Project Overview

A robot mouse in a virtual maze finding an optimal way to the destination by its own – that's what I programed in this project which took an inspiration from the micro mouse competition.

Analysis

The rule is simple: in the first run, the robot mouse tries to map out the maze to not only find the center, but also figure out the best paths to the center. The robot must enter to the goal within the time limit but it is free to continue exploring the maze after finding the goal. In subsequent runs, the robot mouse is brought back to the start location. It must attempt to reach the center in the fastest time possible, using what it has previously learned.

A simplified model of the world is provided along with specifications for the maze and robot. My main objective is to implement a logic that achieves the fastest times possible in a series of test mazes

Data Exploration and Visualization

The shape of every test mazes is a square. The start location is always at the left bottom corner (rows-1, 0), and the goal room always occupies a single with shaded color blue in the center.

The robot has three obstacle sensors, mounted on the front of the robot, its right side, and its left side. Obstacle sensors detect the number of open squares in the direction of the sensor. It is assumed that the robot's turning and movement is perfect.

At the start location, only up and right sides have openings so it can initiate it moves from either. The sensors will return the distance between the robot and walls in a tuple as in (left distance, forward distance, and right distance).

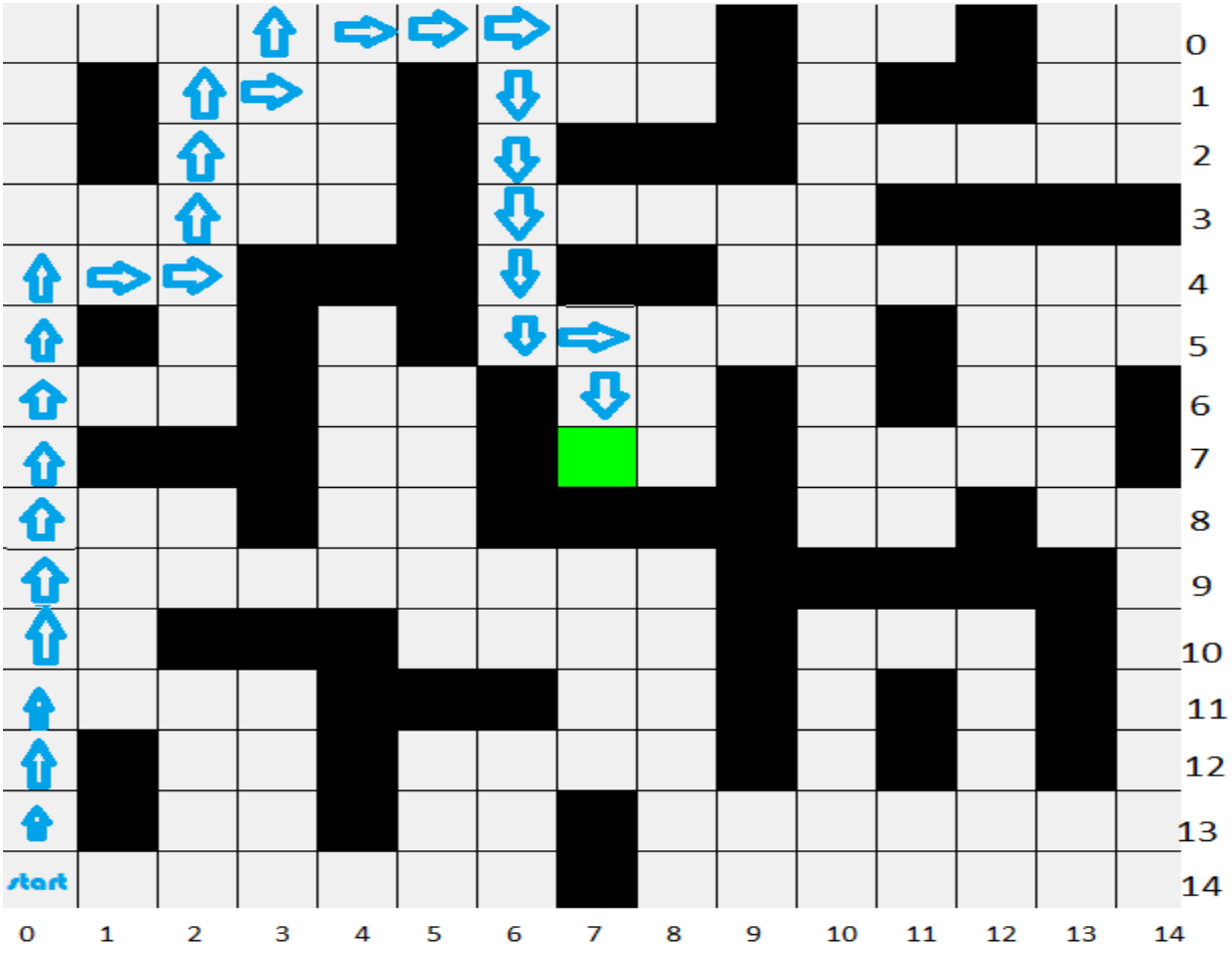
On each time step of the simulation, the robot may choose to rotate clockwise or counterclockwise ninety degrees, and then move forwards or backwards.

If the robot tries to move into a wall, the robot stays where it is. After movement, one time-step has passed, and the sensors return readings for the open squares in the robot's new location and/or orientation to start the next time unit.

Rotation is expected to be an integer taking one of three values: -90, 90, or 0, indicating a counterclockwise, clockwise, or no rotation, respectively. Movement follows rotation.

The robot will attempt to move that many squares forward (positive) or backwards (negative), stopping movement if it encounters a wall.

Maze (15by 15 grid)



The start location is (0, 14) and it is shown with the "start" sign. The goal area is located in the center and it is shown with the red balls. At the start location, the exploitation agent robot is going north (as the optimal path) and its sensors will return (0, 14, 0).

The shortest path from the start location to the goal area is indicated the blue arrows with a reward score of 1.92. It takes 28 single steps from the start location to the goal area.

Algorithms and Techniques

If the robot explores the entire maze, we can use Q-learning algorithm to find the shortest path from the start location to the goal area. Therefore, in the first run, the robot should try exploring the maze as much as possible and expand the mapping area so that, in the second and subsequent runs, the robot can apply the Q-matrix update search algorithm to find the optimal path and moves.

For the first run, I will use the following techniques for the robot controller to explore the maze and expand the mapping area while seeking the goal area:

• Random move

• Random move with dead-end path detection

• Counting number of visits for each location in order to expand into less visited area

- Updating the matrix

• The counting logic with heuristic prediction of the distance to the goal

NB: The Q-learning algorithm also handles walls properly.

Benchmark

Robot Score is simply a number that helps in deciding the winning steps of the algorithm which will consistently complete any maze. In other words, it has to have not only a good score but also be a robust technique.

I'll provide different controllers (Random, Dead-End, Counter, Heuristic) to compare the scores which should improve as more advanced techniques are introduced.

Threshold: We have chosen the threshold for an acceptable Final Normalized

Robot score to be ‘1.92.’ It has built in safety buffer of to account for any possible variation in the test environment and quirks in the algorithm implementation. Any score below this threshold means that algorithm is taking too many steps, potentially wasteful cell revisits, to complete the maze as compared to the optimal solution and hence must be rejected. The best possible Final Score is 1.000. Scores greater than 1 and near 1 are good Scores.

Methodology

Data Preprocessing

The maze specification and robot's sensor data is provided and 100% accurate. Therefore, there is no data pre-processing is required (15 by 15 grid).

Implementation:

When you extract the capstone project we have:

- one additional modules (maze_enviroment) build to design the platform of the 15 by 15 grid maze board.
- Additional “Robotic_pet” icon to beatify and represent the logo of Tk
- And the Robot.py itself which learns and controls the movement of the agent

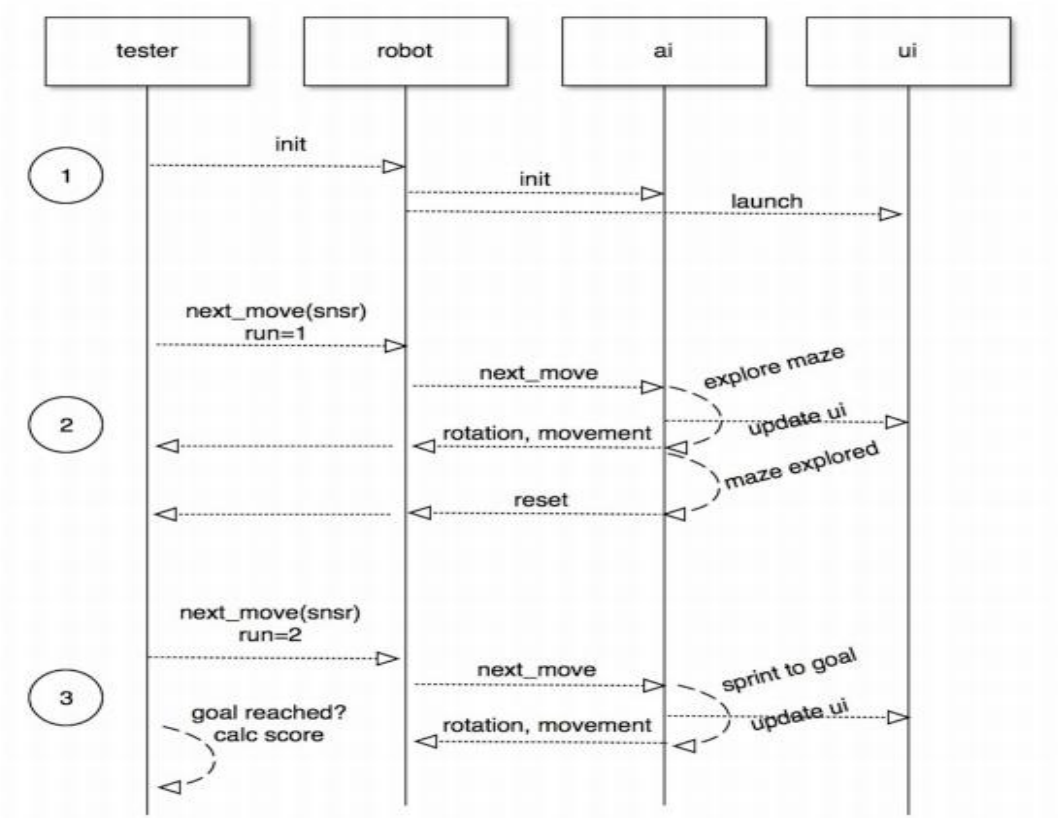
However, in the **maze_ environment** module is Utilities like

```
def call_up(event):
    try_move(0, -1)

def call_down(event):
    try_move(0, 1)

def call_left(event):
    try_move(-1, 0)

def call_right(event):
    try_move(1, 0)
```

Should consider the actions when called by the do function in **Robot.py**

```
if action == actions[0]:
    maze_enviroment.try_move(0, -1)
elif action == actions[1]:
    maze_enviroment.try_move(0, 1)
elif action == actions[2]:
    maze_enviroment.try_move(-1, 0)
elif action == actions[3]:
    maze_enviroment.try_move(1, 0)
```

Sequence Diagram



(Were **ai** is the algorithm)

Abstract AI' is the most important class of the project. It is the parent class of all AI implementations. It carries the bulk of the logic and taps into each AI implementation for AI specific interpretation at relevant points in the flow. It uses 'Maze Model' to build and store the state of the maze. It also uses 'Slam UI' for bringing to life the smart moves of AI.

All the important class attributes and methods have been noted above. For intricacies of each AI implementation, it is best to go through the source code and checkout inline comments.

The sequence of events in 'solving maze' can be broadly divided up in 3 stages:

1. Init: All components are initialized during this stage. Tester.py (main) instantiates

Robot, which in turn initializes chosen AI module and launches UI

2. Run=1: This stage is also called 'maze exploration' stage. Tester in a loop keeps asking Robot to provide its next location and orientation. Robot in turn consults the chosen

AI module to decide its next move based on the supplied sensor data.
In these sequences of movements, Robot slowly discovers the maze. Once maze has been sufficiently explored, Robot 'reset' run=1.

3. Run=n: In this stage, Robot zooms from start to finish for every next move request from the Tester. Robot does this by exploiting the information it has gathered about the maze in the previous stage. Once the goal is reached, Tester calculates and publishes Robot Score. Lower scores are better.

Refinement

Q-learning every moment is precisely guided by heuristics which steer robot straight towards the goal position without any unnecessary turns along the way. This approach is wonderful and works great if entire maze is known beforehand, where computed cells are optimal and the traced path from it is optimal as well.

The algorithm states, as the robot starts navigating towards the goal position following the initial depths and runs into any wall; we recomputed depths for the entire maze with known knowledge of walls. New depths will steer the robot around the walls and towards the goal.

This process is repeated until the robot finally reaches the goal position in the center of the maze. There is no guarantee that the path discovered in this process is optimal, as probably most of the maze is still unexplored. Hence we need an improvement over the textbook approach.

Results

Model Evaluation and Validation

Test maze	Path Length	Required Moves
0	30	17

The above numbers can be achieved if the robot has 100% mapping coverage of the maze as all robots are using the same search implementation in the subsequent runs.

The threshold for an acceptable Final Robot Score is 1.92. Any score above this threshold means that algorithm is taking too many steps, potentially wasteful cell revisits, to complete the maze as compared to the optimal solution. The best possible Final Score is 1.50. Scores greater than 1 and near 1 are good scores.

Having an acceptable Final Robot Score is not enough. A winning algorithm must consistently complete any maze as well. In other words it has to be robust.

Following is the summary of results of evaluating various AI algorithms:

Q Learning has a very powerful algorithm

The reinforcement learning problem as described requires clever exploration mechanisms. Randomly selecting actions, without reference to an estimated probability distribution, is known to give rise to very poor performance. The case of (small) finite Markov decision processes is relatively well understood by now. However, due to the lack of algorithms that would probably scale well with the number of states (or scale to problems with infinite state spaces), in practice people resort to simple exploration methods. One such method is ϵ {\display style \epsilon } \epsilon -greedy, when the agent chooses the action that it believes has the best long-term effect with probability $1-\epsilon$ {\display style 1-\epsilon } 1-\epsilon , and it chooses an action uniformly at random, otherwise. Here, $0<\epsilon <1$ {\display style 0<\epsilon <1} {\display style 0<\epsilon <1} is a tuning parameter, which is sometimes changed, either according to a fixed schedule (making the agent explore less as time goes by), or adaptively based on some heuristics.

Additionally, as you will see later in Conclusion section of this report that this implementation of algorithm went on to win over recent upgrading of **AlphaGo software**. As reference here:https://www.wired.com/2016/03/googles-ai-wins-first-game-historic-match-go-champion.

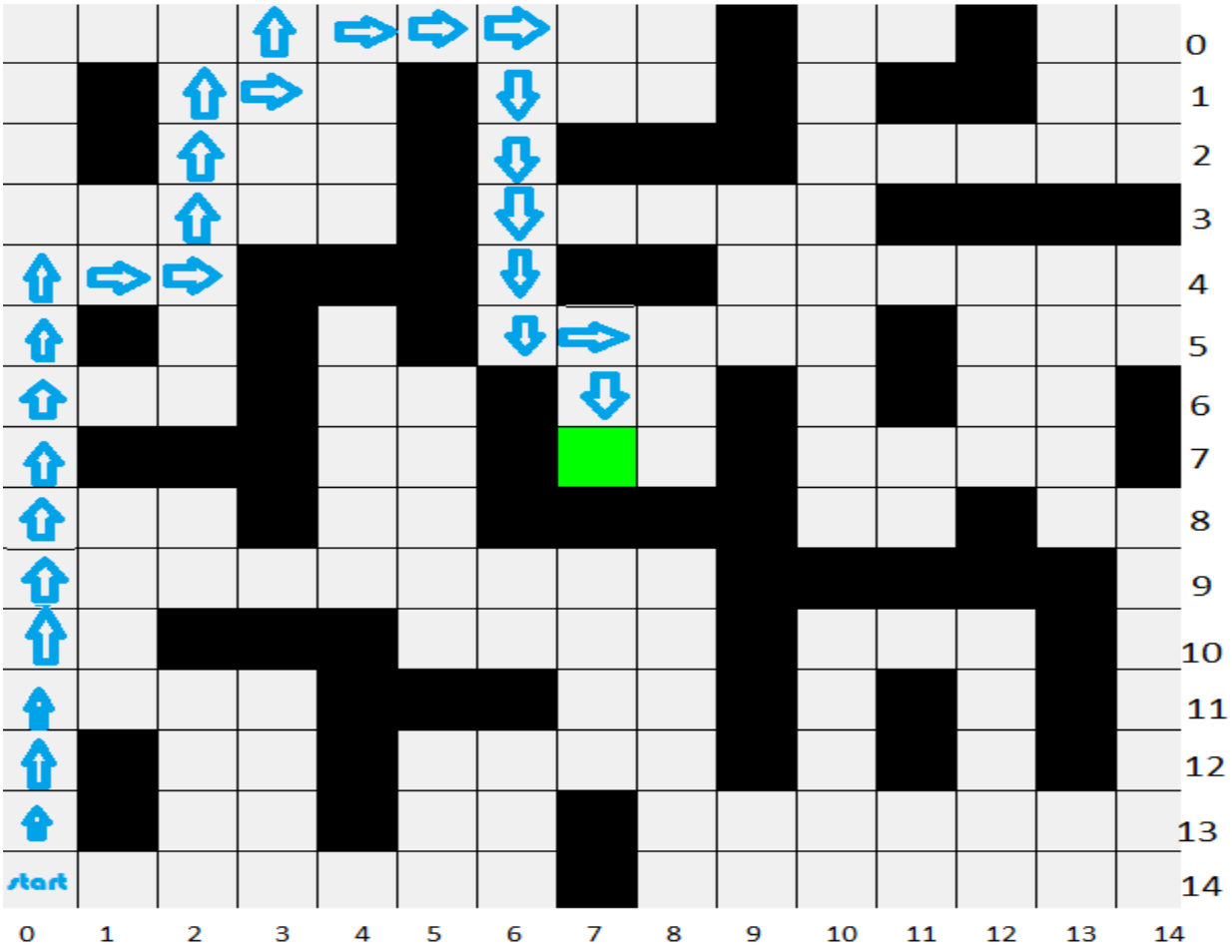
Now to justify, as noted previously in the Benchmark section of this report — Winning algorithm will consistently complete any maze. In other words, it has to have not only a good score but also be a robust technique. We have chosen the threshold for an acceptable Final Normalized Robot Score to be 1.92.

It has built in safety buffer to account for any possible variations in the test environment and quirks in the algorithm implementation. Any score below this threshold means

that algorithm is taking too many steps to complete the maze as compared to the optimal solution and hence must be rejected.

Our implementation of Q-learning algorithm has a winning scored, which is not only within the defined acceptable Robot Score threshold, but also the best possible score. Moreover, it never failed to find a solution in any execution for the maze. Hence it has met both the objective and subjective criteria to be the winning algorithm. And Q-Learning is the only algorithm that can handle the situation best when MDP can't be solved

As from the diagram below,



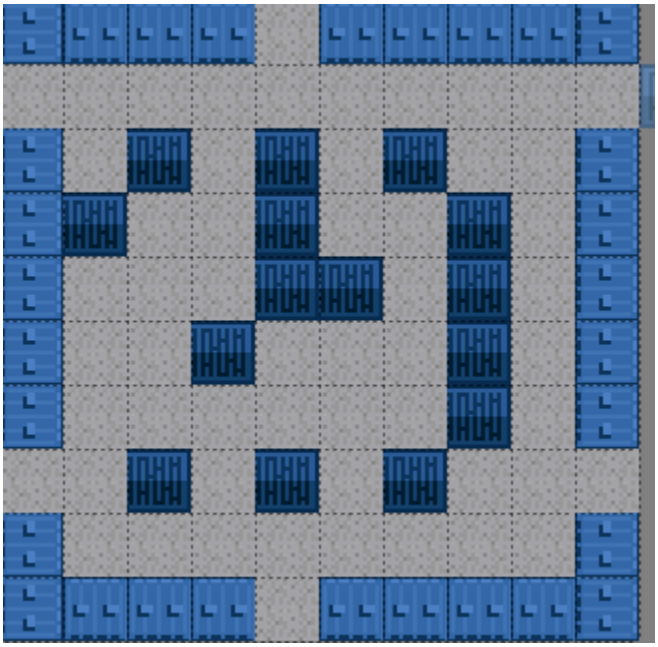
When at start, Q matrix generates a series of zeros action. But when a change of state is applied, the matrix is updated based on the moves of action taken and reward earned

Q(state, action)=R(state, action)+y*max[Q(next state, all action)]

Conclusion

Free-Form Visualization

It is easily possible to make a maze more difficult or easy for particular expansion/goal-seeking logic. I came up with a new maze (Maze special) to prove that point as shown below.



I notice with canvas:

- **Clean interface** - Canvas's straightforward, simple architecture lets instructor content be the focus.
- **Easier building experience** - Canvas is easy to navigate, so instructors can build courses easily and consistently.
- Enhance goal-oriented discussions and creativity
- Easy to use and to update
- **Very agile:** Product vision can be constantly adapte

Reflection

Oh boy! From watching many inspired programming videos and movies. To engaging in my first robotic application has been a blast! :)

It has been an incredible journey. Like many kids, Robots always captivated my imagination.

I see myself doing what I'm doing right now (writing a report on robots) and starting self-driving Nano degree. Thank you Profs! ###. Thank you Udacity!. You have started a chain reaction that will revolutionize human life in a short period.

The initial challenge for me was to divide the project into smaller problems to tackle with. Applying the actual formulas I have learned in the course to the practical application through coding.

I faced numbers of challenges, but what kept me going forward was the idea of visualizing a moving robot studying to accomplish a task based on the rewards you give to him.

Realizing this project was a great deal! For real!

Improvement

Without doubt the current scope of the project is awesome. It fulfils its objective of introducing Robot Motion Planning and then some. That been said, here are few areas of improvement that can bring our robot much closer to real life implementation:

- Robot Motion Noise: In real life, robot motion is hardly 100%. There is always some

Probability that robot ends up in a location different from the intended destination. Although the robot will make it to the final goal, it may take a path different from optimal path.

● Sensor Noise: Real sensors are also rarely 100% accurate. Their output can be drawn from a probability distribution. For practical motion planning we should account for it as well.

● Generating Smooth Path: Real robots cannot take 90 degree turns. Slow, smooth and curvy paths, especially around the corners are much preferred.

- Keeping safe distance: It is not enough for us to plan shortest path to the goal. Doing so may bring robot dangerously close to the obstacles or walls. We should always account for some safety margin and interpolate robot path coordinates to keep it at a
- Diagonal Movement: If a maze in continuous domain is such that it requires robot to go diagonally across the cells, then discrete robot might not follow that path. It will look for bit longer path, but with 90 degree turns. If a such a 90 degree path does not exists, then discrete robot will not be able to solve the continuous maze

<u>EXTRAS:</u>

Neural Nets: I'm sure a real life robot will not work without a NN component. We should look for a place in the project to include the use of NN

Computer Vision: Similarly, we totally missed out on Computer Vision aspect in robot motion planning. It would be really nice if we could incorporate it along with NN in our toy project