Machine Learning Engineer Nanodegree

Plot and Navigate a Virtual Maze

Rushi Patel

Definition

Project Overview

This project is about a virtual robot being able to plot and navigate from the virtual maze's left bottom corner to the virtual maze's center in the fastest time possible. The robot will go through two runs of the maze: one will be an exploration phase where it may be able to explore and plot the maze to find the best path to the center, and the second run will have the robot navigate the fastest path to the center of the maze. The robot will have to plot and navigate through a series of test mazes and obtain the fastest times from each test maze.

The inspiration for this project was taken from the Micromouse competitions where a robot mouse plots the maze from a corner of the maze to it's center. The robot mouse is allowed multiple runs in a maze to achieve the fastest time possible. The mouse's first run is used for plotting the maze then the runs afterwards are used to achieve the fastest time possible based off of what it had learned in previous runs.

Problem Statement

The main problem has been defined in the brief overview statement earlier. The virtual robot needs to plot and navigate a series of test mazes and be able obtain the fastest time possible for each maze. It will need to be able to plot and navigate from the bottom left corner to the center of the maze. The robot will have to be able to sense the walls and be able to move around them rather than through them. There are also possibilities of the robot going in a loop around a certain area in the maze of running into a dead end. It will need to learn how to avoid these obstacles and be able to take the fastest path to the center of the maze. The approach/ implementation to this project will have to keep all of these possibilities in mind in order for the robot to properly get to the center in the fastest time.

The motivation that will be used for this project is plain and simple. Expand upon the starter code that is provided and make sure the code is optimized properly so it may run efficiently. The code provided already has the simple model of the world along with the specifications for the maze and the robot. Q-Learning, a Reinforcement Learning technique, is the approach that will be implemented for the robot to get through the

maze. There will be three approaches that will be taken for this project. I will set the benchmark for the run time through the maze with random movement (Q-Learning with no learning rate). I will compare results of Q-Learning with a learning rate of one (implying the future states for Q-Learning involve no randomness) and Q-Learning with a decaying learning rate and an incrementally increasing discount factor to the benchmark time. Q-Learning with a learning rate of one will consider every recent action the robot has taken. For a maze with a step limit of a thousand max steps, the robot should be able to quickly consider its recent actions to be able to get to the goal quickly. The other method of Q-Learning with decreasing learning rate and an increasing discount factor, from the results of this approach I would like to see the robot be able to move accordingly to the states and actions in the Q-table as the learning rate decreases, but also make movements according to future high rewards due to the increasing discount factor.

Metrics

The scoring metric for this project will be based off of how well the implementation of the code functions. The benchmark time will show how well the actual implementation functions compared to it. I will test the robot a hundred times per Q-Learning variation and compare whether they passed run 1, run 2, success rate of the robot passing both runs within the given number of trials, and the fastest time to the center, if applicable.

Analysis

Data Exploration

The robot will go through the maze by using the sensors the placed on it. The sensors are placed on the left, front, and right of the robot. It will sense the distance to the wall in those directions. According to the the action the robot takes, it will only be able to rotate negative ninety degrees(counterclockwise), stay zero degrees for a forward movement, and rotate positive ninety degrees(clockwise). Within a time step, the robot is limited to moving three squares either forward or backwards.

There are three test mazes (12x12, 14x14, 16x16) given with the starter code. Each maze has been given an even and equal dimension. Each maze is a text file filled the maze dimension in the first line and then the following line numbers correlate to the open edges for valid actions likely to be able to be taken from that square/ location. For instance, if in the following figure the square has a 1 then the robot's only valid action would be for it to go up because it is the only open edge available for the robot to move

through. Each edge has a four bit number attached to it: 1's up, 2's right, 4's down, 8's left.

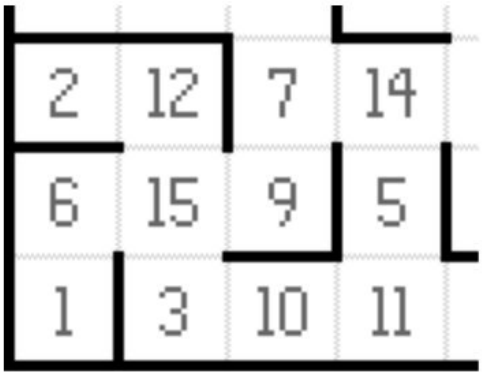

Figure 1: Maze specifications

There are areas in the maze that the robot will have to try to prevent. In the maze, there are possibilities for the robot to go in a loop around an area or run into dead ends like I mentioned earlier in the problem statement. Going in certain areas over and over can not only waste time step, but it also may result in failure. Dead ends are the same since a robot can get stuck in a location and end up having a tendency for trying to waste time steps running into them and trying to get out of them in search of the goal. In the following figure, the majority of dead ends are marked by blue X's and majority of areas for the robot to loop in are circled in red.

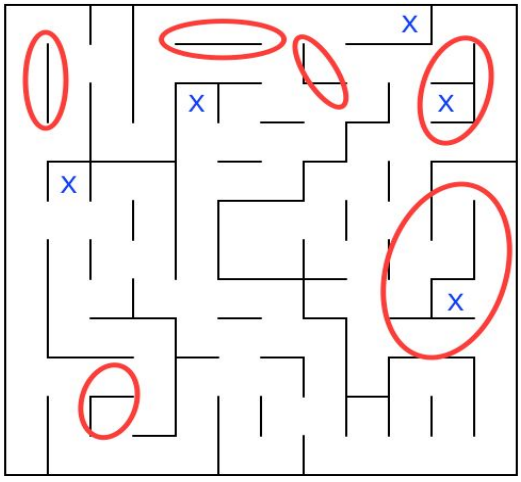

Figure 2: 12x12 Maze (Loops and Dead Ends)

The goal is at the center of every maze that cover four squares. Any goal square in the goal squares can be set as the location for the robot to reach. For this project, the goal square is going to set at the half of the maze dimension minus one by half of the

maze dimension, goal = [(maze dimension/2) - 1, maze dimension/2]. In the below figure, you can see how the goal is at the center of the 14x14 maze.

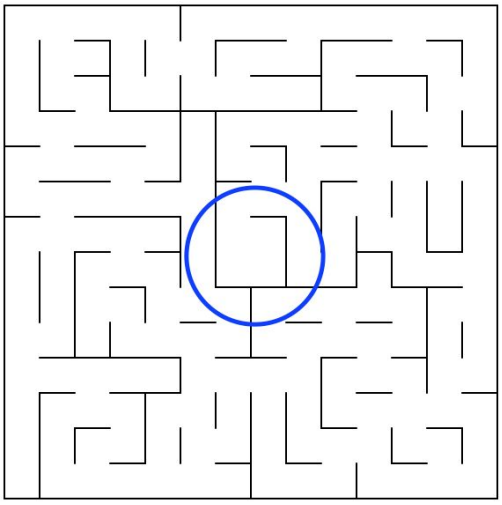

Figure 3: 14x14 Maze(Goal)

Exploratory Visualization

There have been visualizations included in the previous section to demonstrate how the goal is at the center of the mazes, and where there are possible or majority of the dead ends and areas where the robot can be moving in loops are located. In the figure below, the optimal path is shown that the robot takes to reach the goal. This optimal path takes fifty-one steps to reach from the initial position to the goal.

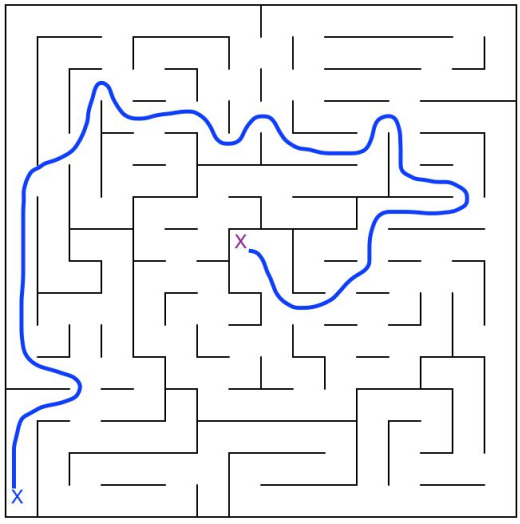

Figure 4: 16x16 Maze(Optimal Path)

Algorithms and Techniques

For this project, there will only be an implementation of one techniques for the robot. That technique will be Q-Learning which is a model-free Reinforcement Learning technique. Q-Learning is based off of the robot's state, action, and reward. Each state and action are appointed a Q value in the Q-table which is comprised of the old Q value along with the learning rate (alpha), reward, and the discount factor (gamma). Following equation is the Q-Learning equation:

$$Q(s,\ a)\ <-\ Q(s,\ a)\ +\ \alpha\ \cdot\ (r\ +\gamma\ \cdot\ \underset{a'}{^{argmax}}\ Q(s',\ s')\ -\ Q(s,\ a))$$

The equation demonstrates how every time the robot selects an action it looks at the reward and the next state depends upon the previous state and action then updates the Q value.

For the random movement, the learning rate will be set to zero so the robot will not be learning, or updating the Q-table. The learning rate will be set to one and the discount factor will be kept at zero, so the robot doesn't consider the future high rewards. The last one will have a decreasing learning rate and an increasing discount factor to see if the robot will be able to run independently based on what it had learned while looking forward to future states and actions that will gain it a higher reward.

There will be an implementation of one algorithm to account for the dead ends and commons areas where looping may occur. The algorithm will focus on checking the Manhattan distance from the current location to the goal and check whether it is greater, equal to, or less than the previous Manhattan distance along with whether or not the current location was previously visited in set list of visited locations. There will be different rewards awarded upon how well the robot is moving along the maze based upon its location. The robot should be able to learn from going into dead ends or previously visited locations with this algorithm.

Benchmark

The benchmark will be based off of how the random movement performs during the hundred tests. The tests will be grouped into sets of ten so it will be easier to compare the data. Each set will show how many tests within the set passed the first run, passed the second run, and the time it took to reach the goal, if applicable. This benchmark data will be compared with the data gathered from the other variations of Q-Learning. An example of how the benchmark will work is provided below.

Q-Learning (Random Movement)			
	Run 1	Run 2	Steps (Fastest Time)
Set 1	1/10 (10%)	0/1 (0%)	N/A
Set 2	0/10 (0%)	N/A	N/A
Set 3	0/10 (0%)	N/A	N/A
Set 4	0/10 (0%)	N/A	N/A
Set 5	3/10 (30%)	0/3 (0%)	N/A
Set 6	1/10 (30%)	0/1 (0%)	N/A
Set 7	5/10 (50%)	1/5 (20%)	150 steps
Set 8	2/10 (20%)	0/2 (0%)	N/A
Set 9	0/10 (0%)	N/A	N/A
Set 10	4/10 (40%)	2/4 (50%)	145 steps
Total	16/100 (16%)	3/16 (18.75%)	145 steps

Figure 5: Example benchmark

Methodology

Data Processing

There is no data preprocessing needed for this project because the sensor data and the maze environment were provided and perfect.

Implementation

Q-Learning

To implement Q-Learning, I created a separate file, q_learning.py, with the class QLearning. It has the functions to build states, create Q-tables, get max Q values, choose an action, and learn from the states, actions, and rewards. It is then called in robot.py for implementation in the ‘next_move’ function.

```
class QLearning(object):
    def __init__(self):
        self.Q = dict()           # Q-table
        self.epsilon = 0.85    # Exploration factor
        self.alpha = 1          # Learning rate
        self.actions = ['up', 'right', 'down', 'left']  # Available actions
        self.discount = 0      # Discount Factor
        self.t = 0                # Time

    def build_state(self, location, sense):
        state = (location, sense)
        return state

    def get_maxQ(self, state):
        # Get the max Q value
        maxQ = max(list(self.Q[state].values()))
        return maxQ

    def create_Q(self, state):
        if state not in self.Q:
                # Create new state key, value new dictionary
                self.Q[state] = {}
                for action in self.actions:
                    # Initialize the actions value to 0.0
                    self.Q[state][action] = 0.0
        return

    def choose_action(self, state):
        self.state = state
        action = None

        # 0 < a < 1
        a = 0.85
        self.epsilon = pow(a, self.t)
        self.t += 1

        # Probability 1 - epsilon
        if self.epsilon > random.random():
```

```
            action = random.choice(self.actions)
        else:
            # Get the max value of Q for the current state
            max_Q_value = self.get_maxQ(state)
            # Get the list of actions correlated to max Q
            max_Q_actions = [Q_value for Q_value in self.Q[state].keys() if
                             self.Q[state][Q_value] == max_Q_value]
            # Make the action a random choice of actions for the max Q
            action = random.choice(max_Q_actions)
        return action

    def learn(self, state, action, reward):
        max_Q = self.get_maxQ(state)
        old_Qsa = self.Q[state][action]
        self.Q[state][action] = (old_Qsa * (1 - self.alpha) + self.alpha *
                                 (reward + self.discount * max_Q))

        # 0 < alpha_t < 1
        # alpha_t = 0.99
        # self.alpha = pow(alpha_t, self.t)
        # self.t += 1

        # 0 < gamma_t <= 1
        # gamma_t = 0.05
        # self.discount += gamma_t
        return

    '''def update(self, sense):
        state = self.build_state(sense)        # Get current state
        self.create_Q(state)                   # Create 'state' in Q-table
        action = self.choose_action(state)     # Choose an action
        self.learn(state, action, reward)      # Q-learn
        return'''
```

Maze

In maze.py, I have implemented a function ‘heuristic’ which returns the Manhattan distance from the current location to the goal. Another function implemented in maze.py is ‘move’, it distributes the rewards for the actions that the robot takes. The

algorithm to check whether or not the robot has previously been to that location (i.e. moving in loops, dead ends) is also implemented under this function.

```
def heuristic(self, goal, location):
        '''
        Using the Manhattan distance to determine how far the
        robot is relevant to it's location.
        '''
        # Current x-value from goal
        self.dx = abs(location[0] - goal[0])
        # Current y-value from goal
        self.dy = abs(location[1] - goal[1])
        # Total distance of x and y from goal
        return self.dx + self.dy

def move(self, goal, location, action):
        '''
        Returns the reward per action taken by the robot. Each action has the same
        reward due having freedom of actions to move freely about the maze. The robot
        will also be rewarded upon how well it changes it location. If it goes in a loop
        or a previous  location is visited then it would receive a negative reward.
        '''
        # Initial reward value
        self.reward = 0.0

        # Manhattan distance from location to goal
        distance = self.heuristic(goal, location)
        # Previous distance; initialized at 0
        previous = 0
        # History of previous locations visited in a list
        history = []

        if location[0] in goal and location[1] in goal:
            self.reward += 10
        else:
            if distance < previous and location not in history:
                self.reward += 0.25
                previous = distance
                history.append(location)
            elif distance == previous or location in history or distance == previous and
location \
                in history:
                    self.reward += -0.25
                    previous = distance
```

```
                history.append(location)
            elif distance > previous or distance in history or distance > previous and
distance in \
                history:
                    self.reward += -0.25
                    previous = distance
                    history.append(location)
            else:
                self.reward += -0.25
                previous = distance
                history.append(location)

        # Reward for actions taken
        if action == 'up':
            self.reward += 0.25
        elif action == 'right':
            self.reward += 0.25
        elif action == 'down':
            self.reward += 0.25
        elif action == 'left':
            self.reward += 0.25
        else:
            self.reward += -0.25

        return self.reward
```

Refinement

The final refinement will come after the results are taken and compared with the benchmark. The initial solution is the random movement variation of Q-Learning which is set as the benchmark, and the intermediate solutions are the Q-Learning with the learning rate of one and discount factor of zero and the Q-Learning with the decreasing learning rate and increasing discount factor. I am using two variations of Q-Learning to compare with the benchmark, so the variation that performs better will be the final solution and will be implemented into the robot. In the justification section, I will explain which variation was better and why I'll implement it.

Results

Model Evaluation and Validation

Following are the evaluations of the benchmark and the solutions:

Test Maze 1

Q-Learning (Random Movement)			
	Run 1	Run 2	Steps (Fastest Time)
Set 1	5/10 (50%)	0/5 (0%)	N/A
Set 2	2/10 (20%)	0/2 (0%)	N/A
Set 3	2/10 (20%)	0/2 (0%)	N/A
Set 4	1/10 (10%)	0/1 (0%)	N/A
Set 5	1/10 (10%)	0/1 (0%)	N/A
Set 6	4/10 (40%)	1/4 (25%)	258
Set 7	2/10 (20%)	0/2 (0%)	N/A
Set 8	2/10 (20%)	0/2 (0%)	N/A
Set 9	0/10 (0%)	N/A	N/A
Set 10	0/10 (0%)	N/A	N/A
Total	19/100 (19%)	1/19 (5.26%)	258

Q-Learning(α = 1, γ = 0)			
	Run 1	Run 2	Steps (Fastest Time)
Set 1	2/10 (20%)	0/2 (0%)	N/A
Set 2	4/10 (40%)	0/4 (0%)	N/A
Set 3	3/10 (30%)	0/3 (0%)	N/A

Set 4	0/10 (0%)	N/A	N/A
Set 5	3/10 (30%)	0/3 (0%)	N/A
Set 6	2/10 (20%)	0/2 (0%)	N/A
Set 7	4/10 (40%)	0/4 (0%)	N/A
Set 8	1/10 (0%)	0/1 (0%)	N/A
Set 9	1/10 (0%)	0/1 (0%)	N/A
Set 10	2/10 (20%)	0/2 (0%)	N/A
Total	20/100 (20%)	0/20 (0%)	N/A

Q-Learning(decreasing α, increasing γ)			
	Run 1	Run 2	Steps (Fastest Time)
Set 1	3/10 (30%)	0/3 (0%)	N/A
Set 2	3/10 (30%)	0/3 (0%)	N/A
Set 3	5/10 (50%)	0/5 (0%)	N/A
Set 4	3/10 (30%)	0/3 (0%)	N/A
Set 5	1/10 (10%)	0/1 (0%)	N/A
Set 6	3/10 (30%)	0/3 (0%)	N/A
Set 7	2/10 (20%)	0/2 (0%)	N/A
Set 8	2/10 (20%)	0/2 (0%)	N/A
Set 9	2/10 (20%)	0/2 (0%)	N/A
Set 10	3/10 (30%)	0/3 (0%)	N/A
Total	27/100 (27%)	0/27 (0%)	N/A

Test Maze 2

Q-Learning (Random Movement)			
	Run 1	Run 2	Steps (Fastest Time)
Set 1	5/10 (50%)	1/5 (20%)	172
Set 2	0/10 (0%)	N/A	N/A
Set 3	1/10 (10%)	0/1 (0%)	N/A
Set 4	2/10 (20%)	0/2 (0%)	N/A
Set 5	0/10 (0%)	N/A	N/A
Set 6	0/10 (0%)	N/A	N/A
Set 7	0/10 (0%)	N/A	N/A
Set 8	0/10 (0%)	N/A	N/A
Set 9	1/10 (0%)	0/1 (0%)	N/A
Set 10	2/10 (20%)	0/2 (0%)	N/A
Total	11/100 (11%)	1/11 (9.09%)	172

Q-Learning(α = 1, γ = 0)			
	Run 1	Run 2	Steps (Fastest Time)
Set 1	0/10 (0%)	N/A	N/A
Set 2	0/10 (0%)	N/A	N/A
Set 3	2/10 (20%)	0/2 (0%)	N/A
Set 4	1/10 (10%)	1/1(100%)	283
Set 5	3/10 (30%)	0/3 (0%)	N/A
Set 6	1/10 (10%)	0/1 (0%)	N/A
Set 7	2/10 (20%)	0/2 (0%)	N/A

Set 8	1/10 (10%)	0/1 (0%)	N/A
Set 9	2/10 (20%)	1/2 (50%)	215
Set 10	1/10 (10%)	0/1 (0%)	N/A
Total	12/100 (12%)	2/12 (16.67%)	215

Q-Learning(decreasing α, increasing γ)			
	Run 1	Run 2	Steps (Fastest Time)
Set 1	1/10 (10%)	0/1 (0%)	N/A
Set 2	1/10 (10%)	0/1 (0%)	N/A
Set 3	2/10 (20%)	0/2 (0%)	N/A
Set 4	0/10 (0%)	N/A	N/A
Set 5	0/10 (0%)	N/A	N/A
Set 6	1/10 (10%)	0/1 (0%)	N/A
Set 7	1/10 (10%)	0/1 (0%)	N/A
Set 8	0/10 (0%)	N/A	N/A
Set 9	0/10 (0%)	N/A	N/A
Set 10	2/10 (20%)	0/2 (0%)	N/A
Total	8/100 (8%)	0/8 (0%)	N/A

Test Maze 3

Q-Learning (Random Movement)			
	Run 1	Run 2	Steps (Fastest Time)
Set 1	1/10 (10%)	0/1 (0%)	N/A

Set 2	0/10 (0%)	N/A	N/A
Set 3	2/10 (20%)	0/2 (0%)	N/A
Set 4	2/10 (20%)	0/2 (0%)	N/A
Set 5	2/10 (20%)	0/2 (0%)	N/A
Set 6	2/10 (20%)	0/2 (0%)	N/A
Set 7	0/10 (0%)	N/A	N/A
Set 8	1/10 (10%)	0/1 (0%)	N/A
Set 9	0/10 (0%)	N/A	N/A
Set 10	0/10 (0%)	N/A	N/A
Total	9/100 (9%)	0/9 (0%)	N/A

Q-Learning(α = 1, γ = 0)			
	Run 1	Run 2	Steps (Fastest Time)
Set 1	0/10 (0%)	N/A	N/A
Set 2	3/10 (30%)	0/3 (0%)	N/A
Set 3	0/10 (0%)	N/A	N/A
Set 4	2/10 (20%)	0/2 (0%)	N/A
Set 5	0/10 (0%)	N/A	N/A
Set 6	0/10 (0%)	N/A	N/A
Set 7	0/10 (0%)	N/A	N/A
Set 8	1/10 (10%)	0/1 (0%)	N/A
Set 9	2/10 (20%)	0/2 (0%)	N/A
Set 10	2/10 (20%)	0/2 (0%)	N/A
Total	10/100 (10%)	0/10 (0%)	N/A

Q-Learning(decreasing α, increasing γ)			
	Run 1	Run 2	Steps (Fastest Time)
Set 1	1/10 (10%)	0/1 (0%)	N/A
Set 2	1/10 (10%)	0/1 (0%)	N/A
Set 3	2/10 (20%)	0/2 (0%)	N/A
Set 4	0/10 (0%)	N/A	N/A
Set 5	0/10 (0%)	N/A	N/A
Set 6	2/10 (20%)	0/2 (0%)	N/A
Set 7	1/10 (10%)	0/1 (0%)	N/A
Set 8	1/10 (10%)	0/1 (0%)	N/A
Set 9	0/10 (0%)	N/A	N/A
Set 10	1/10 (0%	0/1 (0%)	N/A
Total	10/100 (10%)	0/10 (0%)	N/A

The final model will be the Q-Learning with learning rate as one. It has shown to be a little more consistent than the other variation. It was also able to reach the goal a couple of times. The parameters of Q-Learning having the learning factor equal one and discount factor be zero are appropriate. The robot will be able to learn from the most recent information in order to be able to quickly find the optimal path to the goal. Another reason to have the learning rate at one is because it is optimal for deterministic environments such as the maze. There will not be any random changes for the robot to account for in a maze with a set starting and ending location. The discount factor being zero will mean that the robot with only consider current rewards rather than looking ahead for future high rewards. The only problem with this variation is that if there is any changes to the maze then the robot will not be able to properly navigate the maze. It will tend to stick to what it had learned from the previous variation of the maze.

Justification

The Q-Learning with a learning rate of one and a discount factor of zero had better results than the benchmark. It continuously did better in each maze than the benchmark and even reach the goal a few times. The robot was able to learn from the states, actions, and rewards even if was a little bit. This technique can be adequate enough to solve the problem if it ran through more runs of the mazes. Unfortunately, Q-Learning needs an exponentially higher amount of runs to learn the correct path due to the number of states it has to go through. It will be able to navigate the the maze properly and reach the goal using the optimal path.

Conclusion

Free Form Visualization

The robot has not dealt with a lot of dead ends so I created another 12x12 test maze with much more dead ends. The test maze foundation is the first test maze except with many changes including the entry to the center of the maze. There are more dead ends created because I want to see if the the robot gets thrown off or functions below its average when set up to find the optimal path in an advanced setting. The robot needs to be able to find it's way to through the maze under more advanced conditions. The given test mazes were much more simple just to see whether the robot can plot and navigate by finding and using the optimal path. Once that is done the robot should be able to be put into another setting where there are more roadblocks and see how well it functions. The figure below is the fourth test maze (12x12) that I created.

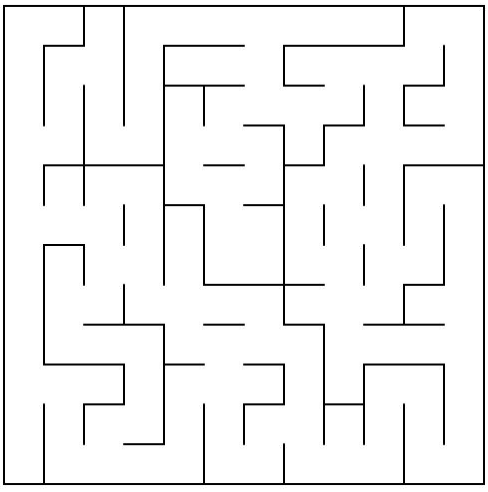

Figure 6: Test Maze 4 (12x12)

Q-Learning(α = 1, γ = 0)			
	Run 1	Run 2	Steps (Fastest Time)
Set 1	2/10 (20%)	0/2 (0%)	N/A
Set 2	1/10 (10%)	0/1 (0%)	N/A
Set 3	1/10 (10%)	0/1 (0%)	N/A
Set 4	0/10 (0%)	N/A	N/A
Set 5	2/10 (20%)	0/2 (0%)	N/A
Set 6	3/10 (30%)	0/3 (0%)	N/A
Set 7	1/10 (10%)	0/1 (0%)	N/A
Set 8	2/10 (20%)	0/2 (0%)	N/A
Set 9	2/10 (20%)	0/2 (0%)	N/A
Set 10	1/10 (10%)	0/1 (0%)	N/A
Total	15/100 (15%)	0/15 (0%)	N/A

The final solution Q-Learning technique doesn't do too bad with the maze in its first hundred runs. It is rather consistent with its previous runs for the other test mazes. The robot should be able to find it's way to the center of the maze after some more trials.

Reflection

This project was an opportunity to apply Reinforcement Learning technique, Q-Learning, to a robot and have it plot and navigate a maze. The robot was able to navigate through the maze through learning from the rewards received for an action per state. It was able to take itself from the starting position in the bottom left corner to the goal in the center of the maze. The final solution decided based upon the results fit my expectations this problem. It can be used in other deterministic environments such as the maze and be successful.

One of the difficult parts about this problem was what technique to apply to it. There are many options such as Q-Learning, A* search, and SLAM (Simultaneous Localization and Mapping). All of these techniques, if implemented correctly, would help the robot plot and navigate the maze on it's own. It is interesting to see how there can be multiple techniques to solve this type of problem. My decision was to go with Q-Learning since it was more part of machine learning than A* search and SLAM which are more AI based for robotics.

Overall it was a great experience implementing Q-Learning to the robot. In the future, I would like to work with implementing A* search or SLAM to the robot and see how well they compared with the Q-Learning technique.

Improvement

There is a lot of improvements that can be made for this project. The implementation of A* search as the benchmark and using Reinforcement Learning are some of the approaches that can be applied to this project which is in the discrete domain. The sensors, turning, and movement and this project are considered to be perfect; no noise to compensate for. In the real-world, there would need to be compensation for the noise by the sensors, turning, and movement. PID controllers can be applied in order to keep the robot moving in the middle of the walls. To make the search for the goal easier, the application of a particle filter or a Kalman filter with SLAM (Simultaneously Localization And Mapping) so you can map an unknown location while localizing your position. There should be consideration for the robot to be able to move diagonally rather than forward/ backwards and left/ right. The virtual robot is given with sensors on the left, front, and right, but if there were options given such as LIDAR can be applied if the robot was real and moving in a real maze. There a lot of approaches for improvement to the robot whether it is virtual or real.

References

Artificial Intelligence for Robotics

"Q-learning." *Wikipedia*. Wikimedia Foundation, 23 Mar. 2017. Web. 02 May 2017. <https://en.wikipedia.org/wiki/Q-learning>.

Reinforcement Learning (Machine Learning Engineer Nanodegree)

Train Smartcab How to Drive Project