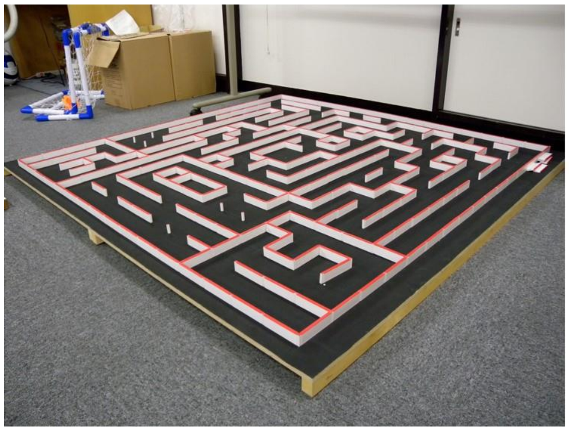
# Intelligent Mouse

Lei Mao

2/24/2017

## Introduction

Micromouse is a contest where small robot mice (micromouse) solve a maze (**Figure 1**) [1]. It is very popular in US, UK and Japan among the young juniors who are interested in designing robots and programming artificial intelligence. In Micromouse contest, the players are going to test their micromouse to solve the maze.



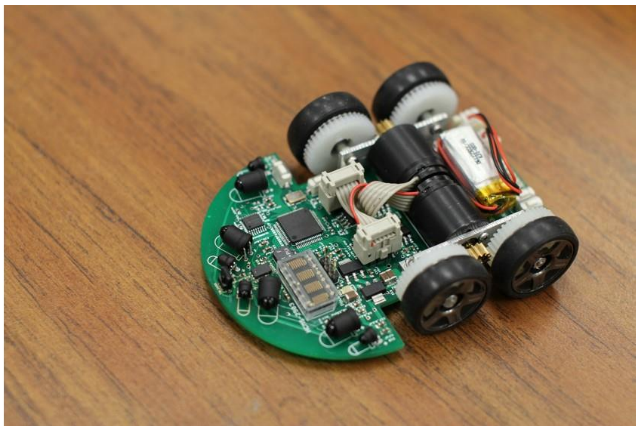


**Figure 1.** Example of maze (left) and robot mouse (right) in Micromouse Contest.

According the "APEC Micromouse Contest Rules" [2], the players were informed the size of maze, usually 16 x 16, before the event. In addition, the start square is always at one of the four corners with some specifications, and the destinations are always four squares located at the center of the maze.

Here I also cited some key rules from the "APEC Micromouse Contest Rules" to help the readers understand the contest. During the contest, the robot mouse shall be put at the start square by the player in the first place and the micromouse will be thought as "activated". The basic function of a micromouse is to travel from the start square to one pf the destination squares. This is called a "run". The time it takes is called the "run time". Traveling from the destination square back to the start square is not considered a run. The total time from the first activation of the micromouse until the start of each run is also measured. This is called the "maze time". The score for each run is calculated by adding the time for each run to 1/30 of the maze time associated with that run and subtracting a 2 second bonus if the micromouse has not been touched yet (For example assume a micromouse, after being on the maze for 4 minutes without being touched, starts a run which takes 20 seconds; the run will have a "handicapped time" of: 20 + (4 × 60 / 30) - 2 = 26 seconds). The run with the fastest "handicapped time" for each micromouse shall be the "official time" of that micromouse. Each contesting micromouse shall be subject to a time limit of 7 minutes on the maze. Within this time limit, the micromouse may make up to 7 runs.

From the rules, we can infer that if your micromouse is able to move to the destination efficiently in a short period of time right after activation, you are very likely to get a very good “official time”. For example, if your micromouse is extremely “lucky” that it happens to go to the destination via the theoretical shortest path in 5 seconds, your “handicapped time” for this run will be 5 + (5 / 30) – 2 = 3.17 seconds. However, in the late stage of the game, say after staying in the maze for around 6 minutes, even if your mouse finds the theoretical shortest path and goes to the destination in 5 seconds, your “handicapped time” will be pretty bad. For this run, the “handicapped time” will be 5 + (6 × 60 / 30) – 2 = 15 seconds. From the examples above, we know that in order to win the contest, in addition to luck and the mechanical agility of micromouse, your micromouse has to be extremely efficient in learning and analyzing the maze during the movement.

In this project, in order to develop and test my micromouse with maximized learning and analytical abilities, the mazes were virtualized and the contest rules were simplified. Here I am going to present a micromouse that is extremely efficient in exploring the maze and figuring out best strategy going the destinations. It collects maze information from its sensor, maps maze in its memory, calculates the most efficient exploration strategy, and determines best series of actions toward the destinations using artificial intelligence techniques.

## Original Problem Description

This “virtual maze” problem was initially proposed by the Udacity Nanodegree Capstone project. Instead of using real maze, virtual mazes of size 12 × 12, 14 × 14 and 16 × 16 were employed for tests (**Figure 2**). The start square (green) is always located at the left-bottom corner of the maze and the four destination squares (red) are always located at the center of the maze. The micromouse has three sensors which allow it to detect its distances to the obstacles in the front, at the left side and right side. The actions of mouse have also been simplified. In each time step, the micromouse collects distance information from the sensors, chooses moving direction, and moves forward a distance from 0 to 3 units. If the micromouse hits the wall, it stays where it is. After each movement, one time step has passed and the next time step begins.

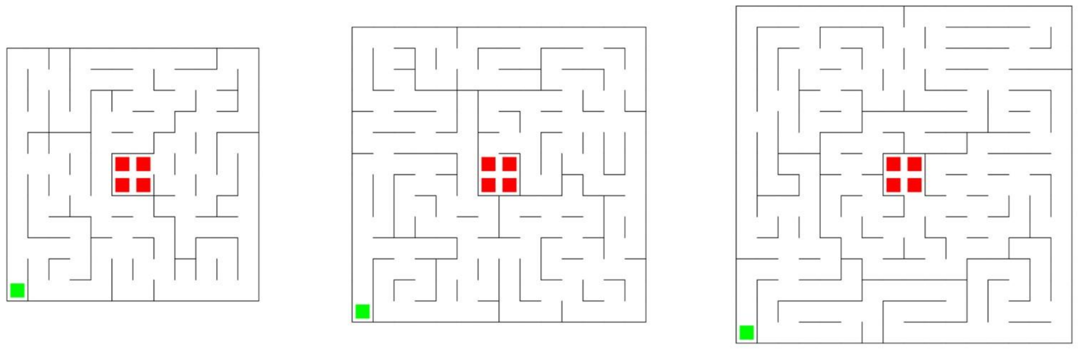


**Figure 2.** Maze 1, 2 and 3 (from left to right) for tests. The start location were labeled using green square and the destinations were labeled using red squares.

The micromouse is allowed to have two runs. In the first run, the micromouse explore the maze and return to the start square. It does not matter if the micromouse did not reach the destination. In the second run, the micromouse shall reach the destination and the simulation ends. In both runs, the time steps were counted. The scoring function is analogous to the one used in real Micromouse contest. The final score is equal to the number of time steps required to execute the second run plus one thirtieth the number of time steps required to execute the both runs. For examples, if it takes 100 time steps to explore the maze and return, and 20 time steps to reach the destination, the score is 20 + (100 + 20) / 30 = 24.

## Problem Generalization

From my point of view, the "virtual maze" problem is not a generalized problem and is therefore not very applicable in real world. The players could use the information, such as the size and shape of maze, the locations of start and destination squares, which were provided before the game, to improve the performance of micromouse. However, real situation could be more complex. If the real world is a maze, the maze could be any kind of shape and size and the start and destination squares could be anywhere. When the micromouse was put into such maze, it should be able to figure out the best strategy without using any information provided from outside. Such micromouse would be powerful in real world applications, such as Mars exploration and driving navigation. So I added several restrictions to the original "virtual maze" problem:

1. The maze could be any shape and size. The player could not "inform" the micromouse any maze information.
2. The referee designates the start and destination squares.
3. There is certain sign on the destination square. The micromouse only knows it has reached the destination when it is exactly on it.

Developed for this challenging and restricted "virtual maze" problem, my micromouse only takes information from its sensors and the player does not have to provide any maze information beforehand. It can start from anywhere in the maze and reach any destination that the referee designates with high efficiency. The shape of maze is not going to affect the efficiency of micromouse as well.

The only thing I did to improve the hardware of the micromouse is that I added a sensor at the back of the micromouse to detect the distance to obstacle at the back and another sensor to detect destination sign. With this design, the micromouse has some advantages in collecting the maze information when it is firstly put in the start square which could be anywhere in the maze (The micromouse with three sensors will have to make turns to collect the maze information when it is firstly put in the start square).

If the micromouse tries to explore the maze from the left-bottom corner of the maze according to the rules described in the original "virtual maze" problem, the maze information it collected from four sensors is exactly the same to that collected from three sensors throughout the contest. Therefore, I would not consider having four sensors "cheating".

## Micromouse Design

There are two runs during the contest. In the first run, the micromouse needs to collect more maze information in less time. In the second run, with the knowledge of learned maze information, the micromouse calculates the optimal path to the destination and the corresponding series of actions that minimizes the number of time steps. So the micromouse requires an efficient exploration strategy during the first run, and a maze solver which gives optimal solutions during the second run. Because it will require the maze solver to make efficient exploration strategy during the first run, I am going to introduce the maze solver first.

## Maze Solver Using Bellman Equation

Given a maze, a start square and destinations, find the best path to the destinations. This is actually a Markov Decision Process (MDP). In our "virtual maze" problem, each square in the maze is a state of MDP. In each state, the micromouse could choose a direction from "up", "down", "left" and "right" and a movement step length of 0, 1, 2, or 3. The combination of direction and movement step length is the actions available at each state. The MDP can be solved by Bellman Equation (**Equation 1**), sometimes referred as Dynamic Programming. In Bellman Equation, because the model is deterministic in our problem, $T(s, a, s')$ can only be 0 or 1.

$$U(s) = R(s) + \gamma \max_{a} \sum_{s\prime} T(s, a, s') U(s') \tag{1}$$

$s$: $the\ state\ during\ MDP$

$a$: $the\ action\ taken\ at\ state\ "s"\ during\ MDP$

$s'$: $the\ next\ state\ after\ state\ "s"\ during\ MDP$

$\gamma$: $decay\ factor$

$R$: $reward\ function$

$T$: $transition\ function$

$U$: $utility\ function$

$R(s)$: $the\ reward\ of\ being\ at\ state\ "s"\ during\ MDP$

$T(s, a, s')$: $the\ probability\ of\ being\ at\ state\ "s'"\ from\ state\ "s"\ after\ taking\ action\ "a"\ during\ MDP$

$U(s)$: $utility\ value\ of\ being\ at\ state\ "s"$

To solve Bellman Equation, one of the traditional way is to do value iteration (). We initialize the equation with arbitrary utility value for each state. Then we update the utility value for each state using the following equation until the utility value for each state converges. The converged utility values become the true utility values.

$$\widehat{U}_{t+1}(s) = R(s) + \gamma \max_{a} \sum_{s'} T(s,a,s')\widehat{U}_t(s') \tag{2}$$

$\widehat{U}_t(s')$: *utility value assigned for state "s" at time step "t" during value iteration*

Because the goal of micromouse is to reach the destination in minimum time steps, I assigned the reward for destination squares 5 × length × width of the maze. If the maze is not rectangle, squares with four walls were added to the maze to make it rectangle. The reward for squares with four walls, if there is any, were assigned -5 × length × width of the maze. The rewards for the rest of squares were all assigned to -1. The decay factor $\gamma$ was assigned 0.95. The utility value for each state was initialized with 0. They can always converge during value iteration.

The optimal series of the actions to reach the destination is the one that maximize the long-term utility values. Mathematically, in each state, the optimal action can be described as follows:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')U(s') \tag{3}$$

$\pi^*(s)$: *the optimal action at state "s"*

Starting from the start square, the optimal series of actions could be found using the above equation sequentially.

Here, given the information of maze 1, 2 and 3 (**Figure 2**), I was able to find the optimal series of actions and path from the start square to the destinations that minimize time steps using Bellman Equation (**Figure 3**, **Table 1**). It should be noted that the path that minimize the time steps are not necessarily the same to the shortest path in length.

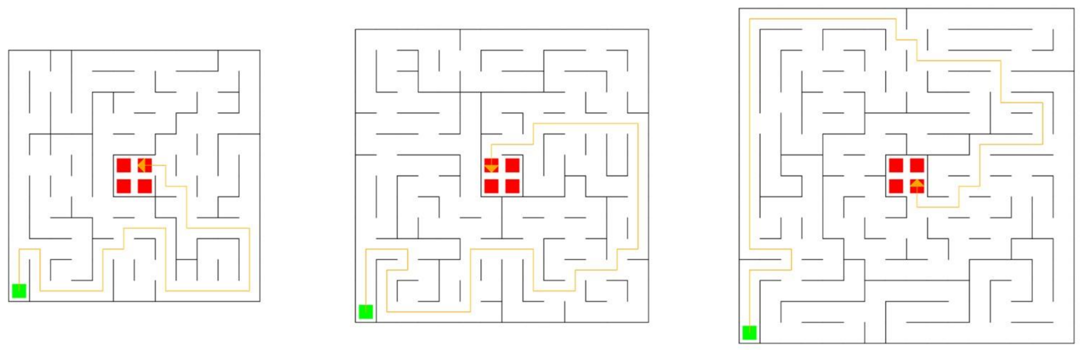


**Figure 3.** The optimal path corresponding to minimum time steps for maze 1, 2 and 3 (from left to right). The yellow line depicts the path calculated from Bellman Equation.

**Table 1.** The summary of optimal path and actions.

| Test Maze | Minimum Time Steps | Path Length | Theoretical Shortest Path Length |
|---|---|---|---|
| 1 | 17 | 32 | 30 |
| 2 | 22 | 43 | 43 |
| 3 | 25 | 49 | 49 |

Similarly, in Micromouse contest, during the second run, the micromouse should be able to find the optimal series of actions and path from the start square to the destinations that minimize time steps using the maze information learned from the first run.

## Maze Explorer Using Maze Solver

In the first run when the micromouse was firstly put into the maze, the micromouse depicts a “partial maze” using the information collected from the sensor, stores it in memory, and updates it routinely during exploration. This partial maze is a map of squares with wall information of each square. If the micromouse is not sure whether there is certain wall at certain direction in certain square, it always assumes there is a wall. In addition, the micromouse holds a “passport” to document the square grids it has been to. If the micromouse has never visited certain square in the maze, it may only have partial wall information. For example, if the micromouse has never visited square (0, 1) but it has visited square (0, 0), it knows whether there is any wall at directions “up” and “down” in square (0, 1). However, the micromouse is uncertain about the wall information at directions “left” and “right” in square (0, 1).

After updating the “partial maze” and “passport”, the micromouse has to decide where to go and what action to take. All the unvisited square are the destination candidates. However, to explore the maze systematically, the intuition is that the micromouse should choose the one that costs minimum time steps. To calculate the minimum time steps going to each destination, Bellman Equation is employed on the “partial maze” the micromouse just learned. If there are multiple such candidates which cost the same minimum time steps, the mouse shall choose randomly among the candidates that are closest. In my micromouse settings, the micromouse calculates its “Manhattan Distance” to all candidates to find the closest candidates. It should be noted that the square grid that costs minimum time steps are not necessarily the closest one. For example, if the micromouse is located in grid (0, 0) and it has never visited any other grids. It can go freely to grid (0, 1), grid (0, 2), grid (0, 3), grid (1, 0), grid (2, 0) and grid (3, 0) within one time step, grid (0, 4) and grid (4, 0) within two time steps. These grids are all the destination candidates since they have not been visited. The decision the micromouse will make is to go to either grid (0, 1) or grid (1, 0) with equal probability according to logics that I provided above. In this way, the micromouse could minimize the chance of visiting or pass the same grid multiple times. This is also consistent to our intuition that we would like to explore unknown space step by step from small region to large region.

During the real practice, it is likely that the number of unvisited squares becomes very large. It might be slow to solve Bellman Equation for each unvisited square, depending on what kind of CPU you have and whether multi-thread calculation is employed. To accelerate the computation, I set one special option in my micrmouse settings. If the option is turned on, the micromouse will calculate the Manhattan distance between its current location and the each unvisited square in the first place, and select no more than four closest unvisited squares as selected destination candidates for Bellman Equation.

This process is repeated until the micromouse reaches the destination squares. After reaching the destination, there are three options for the micromouse:

1. The micromouse goes back to the start square and start the second run.
2. The micromouse does not go back to the start square until it has visited all the square grids in the maze.
3. If the micromouse was informed the size of maze, which is not true in my case, it keeps exploring and calculate the percentage of explored maze. It does not go back to the start square until the percentage of explored maze reaches meets certain criteria.

Because my micromouse is not going to be informed any information about the maze before the game, the third option does not apply. I will test option one and two for all the three mazes provided in this problem. The first option is designated as "incomplete" mode and the second option is designated as "complete" mode.

## Simulation Results

The micromouse was tested in both "incomplete" and "complete" modes in all three virtual maze (**Figure 2**) 20 times and the results were recorded (**Table 2**). I have also recorded one sample video for each maze and mode, saved screen shots, and submitted online (**Figure 4**) [3]. From the results, although without statistical tests, we can see that "incomplete" mode is slightly better than "complete" mode in all three mazes tested. It should also be noted that the difference between "incomplete" and "complete" modes seems to become more significant as the maze size increases.

**Table 2.** Results of micromouse simulations. All data were presented as mean ± standard deviation (n = 20).

| Maze | Mouse Mode | Time Steps of First Run | Path Length of First Run | Time Steps of Second Run | Path Length of Second Run | Observed Coverage | Score |
|---|---|---|---|---|---|---|---|
| 1 | Complete | 201.4 ± 8.3 | 248.1 ± 16.0 | 17.0 ± 0.0 | 32.0 ± 0.0 | 1.00 ± 0.00 | 24.3 ± 0.3 |
| 1 | Incomplete | 138.3 ± 49.8 | 167.6 ± 61.4 | 17.6 ± 1.0 | 32.3 ± 1.5 | 0.71 ± 0.24 | 22.8 ± 1.5 |
| 2 | Complete | 279.0 ± 10.4 | 341.0 ± 17.2 | 22.0 ± 0.0 | 43.0 ± 0.0 | 1.00 ± 0.00 | 32.0 ± 0.3 |
| 2 | Incomplete | 172.6 ± 49.4 | 207.3 ± 60.0 | 24.1 ± 2.5 | 43.7 ± 1.6 | 0.65 ± 0.19 | 30.7 ± 1.8 |
| 3 | Complete | 353.1 ± 16.1 | 442.1 ± 29.8 | 25.0 ± 0.0 | 49.0 ± 0.0 | 1.00 ± 0.00 | 37.6 ± 0.5 |
| 3 | Incomplete | 189.4 ± 75.2 | 233.0 ± 92.3 | 26.6 ± 1.3 | 50.8 ± 2 | 0.56 ± 0.23 | 33.7 ± 1.7 |

I compared my micromouse with Naoki Shibuya's micromouse [4] and found that my micromouse is much better than his. His score definition is slightly different to mine. His time steps of first run does not count the time steps of returning to the starting square grid. In addition, formally his score is equal to the number of time steps required to execute the second run plus one thirtieth the number of time steps required to execute the second runs. So, Naoki Shibuya's score definition will be lower than my score if the same path and action inputs were provided, by the definition.

Even so, my micromouse still got much lower score compared to Naoki Shibuya's micromouse in maze 2 and 3, and very close score in maze 1, suggesting that my micromouse is much better than Naoki Shibuya's micromouse.

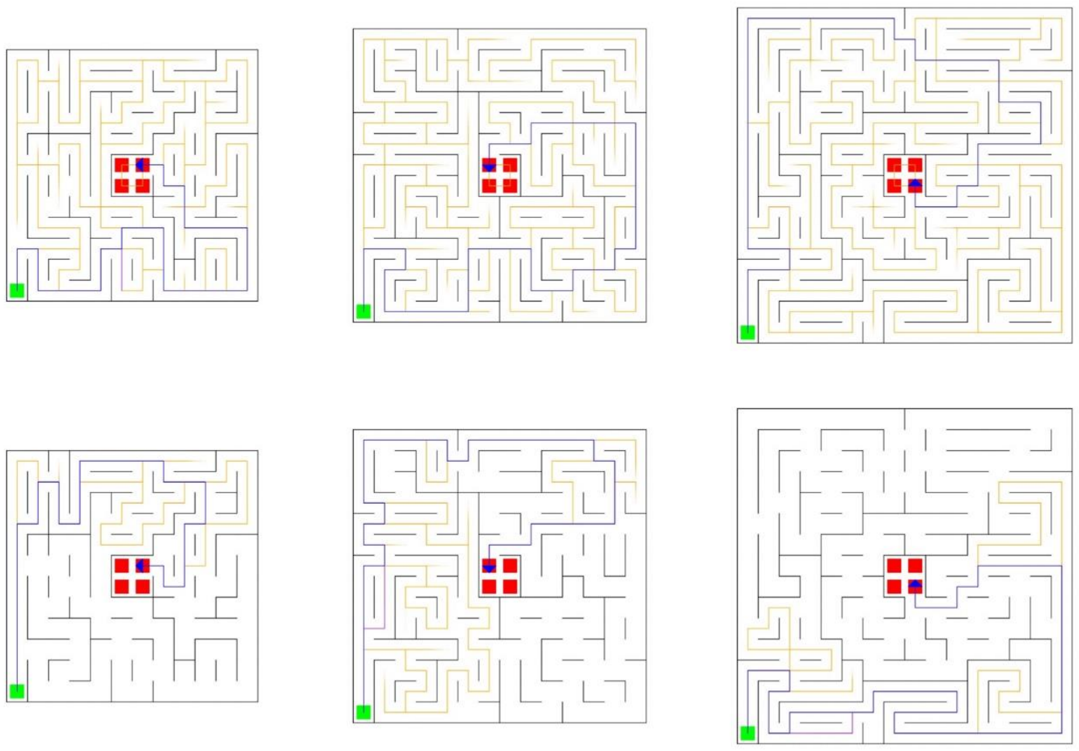


**Figure 4.** Micromouse simulation on maze 1, 2 and 3 (from left to right). The upper panel is the micromouse track in "complete" mode; the lower panel is the micromouse track in "incomplete" mode. The yellow line represents the micromouse track during exploration in the first run. The blue line represents the micromouse track in the second run. The purple line represents the mouse track of returning from destination to start square in the first run. The purple line might be overlapped with the blue line.

My micromouse has also been tested on the maze where the start square is not located at the left-bottom corner and the destination square is not located at the center of the maze (**Figure 5**). I made a maze 4 based on maze 3 simply by changing the locations of start square and destination square. The micromouse ran on it smoothly without any problem.

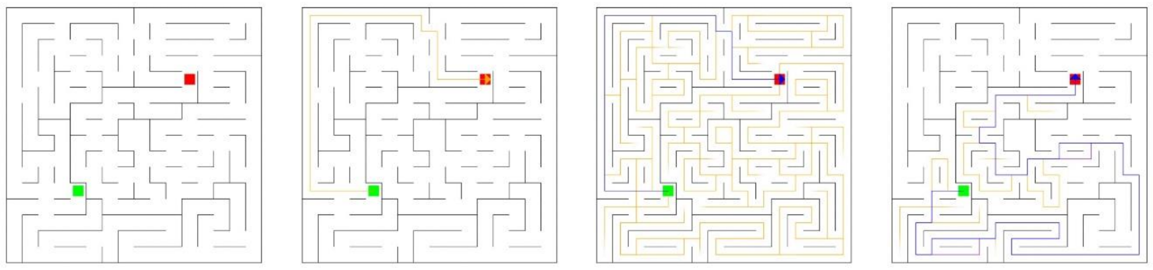


**Figure 5.** Micromouse simulation on the new maze. From left to right, the overview of maze, the optimal path from start square to destination, micromouse movement in "complete" mode, micromouse movement in "incomplete" mode.

## Discussion

To estimate the exploration efficiency, the number of squares visited per time step and per path length of first run was calculated. We can see that the number of squares visited per time step is more than 0.70, which is very high, for both mode in all maze tested.

In addition, from the recorded video, we can see that it is likely to have clustered patches of visited squares. Therefore, with sufficient regional maze information, the micromouse is able to at least calculate the optimal path and corresponding actions to the destination. This is supported by the fact that in both “complete” and “in complete” mode, the time steps of second run is always the same or very close to the theoretical minimum time steps.

**Table 3.** Micromouse exploration efficiencies. All data were presented as mean ± standard deviation (n = 20).

| Maze | Maze Size | Mouse Mode | Number of Squares Visited Per Time Step of First Run | Number of Squares Visited Per Path Length of First Run |
|---|---|---|---|---|
| 1 | 144 | Complete | 0.72 ± 0.03 | 0.58 ± 0.04 |
| 1 | 144 | Incomplete | 0.74 ± 0.03 | 0.61 ± 0.04 |
| 2 | 196 | Complete | 0.70 ± 0.03 | 0.58 ± 0.03 |
| 2 | 196 | Incomplete | 0.74 ± 0.04 | 0.61 ± 0.04 |
| 3 | 256 | Complete | 0.73 ± 0.03 | 0.58 ± 0.04 |
| 3 | 256 | Incomplete | 0.75 ± 0.04 | 0.61 ± 0.05 |

## Future Improvements

During the first run, because the mouse has to run Bellman Equation for each destination candidate to determine the destination of time step, even though I limited the number of destination candidates to at most four, the calculation process might become slow as the size of learned maze become larger in mouse’s memory. Despite this, my micromouse perform extremely well in the simulations under the scoring rule mentioned previous.

However, if the program was installed in a real mechanic micromouse, it might not be efficient because the calculation might be slow. Therefore, it will require further improvements to reduce the computing demand. There are several ways to improve computation efficiency:

1. Use better CPU and multi-thread computing.
2. Add one more destination candidate selection criteria. Sometimes, it is not necessary to visit the square to know all the wall information of that grid. If all the wall information is known before visiting it, then it does not have to be the destination candidate and the micromouse does not have to visit it. This could be achieved by adding another matrix to record whether the micrmouse was sure about the existence of certain wall in the memory.
3. The intuition of exploring an unknown maze is that if there is unvisited square accessible in the neighbors, always visit that square. The Bellman Equation solution is also consistent to this intuition. This intuition could be employed directly in micromouse.

I have implemented suggestion No. 3 in my micromouse and the simulation results were saved (**Table 4**). From the simulation results, we can see that the computation time was reduced for around one-fold while the score does not change significantly.

**Table 4.** Improvement of computation efficiency by introducing intuition. All data were presented as mean ± standard deviation (n = 20).

| Maze | Mouse Mode | Intuition | Time Steps of First Run | Path Length of First Run | Time Steps of Second Run | Path Length of Second Run | Observed Coverage | Score | Computation Time (second) | Maze Size | Number of Squares Visited Per Time Step of First Run | Number of Squares Visited Per Path Length of First Run |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Complete | FALSE | 200.8 ± 8.4 | 245.0 ± 13.1 | 17 ± 0.0 | 32.0 ± 0.0 | 1.00 ± 0.00 | 24.3 ± 0.3 | 92.1 ± 8.4 | 144 | 0.72 ± 0.03 | 0.59 ± 0.03 |
| 1 | Complete | TRUE | 200.9 ± 10.1 | 244.6 ± 17.0 | 17 ± 0.0 | 32.0 ± 0.0 | 1.00 ± 0.00 | 24.3 ± 0.3 | 44.5 ± 3.3 | 144 | 0.72 ± 0.04 | 0.59 ± 0.04 |
| 1 | Incomplete | FALSE | 117.2 ± 33.3 | 141.5 ± 40.8 | 18.1 ± 1.2 | 32.2 ± 1.3 | 0.61 ± 0.18 | 22.6 ± 1.7 | 41.8 ± 15.7 | 144 | 0.75 ± 0.03 | 0.62 ± 0.04 |
| 1 | Incomplete | TRUE | 158.8 ± 42.1 | 193.1 ± 54.2 | 17.7 ± 1.1 | 32.1 ± 1.0 | 0.80 ± 0.19 | 23.5 ± 1.2 | 32.7 ± 12.7 | 144 | 0.73 ± 0.04 | 0.61 ± 0.04 |
| 2 | Complete | FALSE | 273.3 ± 14.1 | 332.9 ± 24.3 | 22.0 ± 0.0 | 43.0 ± 0.0 | 1.00 ± 0.00 | 31.8 ± 0.5 | 171.4 ± 11.0 | 196 | 0.72 ± 0.03 | 0.59 ± 0.04 |
| 2 | Complete | TRUE | 280.4 ± 15.0 | 343.9 ± 24.5 | 22.0 ± 0.0 | 43.0 ± 0.0 | 1.00 ± 0.00 | 32.1 ± 0.5 | 82.4 ± 10.3 | 196 | 0.7 ± 0.03 | 0.57 ± 0.04 |
| 2 | Incomplete | FALSE | 195.9 ± 50.3 | 241.7 ± 61.6 | 22.6 ± 1.1 | 43.1 ± 0.5 | 0.73 ± 0.19 | 29.9 ± 1.5 | 95.9 ± 39.1 | 196 | 0.73 ± 0.03 | 0.59 ± 0.03 |
| 2 | Incomplete | TRUE | 189.0 ± 52.3 | 232.3 ± 67.9 | 23.5 ± 1.8 | 43.3 ± 0.7 | 0.70 ± 0.18 | 30.6 ± 1.8 | 48.0 ± 22.4 | 196 | 0.73 ± 0.03 | 0.59 ± 0.04 |
| 3 | Complete | FALSE | 354.0 ± 15.9 | 443 ± 28.6 | 25.0 ± 0.0 | 49.0 ± 0.0 | 1.00 ± 0.00 | 37.6 ± 0.0 | 287.0 ± 24.6 | 256 | 0.72 ± 0.03 | 0.58 ± 0.04 |
| 3 | Complete | TRUE | 359.4 ± 18.1 | 450.4 ± 37.7 | 25.0 ± 0.0 | 49.0 ± 0.0 | 1.00 ± 0.00 | 37.8 ± 0.0 | 134.4 ± 10.7 | 256 | 0.71 ± 0.04 | 0.57 ± 0.05 |
| 3 | Incomplete | FALSE | 237.5 ± 91.9 | 299 ± 121.0 | 26.0 ± 1.5 | 50.3 ± 1.3 | 0.67 ± 0.24 | 34.7 ± 0.2 | 161.3 ± 90.2 | 256 | 0.73 ± 0.04 | 0.58 ± 0.05 |
| 3 | Incomplete | TRUE | 202.8 ± 81.3 | 253.4 ± 101.0 | 26.0 ± 1.0 | 50.4 ± 1.5 | 0.59 ± 0.24 | 33.63 ± 0.2 | 62.07 ± 39.5 | 256 | 0.75 ± 0.03 | 0.6 ± 0.04 |

## Reference

[1] https://en.wikipedia.org/wiki/Micromouse

[2] http://micromouseusa.com/wp-content/uploads/2013/10/APEC-Rules.pdf

[3] https://www.youtube.com/playlist?list=PLVLJFoX8B37F6t81x2bK_Pe86TU2txIFn

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