Machine Learning Engineer Nanodegree

Capstone Proposal

Rohan Gupta

April 18, 2018

Report

THE MAZE RUNNER

I. Definition

Project Overview

Domain Background of this project if a agent is in a maze and agent want to find a way to the finishing point , then how can agent solve or find a way to a destination in a minimum number of steps. To find a optimal path (minimum number of steps or minimum time taken) from one position to another position in a maze. Agent can run many number of times to explore all maze and find the best way while learning. Agent can learn form its previous runs or steps to improve.

This project is similar to Micromouse competitions where a agent move in a maze and try to find the center of the maze.
A Micromouse competition is an event where a small robot attempts to navigate an unfamiliar maze. The robot starts in a corner and must find its way to a designated goal area commonly located near the center

This project is also makes a robot autonomous and robot learn form their previous mistakes so that it always perform better than the previous robots.

Problem Statement

Problem statement is to find a optimal way in a maze by agent from starting point to destination point in minimum time taken and minimum steps taken. In a maze there can any number of walls (so that agent can't penetrate through it) and any number of dead zones (after go to the dead zone agent start form its starting point).

In first run agent choose random steps (forward, left, right, backward) and try to find out the way to destination.

In second run agent choose what will be the next by step form Q learning algorithm and agent updates itself after every steps. After exploring all the maze (nearly) it knows the best way to reach destination.

In third run agent follows the steps (which learned earlier in second run) and go to destination in minimum number of steps.

Metrics

Our task is to reach the destination in minimum time , so by taking minimum number of steps agent can reach to destination.

The evaluation metric is the number of steps taken by agent to reach its destination. It is an integer type from zero(if agent is already on finishing point) to infinite(if agent never reaches its destination). Our goal is to minimize the steps take by agents.

II. Analysis

Data Exploration

In this project, a maze of constant or fixed size of rows and columns. A set of walls may be present between rows and columns. The walls are generated at maze generation time, and are static throughout the learning and testing processes. The maze also contains a goal point or a finishing point. The agent we are training will be at given starting point, and must determine the shortest path from that starting point to the destination point. It is guaranteed that such a path exists. The finishing point is mark as green.

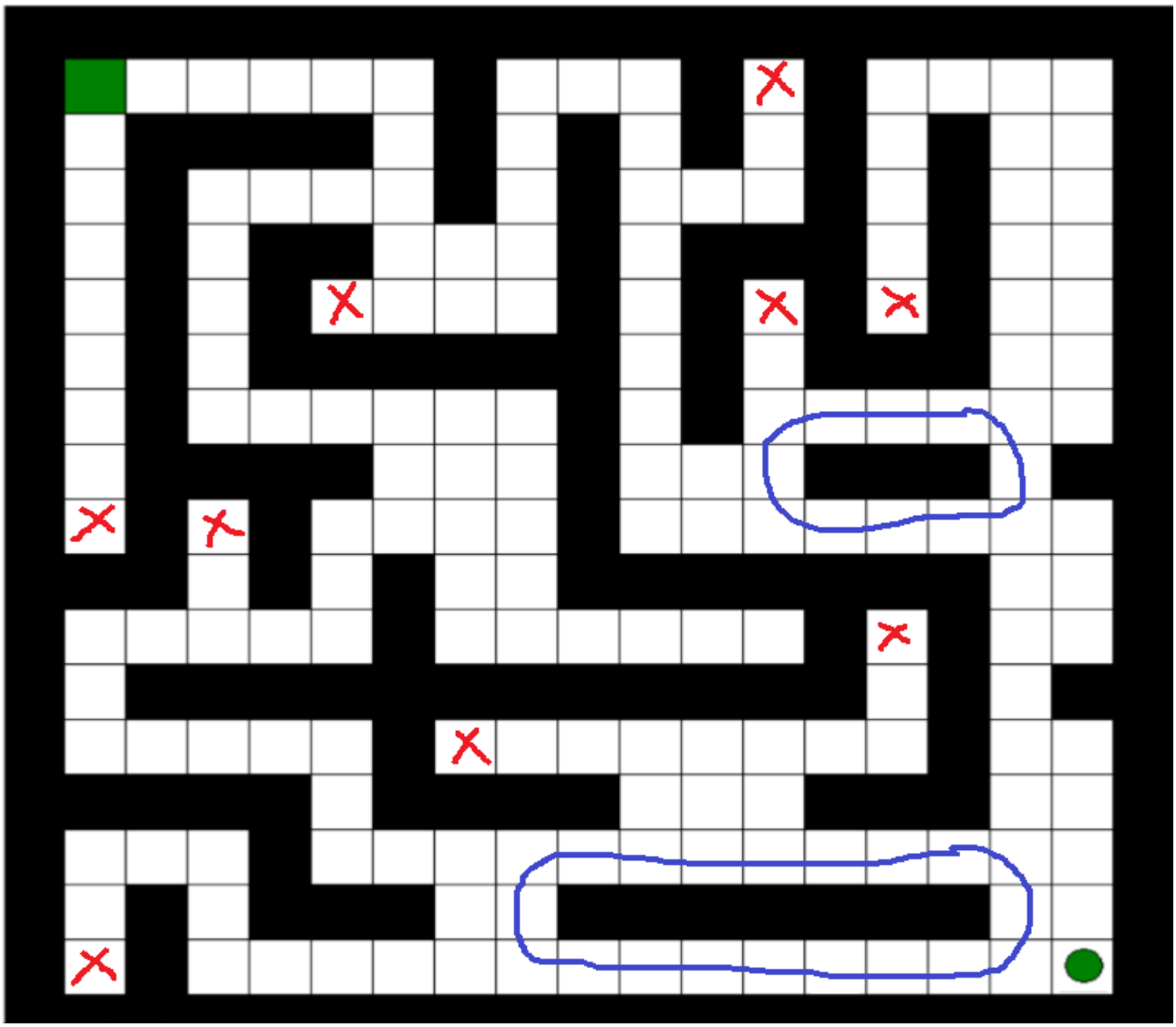
Maze is 19*19 matrix and agent starts from bottom right corner and finishing point is at top left point. Maze is made up of 19 columns ans 19 columns and at every state agent can move left, right, forward, backward. So there are total 19*19*4 =144 possible states that the agent can present in that state.

Agent move to another state if their is no wall between them. If wall is present then stay where it is present.

On every run their is counter which count the number of runs taken by agent .

After every step agent receives reward form environment according to the next state. Agent uses this reward to learn and choose what will be the next state.

Exploratory Visualization



As we seen from above figure , agent is green colour ball and green square is finishing point . Agent might face loops (like blue lines) in this maze and some dead zones (like red crosses).

Loops where agent run around the walls infinite number of times.

Dead zones where agent can not move left , right or forward direction ,last option for agent is to move backward direction.

Algorithms and Techniques

Q Learning Algorithm is used in this project because :
Q algorithm is completely generic. No previous knowledge is assumed about the environment (maze). No training data is needed - the algorithm learns organically. After training, the optimal action for each state is known. The algorithm is mathematically guaranteed to converge .

In this algorithm, agent has finite number of states in which agent can land. At every step the agent take action form current state to new state. On going to new state the agent receives rewards form the environment (it can positive or negative depending on the new state) . Agent train itself such that agent will receive maximum rewards from the environment.

Their is Q table of states and actions, initially it is zero matrix. On training agent make changes in this Q table after receiving rewards. After training agent use this table to choose what next step should be taken. This formula is used to update Q table.

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

General way of Q learning

Initialize the Q table with 0.1 value
while:
    analysis current state
    choose the action form max value in Q table
    take action
    receive reward
    update Q table from above equation

    if reaches finishing point :
        break

Parameter used in Q learning

alpha -It is the learning rate which makes the agent to learn. If it is near to 0 then agent learn very slowly but if it is near to 1 then agent learn very fast. Varies form o to 1
Gamma- It is discount factor . It determines the importance of future rewards. It confirms that the next reward are worth less than current rewards. With this factor algorithm converges. Varies form 0 to 1.
Q table- It is matrix of all states to actions . Initialize with zeros and changes while agent runs.

Benchmark

Benchmark model is the number of steps taken by an agent to reach its destination by using Depth-First Search Algorithm. Our agent should reach the destination by taking less number of steps than a agent which run using depth first search algorithm. It is an integer type from zero(if agent is already on finishing point) to infinite(if agent never reaches its destination). Our goal is to minimize the steps take by agents.

III. Methodology

Data Preprocessing

No need of data preprocessing because all the data provided in making maze and do actions are static and constant.

Implementation

Maze is made up of 19 columns ans 19 columns and at every state agent can move left, right, forward, backward. So there are total 19*19*4 =144 possible states that the agent can present in that state.

****

```
(x, y) = (19, 19)
actions = ["up", "down", "left", "right"]
finishing_point = [(1, 1)]
board = Canvas(mazemap, width=x*widthOfMaze, height=y*widthOfMaze)
agent = (y-2, y-2)

def render_grid():
    global walls, widthOfMaze, x, y, agent
    for i in range(x):
        for j in range(y):
            create whit rectangle boxex

   create finishing rectangle green box
    for (i, j) in walls:
        create balck walls

render_grid()
```

****

Make Q table of all states with respect to all actions ie.. 19*19 rows and 4 columns matrix. Initialize the Q table with 0.1 value. But initialize finishing point Q table with 10.

****

```
Q = {}
for i in range(Environment.x):
    for j in range(Environment.y):
        states.append((i, j))

for state in states:
    temp = {}
    for action in actions:
        temp[action] = 0.1
    Q[state] = temp

for action in actions:
```

```
    Q[(1,1)][action] = 10
```
****

After that we decide that how much agent receive the reward. Reward receives by the agent is -0.05 on every state(except final state) , and on final state agent receive +10 reward.

****

```
if(agent is at final state):
        reward=10
else :
        reward=-0.5
```
****

If agent is not learning then agent took random actions and receive rewards. And these reward output save in SaveWithoutLearning.txt file.

****

```
if(learn==False):
        print("Without Learning Phase")
        for i in range(0,10000):
        do_action(random.choice(actions),False)
        learn=True
```
****

Now ,we have to train the agent by taking actions (choose action which has maximum value in Q table) and receive rewards then update the value in q table . Repeat this process until the agent will get positive result.

****

```
def max_Q(s):
    value = None
    action = None
    for i, j in Q[s].items():
        if value is None or (j > value):
            value = j
            action = i
    return action, value

while True:
        s = Environment.agent
        max_act, max_val = max_Q(s)
        (state, action, reward, new_state) = do_action(max_act,True)

        max_act, max_val = max_Q(new_state)
        UpdateQ(state, action, alpha, reward + discountFactor* max_val)
```

****

Update the Q table while training and saves in Qtable.txt file

****

```
def UpdateQ(s, a, alpha, reward):
    Q[s][a] =Q[s][a]*( 1 - alpha ) +  (alpha * reward)
```

****

Once agent trained (appropriate taking 125 runs) then agent can easily go to the destination point by taking optimal path.

To plot the graphs take the data from ScoreWithoutLearning.txt and ScoreWithLearning.txt file .

****

```
with open('ScoreWithoutLearn.txt','r') as csvfile:
    plots=csv.reader(csvfile,delimiter=' ')
    for row in plots:
        x.append(float(row[0]))
        y.append(float(row[1]))

plt.plot(x,y,label='loaded from ScoreWithoutLearn')

plt.xlabel("Number of steps")
plt.ylabel("Score")
plt.legend()
plt.show()


with open('ScoreWithLearn.txt','r') as csvfile:
    plots=csv.reader(csvfile,delimiter=' ')
    for row in plots:
        x.append(float(row[0]))
        y.append(float(row[1]))

plt.plot(x,y,label='loaded from ScoreWithLearn')

plt.xlabel("Number of Runs")
plt.ylabel("Score")
plt.legend()
plt.show()
```

****

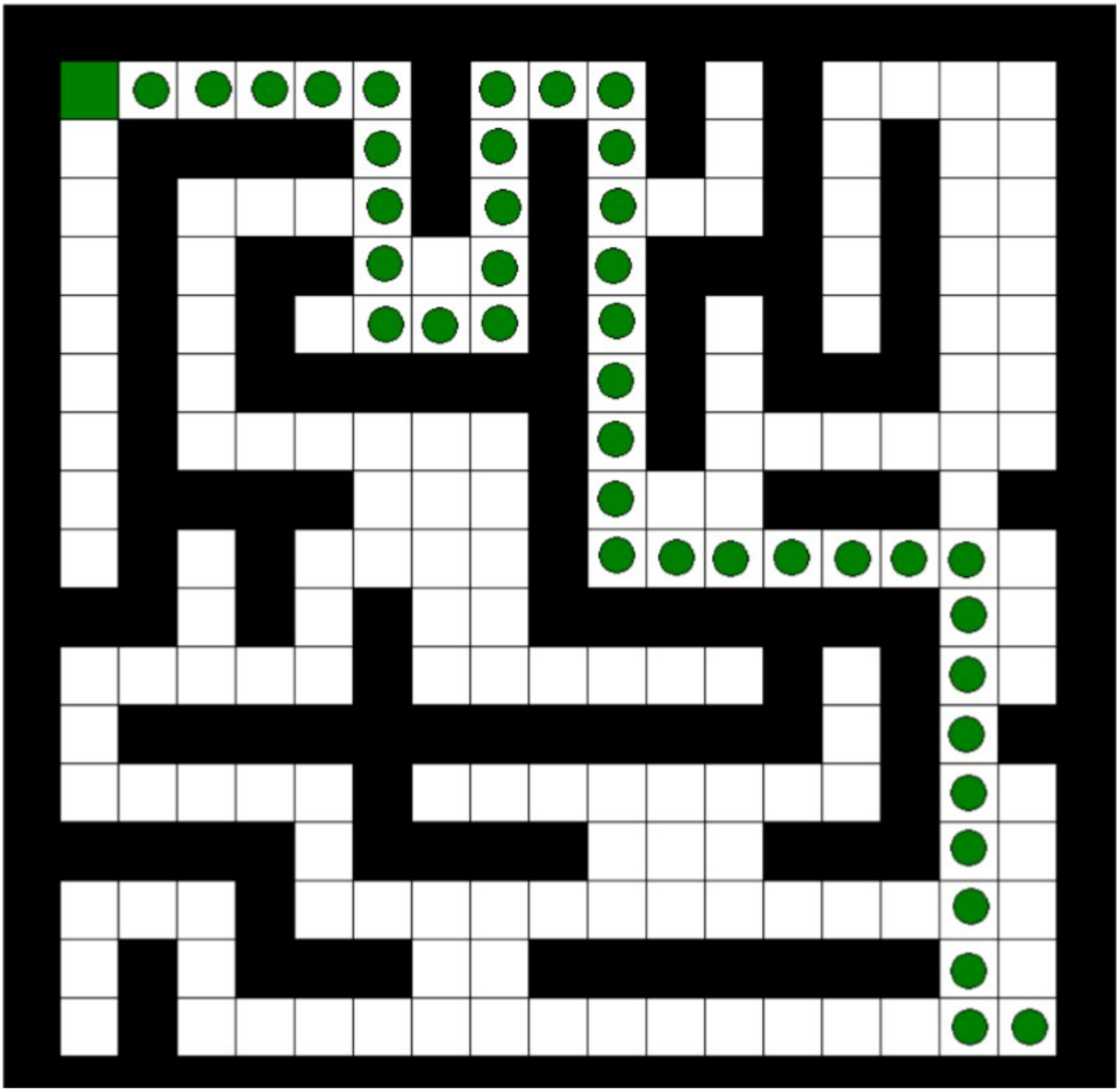
Refinement

Choose a appropriate value of alpha is needed because in my previous model , alpha is close to zero so it takes so much time to learn .Now my alpha is close to one and agent learn at decent rate.

Previously my Q table is initialize with zero but during first updating the Q table values by this equation.
Q[s][a] =Q[s][a]*(1 - alpha) + (alpha * reward)
Value of Q[s][a]*(1 - alpha) becomes zero and Q table did nit change significantly. So,I also initialize Q table with value 0.1.

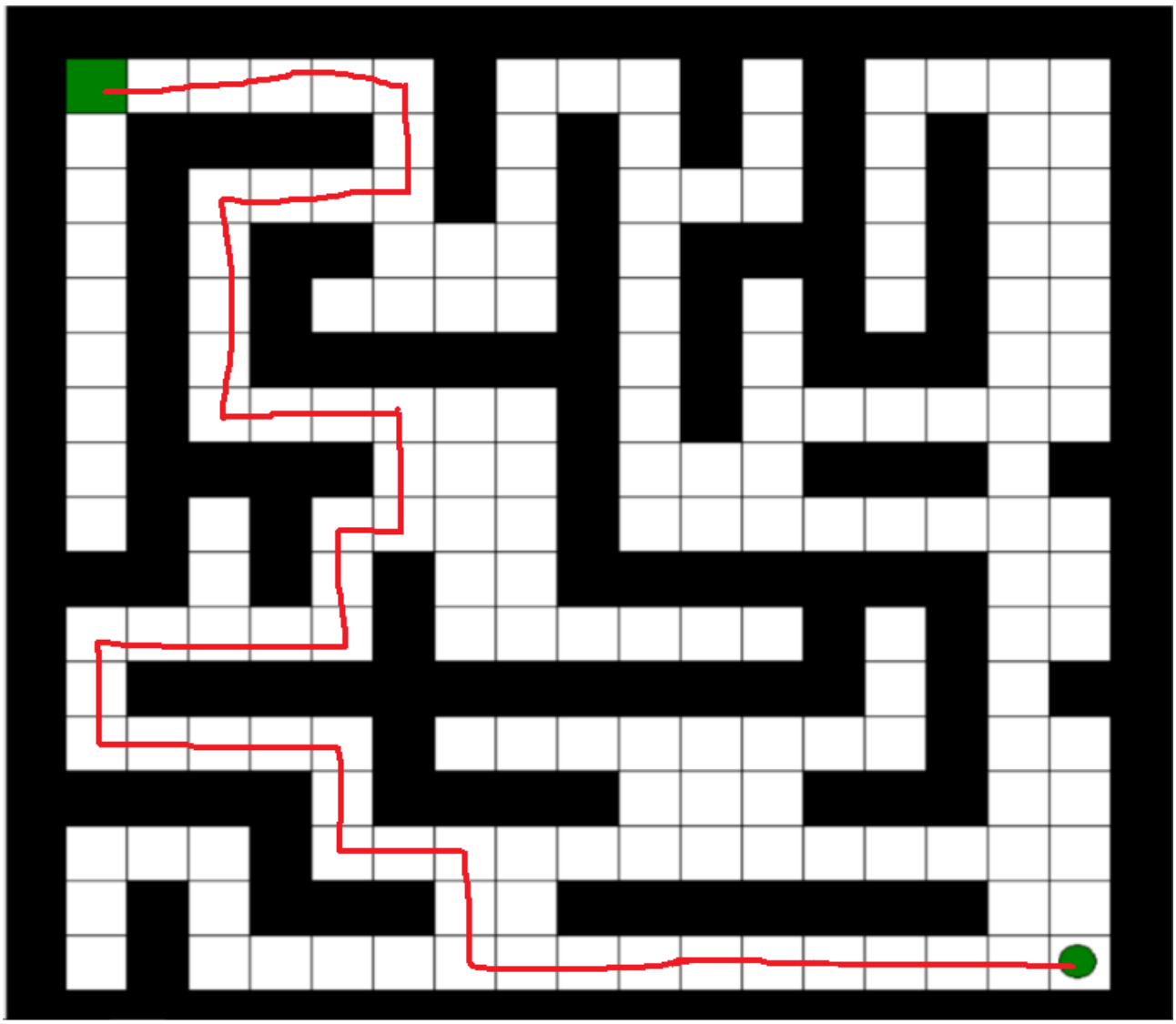
Rewards that agent getting from environment. Like when reaching finishing point agent should have a final score greator than zero . Agent receive reward which is greater than the (total number of steps) * (reward getting on state)-(initialized score) . Optimal number of steps is 40 so, reward at finishing point should be greater than (39*0.05)-(1)=1 So, that agent should know clearly difference between final state and rest states.

IV. Results

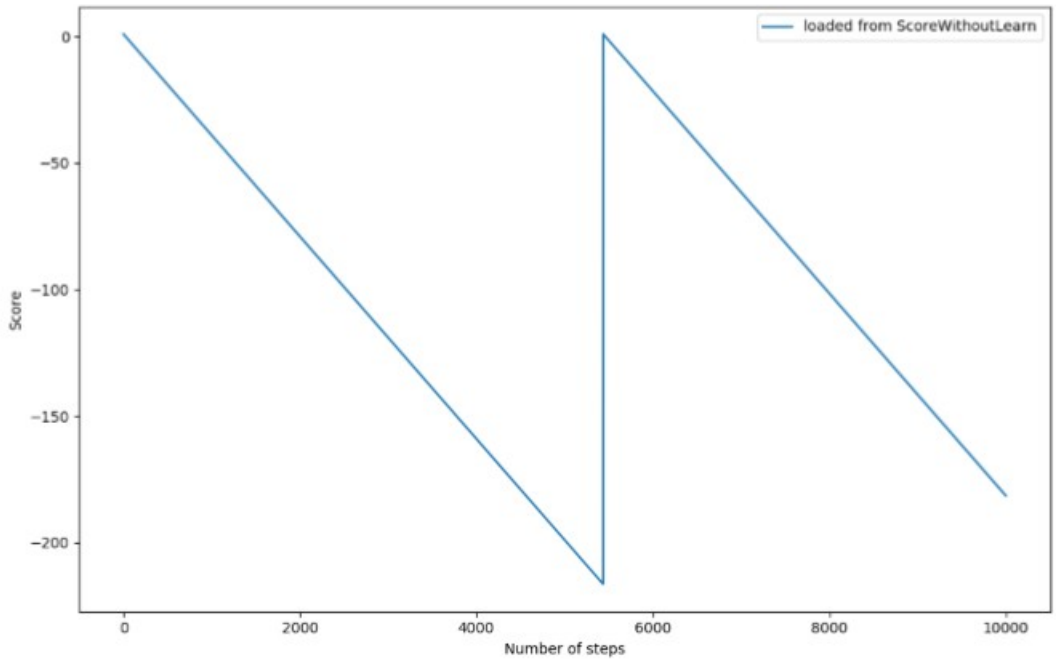


Model Evaluation and Validation

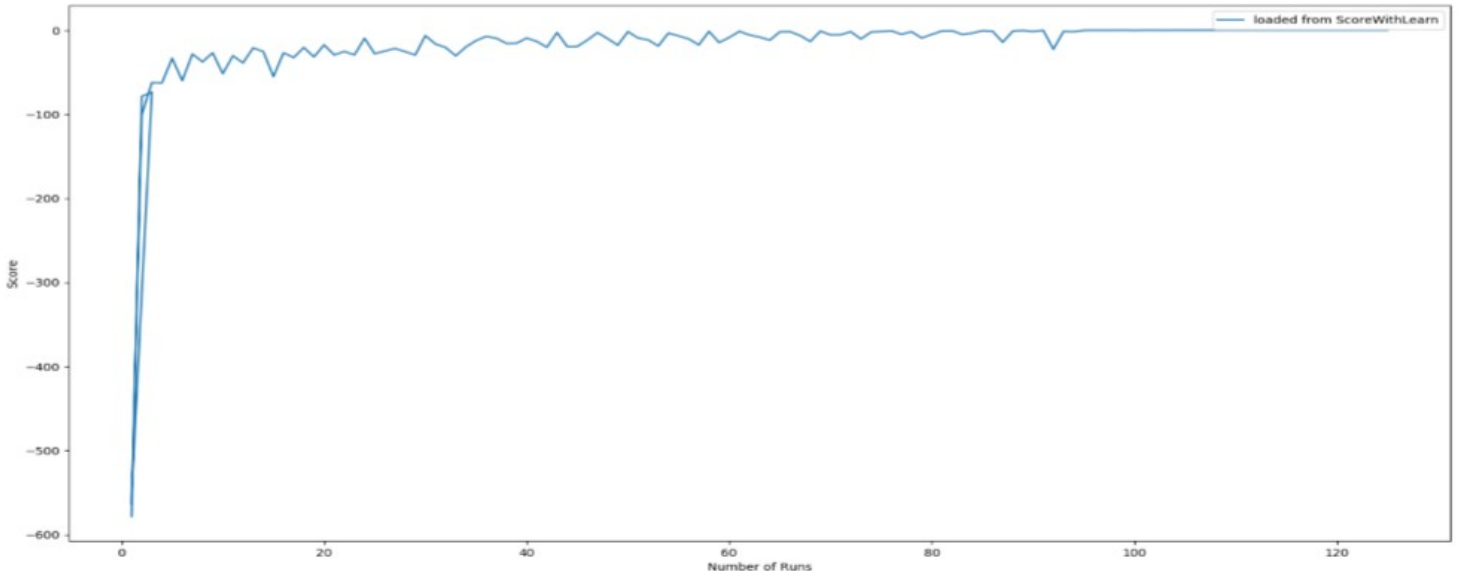
Now the agent know the optimal path to destination so after that agent reaches the destination by 100% accuracy.



As we seen form above figure their is another way to reach the destination by agent did not choose this path because by taking this path agent can reaches the destination in 48 steps which is more than the previous path.

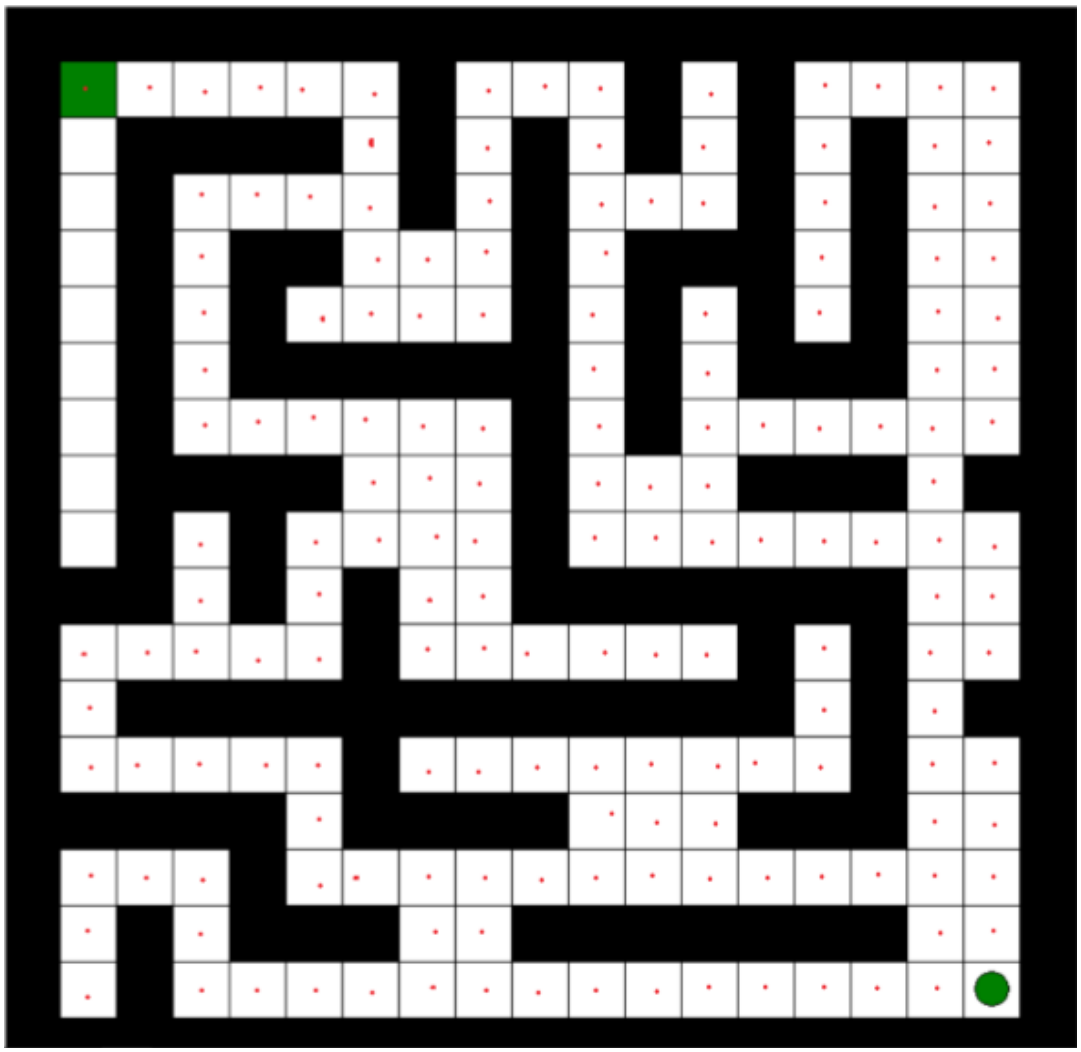


Above figure shows that when agent is taking random steps then agent can never reach(approximately) to the destination in 10000 steps.



Above figure shows that when agent is taking random steps then agent can never reach(approximately) to the destination in 10000 steps.
Above figure shows that while traing agent took 125 runs to get the optimized path of maze.

Justification



Number of steps taken by agent using Q learning is 40 steps which is smaller than if agent solve this maze using depth-first search algorithm. Using Depth-First search algorithm agent uses 169 stpes to reach destination.

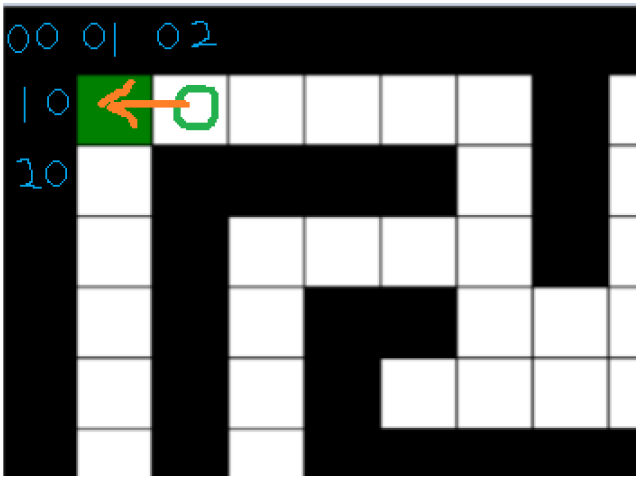
V. Conclusion

Free-Form Visualization

Q table is important in this algorithm by which agent can choose which path will take to your destination . This Q table updated while learning phase according to reward receives by environment. This Q table is 19*19 matrix present in Qtable.txt file.

Some values of Q table with respect to their actions.

On line 41 in Q table ie.. position is 1,2 (row, colomn)



"up"	"down"	"left"	"right"
0.04263675086926559	0.04264032367926633	4.0	0.1

Maximum value form above value if 4.0 which show agent takes left action which is correct step.

Reflection

This algorithm waste so much of time while exploring maze through random states so that it reaches its destination. The Q table requires so much of memory because every state has 4 actions so, total number of combinations is rows*columns*4 which is huge is if maze has greater than 500 columns or rows .

Improvement

Getting value of rewards and iterate over all states is not possible if maze if big.
To solve this problem value of Q[state][action] can be predict with the help of Deep convolutional network which is Deep-Q-Learing algorithm.