I different types of mazes. This means the model is robust. Because I want the model to discover a certain percentage of cells it can discover different optimal paths to the goal. The model gets robust and can be used for different mazes. At the end the model yields even better results than the benchmark model.

The results of the model are figured out in the command-prompt.

```
steps taken in exploration trial  311 [0, 0, 3]
robot is at location:  [10, 10]
robot has explored 99.31% of the maze.

steps taken in exploration trial  312 [6, 2, 0]
robot is at location:  [9, 10]
robot has explored 99.31% of the maze.

steps taken in exploration trial  313 [1, 5, 0]
robot is at location:  [9, 9]
robot has explored 99.31% of the maze.

steps taken in exploration trial  314 [0, 0, 0]
robot is at location:  [10, 9]
robot has explored 100.00% of the maze.

robot dead end. reverse
robot ended exploration trial. Optimization trial starting.
Goal: [5, 6]
```

Figure 4 result of exploration trial

```
 Maze Grid
[[ 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]]

 Path Grid
[[2 2 4 3 3 3 4 2 2 1 1 1]
 [2 2 3 2 3 2 5 2 1 1 2 1]
 [3 2 2 2 2 3 3 2 2 2 3 1]
 [2 2 3 2 1 2 2 2 2 2 2 1]
 [2 2 2 2 4 3 3 3 2 1 1 1]
 [2 2 4 3 2 2 2 2 3 2 1 1]
 [2 2 2 3 2 2 3 2 3 2 1 1]
 [2 2 2 2 2 3 4 3 1 2 3 4]
 [4 2 2 4 5 3 2 3 2 2 3 2]
 [2 2 2 2 2 3 3 2 3 2 2 2]
 [2 2 2 2 2 2 2 2 1 1 1 1]
 [2 2 2 3 4 2 2 2 1 1 1 1]]
```

Figure 5 Maze grid and path grid.

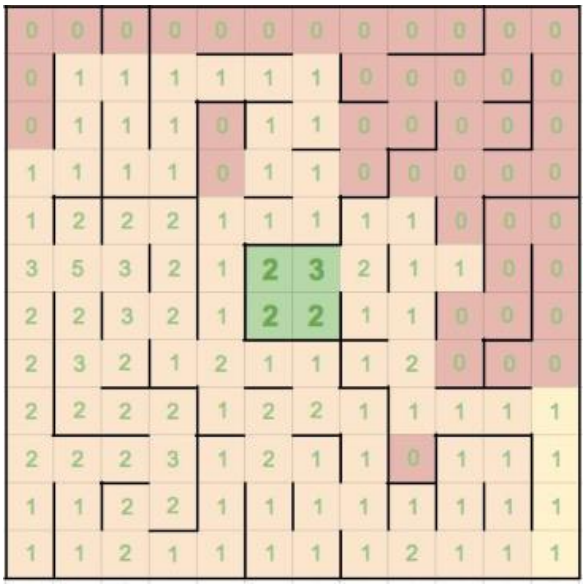

Figure 6 path grid test_maze_01

```
 Path Value
[[31, 30, 29, 28, 27, 26, 25, 26, 25, 26, 25, 24], [26, 27, 28, 25, 24, 25, 24, 25, 24, 23, 2
2, 23], [25, 26, 29, 26, 23, 22, 23, 24, 19, 20, 21, 22], [24, 27, 28, 27, 20, 21, 22, 23, 18
, 17, 16, 15], [23, 22, 21, 20, 19, 18, 17, 16, 15, 16, 15, 14], [20, 21, 20, 21, 20, 1, 0, 1
5, 14, 13, 14, 13], [19, 18, 19, 20, 21, 2, 1, 14, 13, 12, 13, 12], [16, 17, 18, 19, 6, 3, 2,
 3, 12, 11, 10, 11], [15, 16, 7, 6, 5, 4, 5, 4, 5, 10, 9, 12], [14, 15, 16, 7, 6, 7, 6, 7, 6,
 7, 8, 13], [13, 14, 15, 8, 11, 8, 9, 10, 7, 8, 9, 10], [12, 11, 10, 9, 10, 11, 12, 11, 8, 9,
 10, 11]]

 Policy Grid
[['^', '^', '>', '^', '^', '^', '>', 'v', '>', 'v', '^', '>'], ['>', 'v', 'v', '^', '>', 'v',
 '>', 'v', '^', '^', '>', 'v'], ['>', 'v', '<', '<', '^', '>', 'v', 'v', '>', 'v', 'v', 'v'],
 ['>', '<', 'v', '<', '>', 'v', 'v', 'v', '^', '^', '>', '>'], ['^', '^', '>', '^', '^', '^',
 '^', '>', '>', 'v', '>', '>'], ['>', 'v', '>', '<', '<', '^', '~', '>', '>', '>', '>', '>'],
 ['^', '>', 'v', '>', '<', '<', '<', '^', '>', '>', 'v', '>'], ['>', 'v', 'v', 'v', '>', '^',
 '<', 'v', '^', '>', '>', 'v'], ['>', 'v', '^', '^', '^', '<', 'v', '<', 'v', '^', '>', '<'],
 ['>', 'v', 'v', '<', '<', 'v', '<', 'v', '<', 'v', 'v', '<'], ['>', 'v', 'v', '<', '>', '<',
 'v', 'v', '<', '<', '<', 'v'], ['^', '^', '^', '<', 'v', 'v', 'v', '<', '<', 'v', 'v', '<']]
```

Figure 7 path value and policy grid.

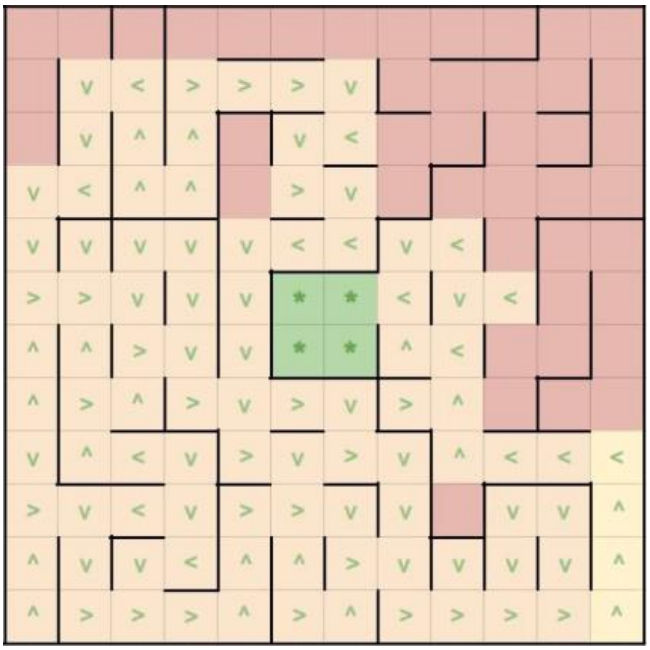

Figure 8 policy grid test_maze_01

```
Ending first run. Starting next run.
Starting run 1.
Optimization Trial Step #:  0 [0, 11, 0] [0, 0]
Optimization Trial Step #:  1 [0, 9, 3] [0, 2]
Optimization Trial Step #:  2 [0, 2, 2] [1, 2]
Optimization Trial Step #:  3 [3, 0, 0] [1, 0]
Optimization Trial Step #:  4 [2, 0, 0] [4, 0]
Optimization Trial Step #:  5 [0, 0, 2] [4, 2]
Optimization Trial Step #:  6 [0, 0, 2] [6, 2]
Optimization Trial Step #:  7 [1, 1, 0] [6, 1]
Optimization Trial Step #:  8 [2, 0, 1] [7, 1]
Optimization Trial Step #:  9 [4, 0, 0] [7, 0]
Optimization Trial Step #:  10 [2, 1, 0] [10, 0]
Optimization Trial Step #:  11 [7, 0, 0] [11, 0]
Optimization Trial Step #:  12 [3, 4, 0] [11, 3]
Optimization Trial Step #:  13 [1, 0, 5] [8, 3]
Optimization Trial Step #:  14 [1, 3, 0] [8, 5]
Optimization Trial Step #:  15 [0, 0, 2] [7, 5]
Optimization Trial Step #:  16 [2, 1, 0] [7, 6]
Goal found; run 1 completed!
Task complete! Score: 27.500
```

Figure 9 optimization trial

The results are different for the minimum percentage the robot has to uncover. The dependency is shown in the following table.

Minimum Coverage	Dimensions	Maze 1	Maze 2	Maze 3
	Benchmark	29	39.33	46.33

25	Exploration Steps	141	161	197
	Optimization Steps	17	31	29
	Score	21.70	36.433	35.567
50	Exploration Steps	141	170	197
	Optimization Steps	17	31	29
	Score	21.7	36.667	35.567
60	Exploration Steps	141	229	267
	Optimization Steps	17	28	29
	Score	21.7	35.633	37.9
75	Exploration Steps	170	279	325
	Optimization Steps	17	28	29
	Score	22.67	37.3	39.833
100	Exploration Steps	315	659	867
	Optimization Steps	17	26	27
	Score	27.5	47.967	55.9

Table 1 Results

The results of the model seem to be quite good. The score and the number of steps outperformed the benchmark model. Only the optimal number of steps for maze is not reached (28 instead of 23).

Improvement and Conclusion

I am currently doing my master in control technology. Working in this field I get in touch with robots and different automation processes. Because of this background I decided for this project because it is a topic that is really affecting me. But I never

wrote an algorithm like this before so I was very happy to see the code working for the first time. Besides I can image possible applications in factories that are becoming intelligent.

The whole project flow can be divided in the following steps: Interpreting robot location and heading, Updating the map, Navigate the robot to the goal during exploration trial, Update robots location, Continue exploring the maze, Find optimal path to goal, Optimized run.

The solution is very computational. I would simplify the model in a next step. This could lead to a model that identifies optimal paths easier and takes multiple steps at once. The model's robustness could be increased by preventing the robot from getting stuck.

The implementation of another algorithm could also lead to better solutions for different mazes. Different type of mazes may require different exploration algorithms in order to find a best solution for this specific problem. I considered some other algorithms that could be used for this problem.

- Wall Follower: The mouse is asked to take every turn right or left possible. This should eventually lead to the goal. But the assumption is that the mouse is not caught in a circular loop.
- Dijkstra's: Is an algorithm to find a shortest path between nodes in a graph.
- Dead Reckoning: This algorithm makes random decisions at each step.
- A* Is recognized as best first search algorithm. It is detecting all possible solutions and chooses the shortest path to the goal.
- Breadth First Search: Is an algorithm for searching tree or graph data structures. The closest neighbours are considered instead of looking for next level neighbours.
- Depth First Search: Is an algorithm for searching tree or graph data structures. It starts at the root and explores as far as possible along each branch.
- Flood Fill: The connected area to a given node is presented in a multidimensional array.

The implementation of one these algorithms would be interesting. The performance of the algorithms could be compared. Certain algorithms are considered to work better for certain maze styles. The best solution may be to allow the robot to choose one of these algorithms depending on the maze that has to be solved.

Sources:

- Udacity Capstone Proposal template

https://github.com/udacity/machinelearning/blob/master/projects/capstone/capstone_proposal_template.md

- Robot Motion Planning Capstone Project

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

- Micromouse Competition

http://micromouseusa.com/ and

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

- A* algorithm, Breadth-first-search, Width-first-search

http://bryukh.com/labyrinth-algorithms/

- Solving Mazes autonomously

http://soe.rutgers.edu/sites/default/files/imce/gov2017/Autonomously%20Solving%20Mazes%20with%20Robots.pdf