National University of Singapore

ME5404 Neural Network Part II

Project 2

Q-Learning for World Grid Navigation

Name : Chua HongWei

Matriculation Number: A0074741E

Email: e0823141@u.nus.edu

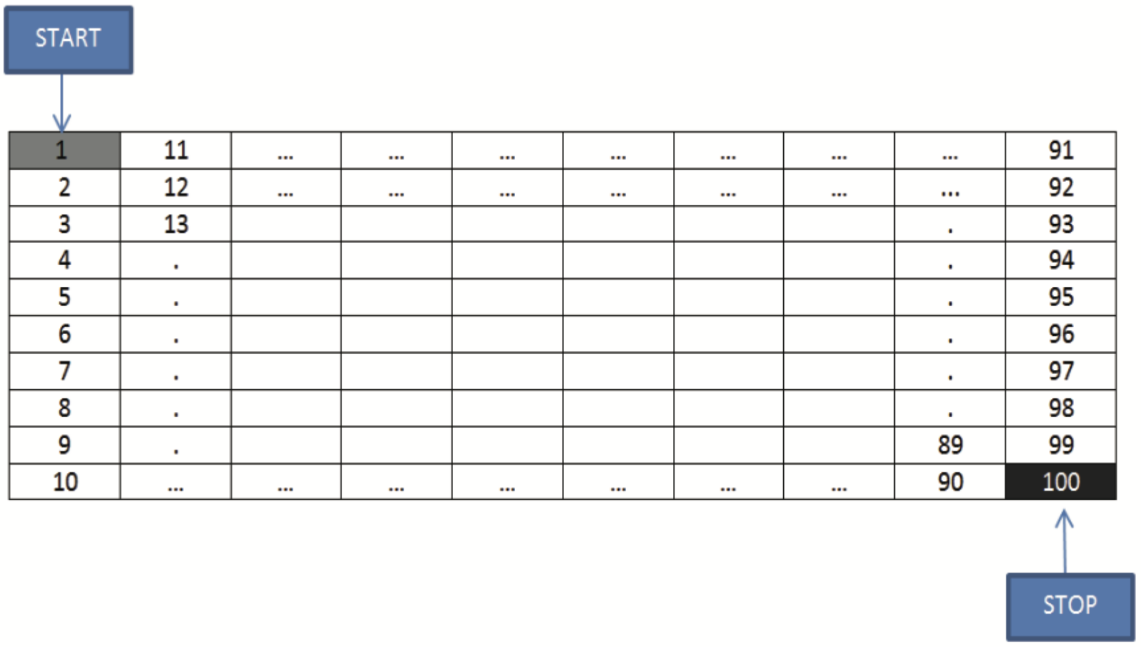
11 Apr. 22

1. Introduction

Reinforcement learning (RL) is slowly gaining traction in the world of machine learning due to its ability to self-learn from agent's past experience with only reward provided. It is proven to be versatile and efficient way of solving various AI task especially in the field of robotics. Reinforcement learning can be divided into model-based reinforcement learning and model-free reinforcement learning. In this task, we will look into one of the model-free RL method, Q-learning and its ability to solve a 10X10 world grid navigation problem.

2. Project Task

In this problem, we needs to simulate the movement of a robot across a 10X10 world grid starting from start state (1) to reach the goal state (100). An illustration of the grid world is shown below.



1	11	...	...	...	...	...	...	...	91
2	12	...	...	...	...	...	...	...	92
3	13							.	93
4	.							.	94
5	.							.	95
6	.							.	96
7	.							.	97
8	.							.	98
9	.							89	99
10	...	...	...	...	...	...	...	90	100

Figure 1 Illustration of 10x10 Grid World

At each state, the robot is able to move up (action = 1), right (action = 2), down (action = 3) and left (action = 4). The robot is bounded by the 10X10 grid world and is not able to move beyond the bounded environment. In this task, we are supposed to code the movement of the robot based on Q-learning update rules and experimenting on different learning rates and epsilon greedy methods.

3. Project Algorithm

Reward Matrix

A reward matrix of [100, 4] is given for this project with the rows representing state and each column indicating action 1 to 4, thus making each entry in the matrix a state- action pair. Upon further investigation, I found that all '-1' reward indicating actions which lead to robot moving of bound. With a simple find function in Matlab, I could determine the valid action for each state and ensure that the robot always stay within the grid world.

Q-Learning

Q-learning is the reinforcement learning method used for this grid world navigation. Q-learning is useful when the state-transition model is not known to the agent. The agent will learn through the reward obtained from each trial. Q-learning uses an update rule to update the estimated Q value with the temporal difference. Temporal difference is formulated based on the Bellman's equation. The Q-learning update rule is given as follows, where α is the learning rate and γ is the discount factor.

$$Q_{k+1}(s_k, a_k) \leftarrow Q_k(s_k, a_k) + \alpha(r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k)$$

Epsilon-Greedy Method

As Q-learning required all state-action pair to be visited indefinitely, the agent needs to explore more state to have a more accurate Q-learning estimation. However, it is also important for the agent to exploit a good policy towards the end of the trial. This balance can be achieved via epsilon greedy method, where agent is encourage to explore as many state as possible in the early stage of the trial and slowly exploit learned policy towards the end of the trial.

In this project, four different epsilon greedy method is experimented. An additional epsilon decay method is also implemented by me, which would be used for task 2. A list of epsilon greedy decay method used in this project is as follows.

1) $^1/_k$
2) $^{100}/_{100+k}$
3) $^{1+log(k)}/_k$
4) $^{1+5log(k)}/_k$
5) $e^{(-0.001k)}$, where 0.001 is the decay rate

Epsilon-greedy method is implemented in Matlab with the cumsum function shown below.

```
r = rand;
x = sum(r >= cumsum([0, 1-epsilon, epsilon]));
```

If x equals to one, the algorithm will choose the greedy action which exploit the action with highest Q-value, else the algorithm will explore all other actions besides the greedy action. The learning rate is set to be same as epsilon and follows the same decay equation.

Training Flowchart

The whole training process for Q-learning in this grid world is summarize in the flowchart below. Each trial follows the following flow chart.

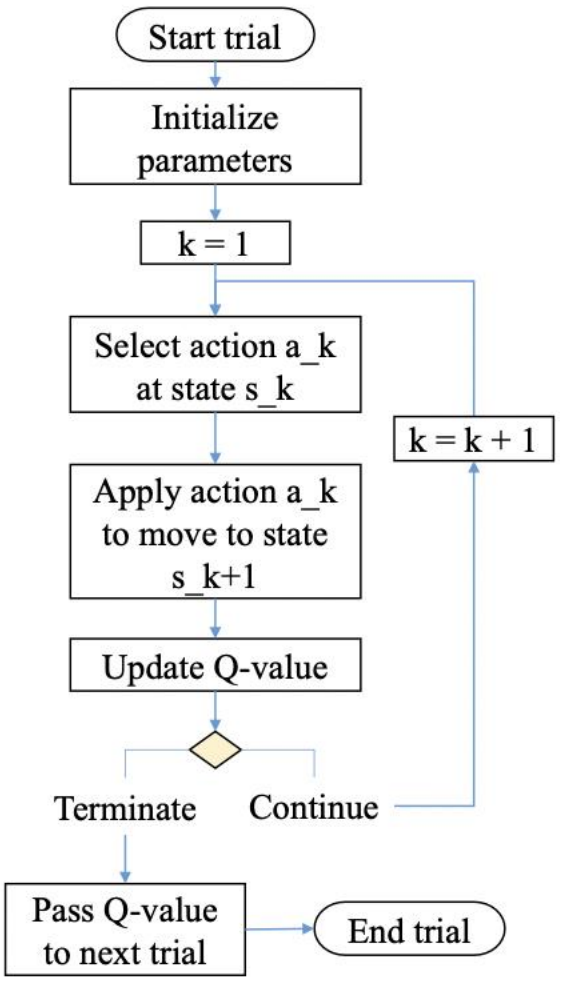

Start trial
Initialize parameters
k = 1
Select action a_k at state s_k
Apply action a_k to move to state s_k+1
Update Q-value
k = k + 1
Terminate
Continue
Pass Q-value to next trial
End trial

Figure 2 Flowchart of Each Q-learning Trial

Each trial ended if any of the following condition is satisfy

1) Maximum trial of 3,000 episodes has been reached.
2) Q-table value converged. In this project, Q-value converge if the difference of current Q-value and the previous trial is less than 0.001.

In this project, the program will be run 10 times and all the Q-learning policies will be recorded as per figure 3. A function named get_policy is coded to analyze all the 10 Q-learning runs. The Q-learning runs will be labelled as optimal policy if the Q-learning policy managed to reach state 100 without visiting a repeated state in one run. The number of run which is optimal. It's execution time and it's corresponding rewards collected will be recorded.

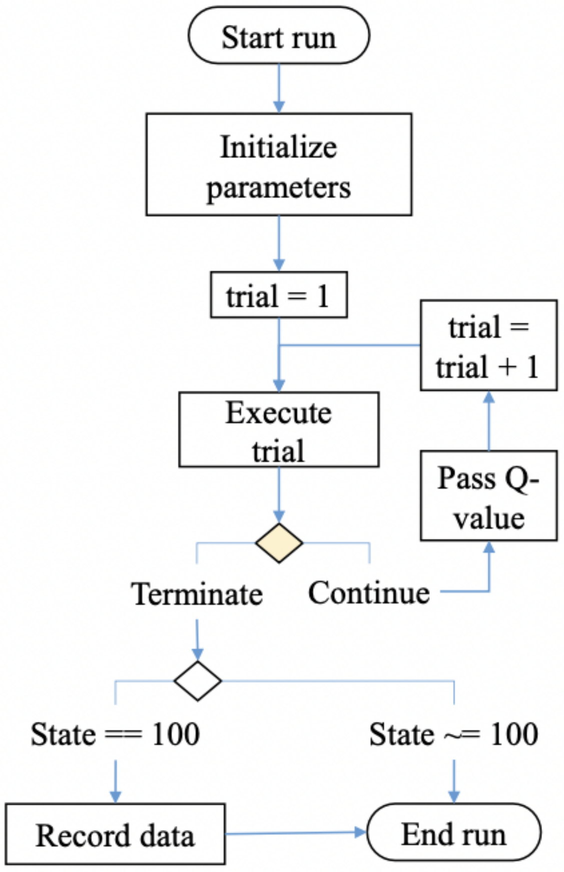


Figure 3 Flowchart of Each Q-learning Run

Finally, the optimal policy which yield the maximum reward will be plotted in three ways described in the project sheet.

4. Results

ε_k, α_k	No. of goal-reached runs		Execution time (sec.)	
	$\gamma = 0.5$	$\gamma = 0.9$	$\gamma = 0.5$	$\gamma = 0.9$
1 / k	NA	NA	NA	NA
100 / (100 + k)	NA	10	NA	1.1772
(1 + log(k)) / k	NA	NA	NA	NA
(1 + 5log(k)) / k	NA	9	NA	8.399

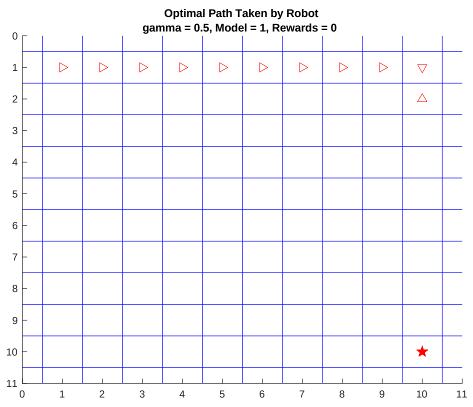
* NA = No Optimal Policy

5. Results Discussion

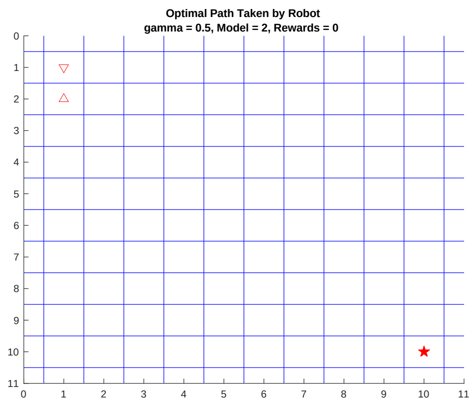
<u>Effect of Learning Rate, γ</u>

<u>γ = 0.5</u>

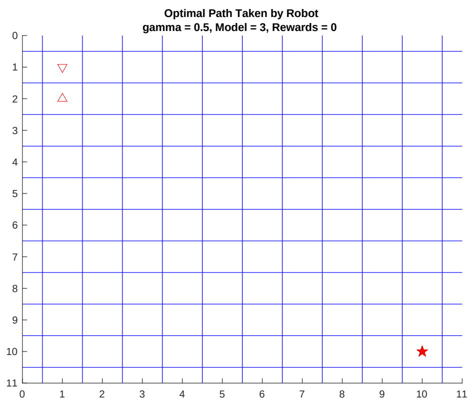
From the result above, with learning rate, γ of 0.5, optimal policy is not achieved. Learning rate of 0.5 encourage the agent to only looks for immediate reward and put less consideration on future rewards. This cause the agent to only look at short term gain and visit a repeated state and thus fail the optimal policy criteria. The following four plots shows the path taken by the agent when γ = 0.5 for all 4 epsilon greedy methods proposed.



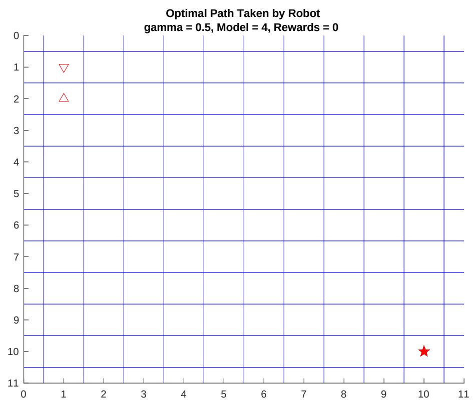
(a) ε = 1/k



(b) ε = 100 / (100 + k)



(c) ε = (1 + log(k)) / k



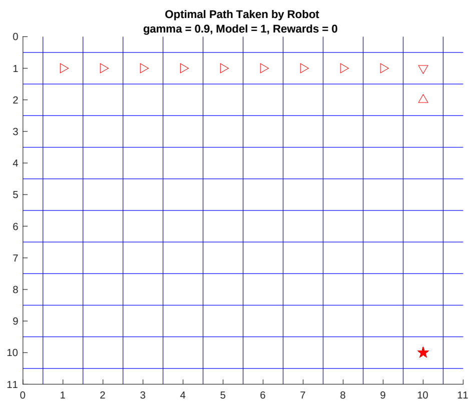
(d) ε = (1 + 5log(k)) / k

Figure 4 Plots of Path Taken by Agent for Different Epsilon for γ = 0.5

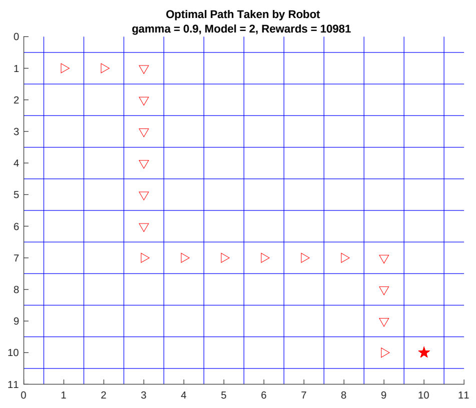
As observed from Figure 4, all agent's movement stop abruptly after it visit the same state it went through before. This shows that the agent is short sighted and is insufficient to complete the movement towards state 100.

<u>γ = 0.9</u>

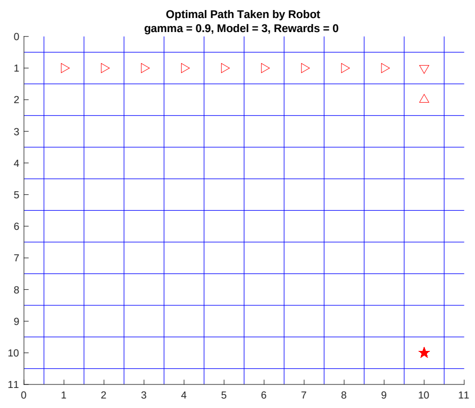
Similar plots as per Figure 4 but with γ = 0.9 were plotted and shown below.



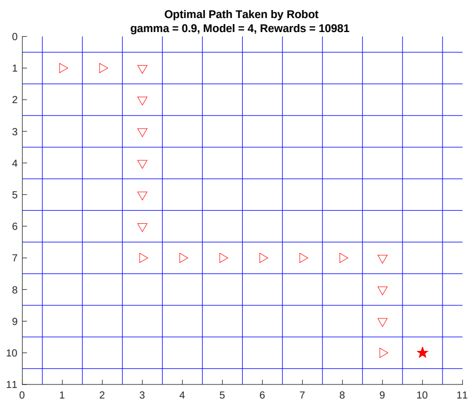
(a) ε = 1/k



(b) ε = 100 / (100 + k)



(c) ε = (1 + log(k)) / k



(d) ε = (1 + 5log(k)) / k

Figure 5 Plots of Path Taken by Agent for Different Epsilon for γ = 0.9

When the γ is set to 0.9, the agent is more far sighted and would sacrifice short term gain for better reward in the future. This is shown clearly on 5(b) and 5(d) where the agent is able to find an optimal policy and reach state 100. Even if optimal policy is not obtained such as in 5(a) and 5(c), the agent is able to travel further when compare to γ = 0.5.

<u>**Effect of Epsilon, ε**</u>

Next, we will investigate the effect of different epsilon greedy method on the optimal policy. We would only consider cases where γ = 0.9, as the agent only obtained optimal policy when γ = 0.9. From the result table in Chapter 4, only when ε = 100 / (100 + k) and ε = (1 + 5log(k)) / k, the optimal policy is achieved. This can be explain by the plot below.

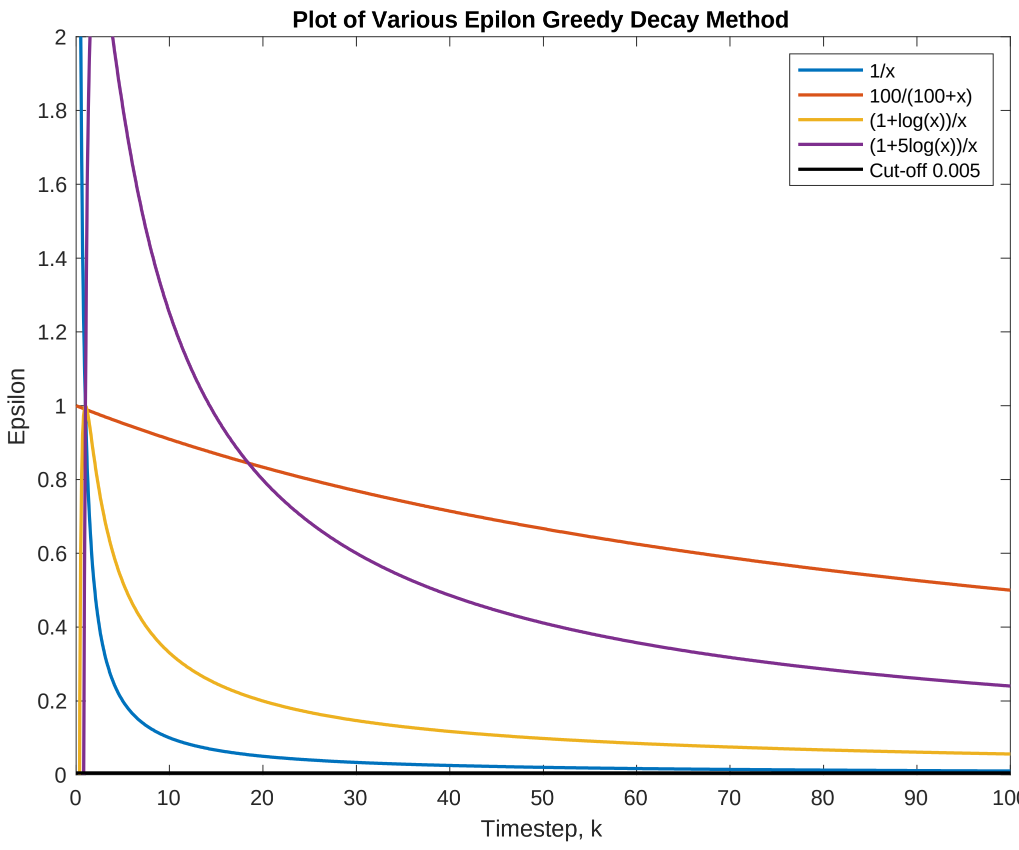
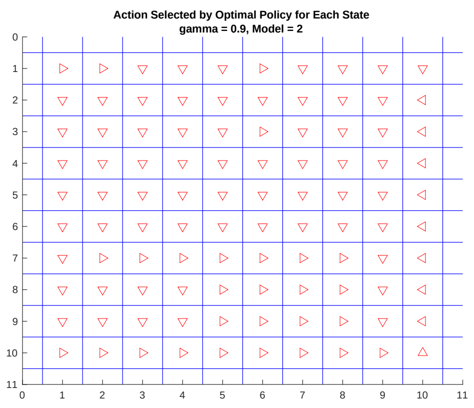


Figure 6 Various Epsilon Decay Method Vs Time Step

From Figure 6, it is observed that the epsilon decay method must show a slower decay rate as the time step k increases. This is because if the decline is too steep, the alpha and epsilon decay to below the threshold of 0.005 in a very short time step and end the trial early. Also, a steep decay in epsilon also reduce the exploration time for the agent and thus the agent has a bad understanding of the environment and less accurate estimation of the Q-value.

The path taken by the agent for ε = 100 / (100 + k) and ε = (1 + 5log(k)) with γ = 0.9 is shown below.



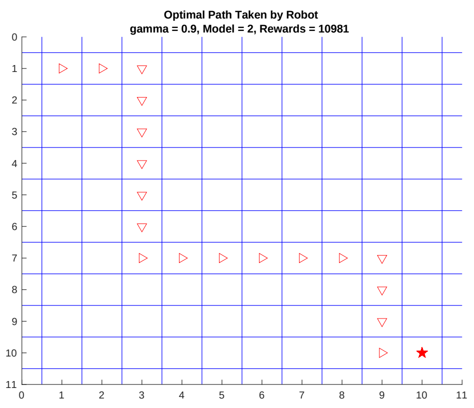

Optimal Path Taken by Robot
gamma = 0.9, Model = 2, Rewards = 10981

Figure 7 Plots of Action Selected for all States and Path Taken by Agent for ε = 100 / (100 + k)

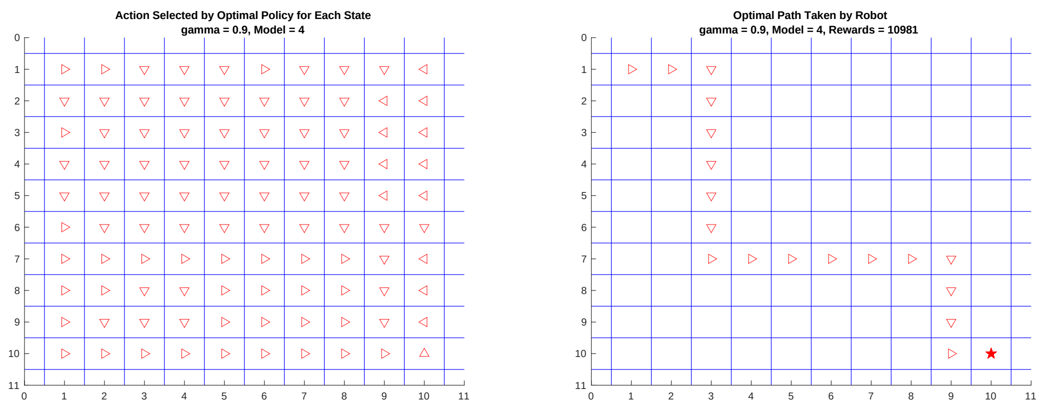


Figure 8 Plots of Action Selected for all States and Path Taken by Agent for ε = (1 + 5log(k))

6. Further Investigation

I further tried the epsilon greedy method with exponential decay $e^{(-0.001k)}$, where 0.001 is the decay rate. The results is as follows.

ε_k, α_k	No. of goal-reached runs		Execution time (sec.)	
	$\gamma = 0.5$	$\gamma = 0.9$	$\gamma = 0.5$	$\gamma = 0.9$
$e^{(-0.001k)}$	NA	10	NA	0.60573

From the table above, exponential decay method at $\gamma = 0.9$ can achieve optimal policy in all 10 runs and at a faster execution time than all the other methods provided. This means that the Q-table converge to optimal policy faster with the exponential decay method. Path taken by the agent with this decay method is shown below.

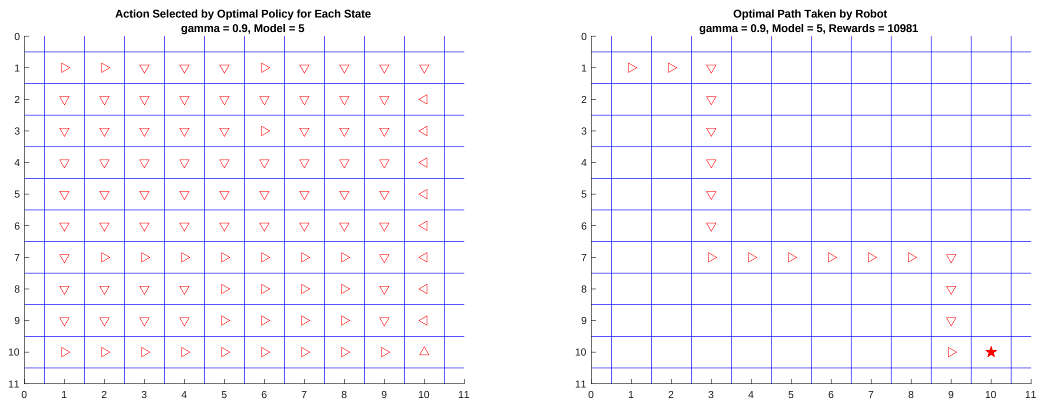


*Figure 9 Plots of Action Selected for all States and Path Taken by Agent for ε =e^(-0.001k)^*

This exponential epsilon decay method together with $\gamma = 0.9$ is chosen as the parameter for the task 2.

7. Code File

RL_task1.m - Matlab file for task 1.

RL_main.m - Matlab file for task 2.

epsilon_plot.m – Plot of various epsilon decay method.