PROJECT REPORT

CSE574
INTRO TO MACHINE LEARNING

PROJECT 4

BY
YASH AHUJA
50245092
yashahuj@buffalo.edu

1. Introduction

In this machine learning project, the basics of reinforcement of is understood and learned. It is the combination of reinforcement learning and Deep Learning (Deep Neural Networks). The purpose of this project is to understand and implement Deep Q-learning, and reinforcement learning in different environments and get a feel of it. In this project, the agent (Tom) is supposed to explore and find a path to get to the goal (Jerry). This project also uses discreet values and a 5x5 grid.

- **About Reinforcement Learning**
  Reinforcement Learning is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences. In reinforcement learning the goal is to find a suitable action model that would maximize the total cumulative reward of the agent.

The main elements used in this project and that are also a major part of reinforcement learning are:

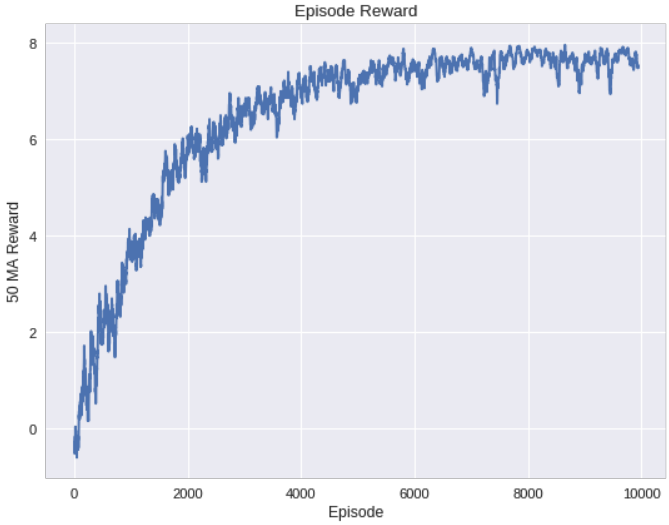
1. The Environment: This is where the game is played. It can be in many forms, example, a grid (like in this project), or images (like in Atari or Stable-baselines)
2. Agent: This is the element which is trained. The responsibility of this element is to find a path (in this case) or a way to get/achieve the goal.
3. Goal: This is supposed to be achieved/caught by the agent.
4. State: This is the phase/situation that the agent is in.
5. Reward: It is the feedback obtained from the environment regarding the progress of the agent.
6. Policy: This is the method to map the agent's state to its action.
7. Q-Value: Future reward to be received by the agent based on the future action.

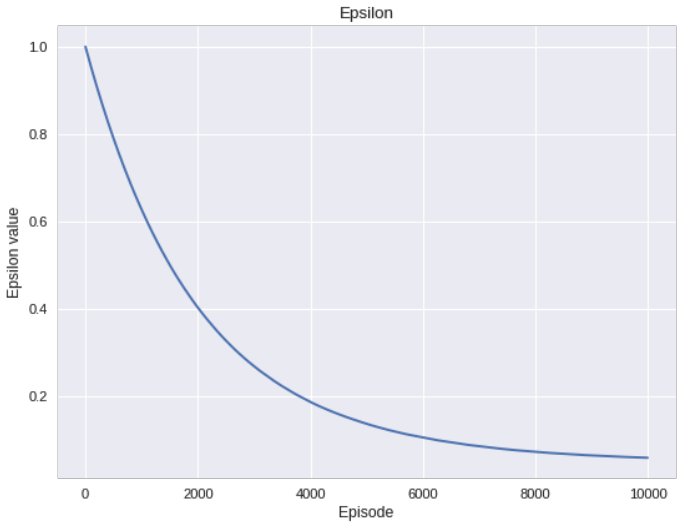
DEFUALT MODEL:

The default model was run with the following parameters:

SI.No	Parameter	Value
1.	Max Epsilon	1
2.	Min Epsilon	0.05
3.	Number of Episodes	10000
4.	Gamma	0.99

The following graphs were obtained:





Comment: As seen from the first figure, the mean rewards increase with increase in number of episodes, but saturates at 6000 episodes. 10000 episodes were taken to ensure that the mean rewards don't fluctuate and remain constant for a certain time. So, we don't take 6000 and instead we take 10000.

As seen from the second figure, our epsilon value is exponentially distributed and decaying as the number of episodes increase.

The mean rewards noted is **6.4095.**

The time taken to train the agent is **775.59** seconds.

NOTE: The mean reward above, and in the project, calculated is the mean of the last 100 rewards of the particular cycle.

2. CODING TASK – Questions to be answered

Question 1 & 2 – What parts have you implemented? What is their role in training the agent?
Answer: There were three parts implemented in the code – Neural Networks, Epsilon formula (using exponential decay function), and the Q-function. Further explanation is given below:

- *Neural Networks:* Neural networks was implemented using the keras library. It helps to agent decide what action to take next. The default code was run using two hidden layers with 128 nodes each. The input dimensions for the first hidden layer is the size of observation space and the output dimensions, the size of the action space. 'Relu' is used as activation function for the first two hidden layers and 'linear' for the output layer. Neural networks can approximate any continuous differentiable function since they are known to be universal function approximators. It might be possible for them to get stuck in local extrema, still many proofs of convergence from reinforcement learning are not valid anymore when throwing neural networks in the equation. Although, in most cases, they can be very powerful with the correct, finely tuned hyperparameters.
- *Epsilon decay:* In this project, the epsilon value used the function of exponential decay. The epsilon value is quite significant so as to avoid the agent greedy actions. If the random value is lower than the epsilon value, then the agent will pick a random action to go forward, and if it is vice versa, the agent will pick the best action to take in order to maximize the rewards. As the epsilon value decreases, the number of random moves decreases, therefore helping us to reduce the number of random moves taken by the agent to reach the goal. Additionally, it decreases the discount factor(gamma) and penalizes the further rewards in order to strategically make the move.
- *Q-Function:* This is what makes the agent take the steps to reach the goal. The past experiences gets collects in the memory and forms the training set, which helps the agent make better moves. The agent uses its past experiences to learn the optimal policy, which help get the optimal actions to receive optimal rewards. This is a value function which is estimated by learning this policy.

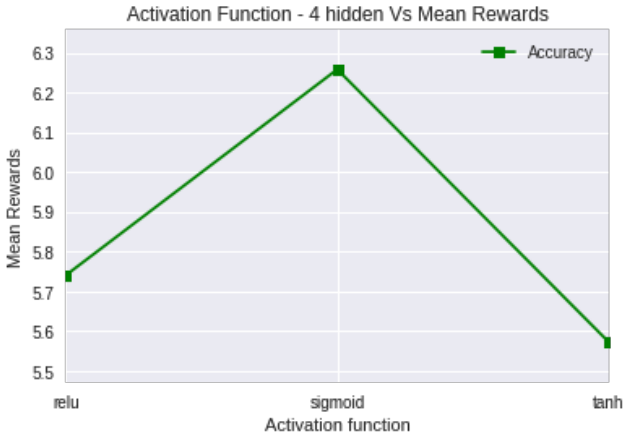
Question 3 – Can these snippets be improved and how it will influence training the agent?

Answer: Yes. The snippets can be improved by tuning the hyperparameters of each snippet respectively. It is a trial and error method, because tuning the hyperparameters can get you either a better or worse value. It also depends on which hyper-parameters you tune and how you tune it. In my case, I have tuned the hyper-parameters of the first two snippets, as instructed.
POINT TO BE NOTED: Default mean rewards for the last 100 values are taken and is 6.4095

(i) Tuning the hyperparameters of the neural network snippet
I have done the following:

(a) Run the NN with 4 hidden layers instead of 2
(b) Run the NN with 5 hidden layers instead of 2

(a) Run the NN with 4 hidden layers instead of 2

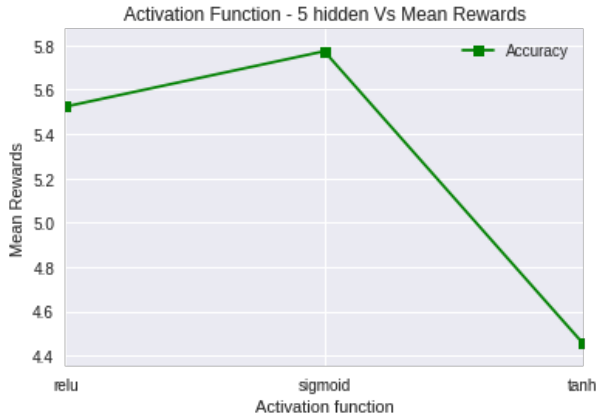


Comments: As noticed from the graph, for every input function, the mean reward is less than the default mean reward ~ 6.4095.
Only the input activation layer (1st hidden layer) has been varied here.

S. No.	Number of hidden-layers	Input activation function	Time Elapsed
1	2 – *Default*	relu	785.50
2	4	relu	947.37
		sigmoid	934.28
		tanh	947.39

Note: The Activation Function values taken were [‘relu’, ‘sigmoid’, ‘tanh’]. As observed, the time taken for training is higher and the mean rewards are less, hence this is not advisable.

(b) Run the NN with 5 hidden layers instead of 2

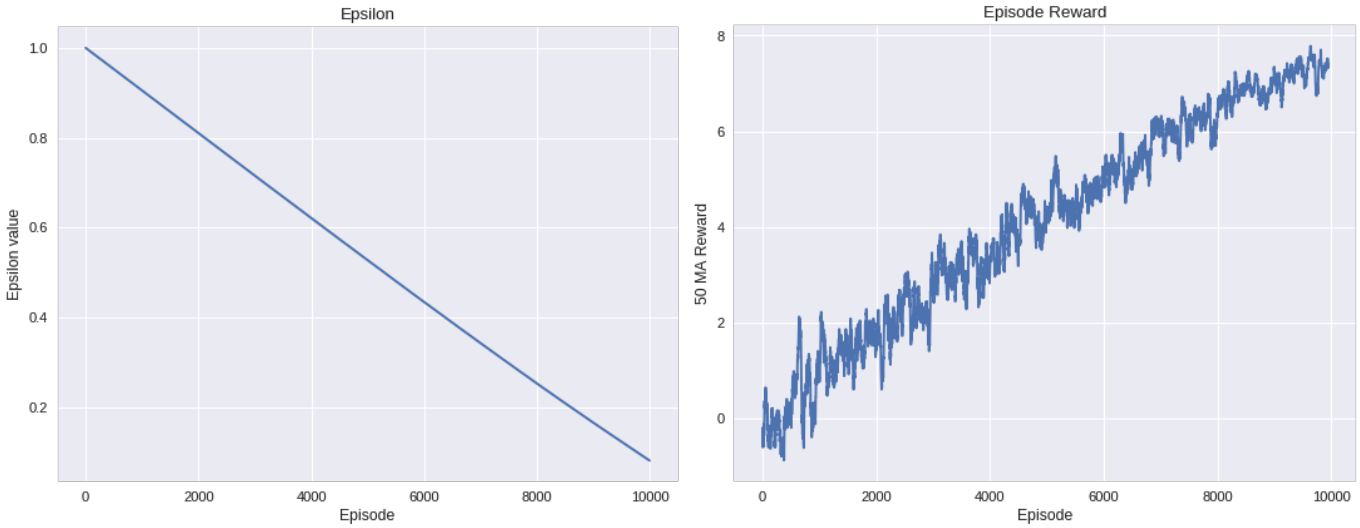


Comments: As noticed from the graph, for every input function, the mean reward is less than the default mean reward ~ 6.4095.
Only the input activation layer (1st hidden layer) has been varied here.

S. No.	Number of hidden-layers	Input activation function	Time Elapsed
1	2 – *Default*	relu	785.50
3	5	relu	1093.45
		sigmoid	1183.48
		tanh	1214.46

Note: The Activation Function values taken were [‘relu’, ‘sigmoid’, ‘tanh’]. As observed, the time taken for training is higher and the mean rewards are less, hence this is not advisable.

(ii) Changing the function of epsilon from exponential to linear
After changing the function, the following graphs are obtained:



Comments: The time taken to train is 820 seconds which is greater than the default time taken to train the agent with the epsilon value having the exponential function (775.59 seconds). The mean rewards are increasing at a slower rate with the linear function when compared to the exponential function.

The above to ways can be used to help influence the training of the agent.

Question 4: How quickly your agent was able to learn?

Answer: Tuning the number of episodes shines some light into this area.

SI. No	Number of Episodes	Training time in seconds
1.	1000	81.93
2.	2000	169.85
3.	3000	263.09
4.	4000	349.92
5.	5000	441.35

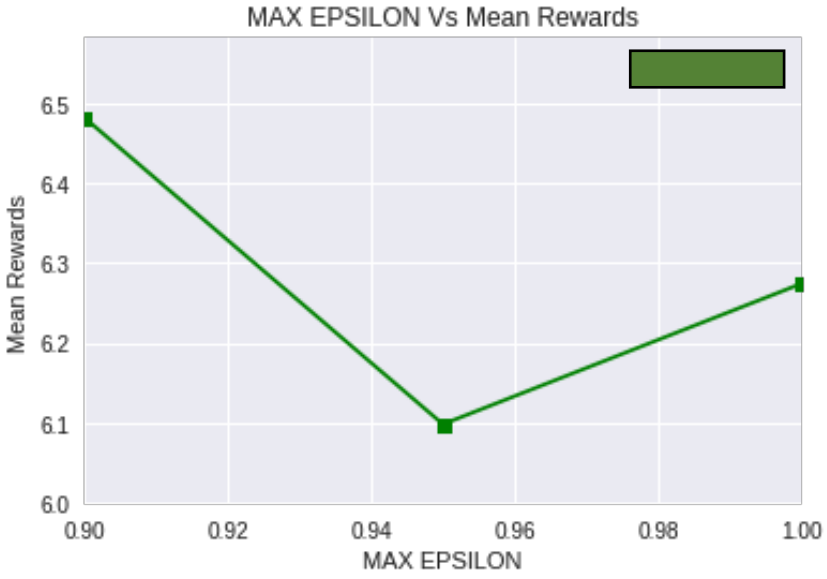
Comments: It also can be observed that at '*number of episodes – 5000'*, the time taken to train the agents was the most. Hence there is a complete trade-off between the training time and the mean rewards. You have to choose if you want poor results and less training time or good results and a higher training time, in this case.

3. Hyperparameter tuning with plots

There are numerous hyper-parameters tuned in this project, and they are as follows:

- MAX EPSILON,
- MIN EPSILON,
- Number of Episodes, and
- Gamma

(i) MAX EPSILON



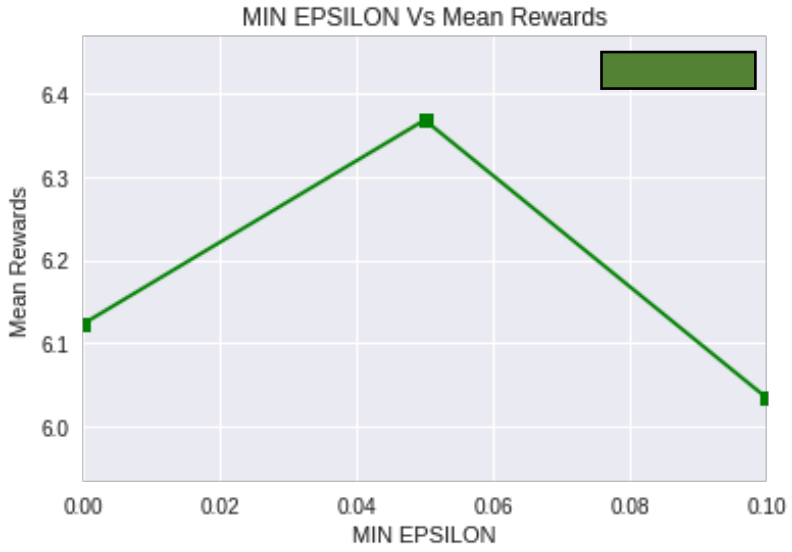
Comments: The hyper-parameter values, max-epsilon, are varied in this graph. As observed, the highest mean rewards obtained are at the *max-epsilon value – 0.9*. The total time taken for training the agent at each value of maximum epsilon is given as follows:

SI. No	Max Epsilon value	Training time in seconds
1.	0.9	819.77
2.	0.95	838.22
3.	1	830.06

It also can be observed that at *'max-epsilon = 0.9'*, the time taken to train the agents was the least. So, it would be more beneficial to select 0.9 as Max Epsilon and there

Note: The Max Epsilon value taken were [0.9, 0.95, 1]

(ii) MIN EPSILON



Comments: The hyper-parameter values, min-epsilon, are varied in this graph. As observed, the highest mean rewards obtained are at the *min-epsilon value – 0.05*. The total time taken for training the agent at each value of maximum epsilon is given as follows:

SI. No	Min Epsilon value	Training time in seconds
1.	0	829.72
2.	0.05	848.24
3.	0.1	863.25

It also can be observed that at *'min-epsilon = 0.9'*, the time taken to train the agents was NOT the least. Hence there is a trade-off between the training time and the mean rewards. Since there is a drastic change in the mean rewards between all the values of minimum epsilon, it is better to select 0.05 to maximize the output.

Note: The Min Epsilon value taken were [0, 0.05, 0.1]

(ii) Number of Episodes

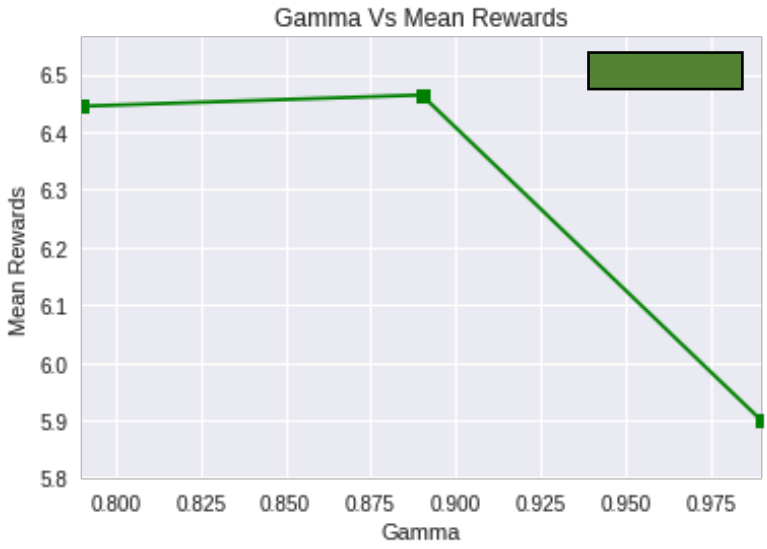
Comments: The hyper-parameter values, number of episodes, are varied in this graph. As observed, the highest mean rewards obtained are at the *number of episodes – 5000*. The total time taken for training the agent at each value of maximum epsilon is given as follows:

SI. No	Number of Episodes	Training time in seconds
1.	1000	81.93
2.	2000	169.85
3.	3000	263.09
4.	4000	349.92
5.	5000	441.35

It also can be observed that at '*number of episodes – 5000'*, the time taken to train the agents was the most. Hence there is a complete trade-off between the training time and the mean rewards. You have to choose if you want poor results and less training time or good results and a higher training time, in this case.

Note: The Number of Episodes taken were [1000, 2000, 3000, 4000, 5000].

(ii) Gamma



Comments: The hyper-parameter values, gamma, are varied in this graph. As observed, the highest mean rewards obtained are at the *min-epsilon value – 0.89*. The total time taken for training the agent at each value of maximum epsilon is given as follows:

SI. No	Gamma	Training time in seconds
1.	0.79	1457.48
2.	0.89	1486.33
3.	0.99	1519.13

It also can be observed that at '*gamma – 0.89*, the time taken to train the agents was NOT the least. Here, there is not too much of difference in the time between the three values. There is also not too much of a defference in the mean rewards for *gamma = 0.79* and *gamma = 0.89*. But 0.99 definitely gives the worst value.

Note: The Gamma value taken were [0.79, 0.89, 0.99]

2. WRITING TASK – Questions to be answered

Question 1 – Explain what happens in reinforcement learning if the agent always chooses the action that maximizes the Q-value. Suggest two ways to force the agent to explore.

Answer: When the agent always chooses the action that maximizes the Q-value, the behavior is said to be greedy. This is also called exploitation, when the agent wants to maximize the reward based only on what it knows, i.e., without further exploration. This prevents the agent from further exploration of non-greedy actions, which might have the possibility to maximize the future reward. Two ways to force the agent to explore are as follows:

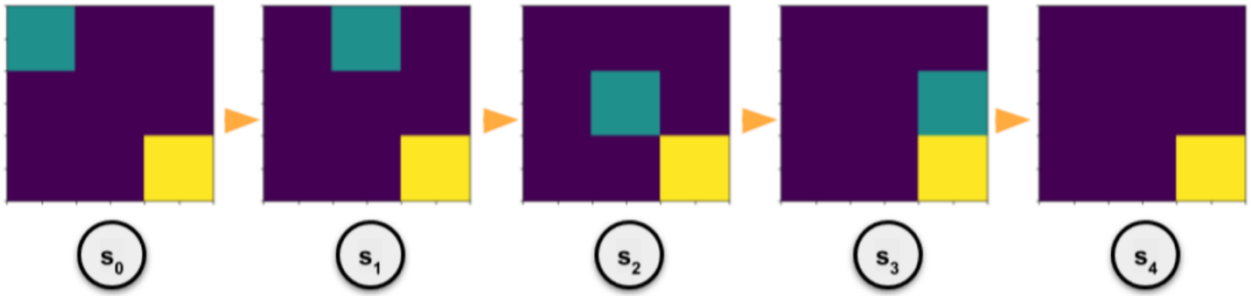
1. *Epsilon greedy (on-policy):* This uses the concept of epsilon greedy policy, in which most of the time they use maximal estimated action value, but with probability epsilon, an action is selected at random. This policy can have different functions (exponential, linear, quadratic, etc), and depending on the problem, the function can be selected. In this project, we have used exponential decay function.
2. *UCB-1*: The UCB-1 strategy keeps a count of the number of times an action a is executed in state *s*, for all actions and all states. After selecting an action, the strategy increases the count of the action taken in a state by one. The UCB-1 strategy first selects all actions exactly one time when a state is visited, so their counters have been populated. Afterwards, UCB-1 calculates an exploration bonus using:

$$bonus(s_t, a^i) = 100 \times C \times \sqrt{2 \times \frac{\log N(s_t)}{N(s_t, a^i)}}$$

where $N(s_t) = \sum_i N(s_t, a^i)$ denotes how often an action has been selected in state s_t, and $N(s_t, a^i)$ counts the number of times action a^i has been selected in state s_t. The bonus becomes larger when the fraction of counts for all actions versus the count for action a^i increases. A larger value of *C* increases exploration.

Question 2 – Calculate Q-value for the given states and provide all the calculation steps.

Answer:



AGENT → ■
GOAL → ■
ACTION TAKEN BY AGENT → RIGHT, DOWN, RIGHT, DOWN
GAMMA → 0.99

Rewards are as follows:
-1 → when it moves away from the goal
0 → when it does not move at all (i.e., tries to move to an edge)
1 → when it moves closer to the goal

Q-function used is as follows:

$$Q(s_t, a_t) = r_t + \gamma * \max_a Q(s_{t+1}, a)$$

As suggested, starting from the last state, **S_4:**
Since the agent has already reached the goal in this state, there will be no further actions to take, and hence the immediate and future rewards will be 0 for all actions.

At S_3:
Case(i) Action: Down

$$Q(s_3, D) = 1 + 0.99 * \max_a Q(s_4, a)$$

$$\Rightarrow \quad Q(s_3, D) = 1 + 0.99 * (0)$$

$$\Rightarrow \quad Q(s_3, D) = 1$$

Case(ii) Action: Left

$$Q(s_3, L) = -1 + 0.99 * \max_a Q(s_2, a)$$

$$\Rightarrow \quad Q(s_3, L) = -1 + 0.99 * (1.99)$$

$$\Rightarrow \quad Q(s_3, L) = 0.97$$

Case(iii) Action: Up

$$Q(s_3, U) = -1 + 0.99 * \max_a Q(s_2, a)$$

$$\Rightarrow \quad Q(s_3, U) = -1 + 0.99 * (1.99)$$

$$\Rightarrow \quad Q(s_3, U) = 0.97$$

Case(iv) Action: Right

$$Q(s_3, R) = 0 + 0.99 * \max_a Q(s_3, a)$$

$$\Rightarrow \quad Q(s_3, R) = 0 + 0.99 * (1)$$

$$\Rightarrow \quad Q(s_3, R) = 0.99$$

At S_2:

Case(i) Action: Down

$$Q(s_2, D) = 1 + 0.99 * \max_a Q(s_3, a)$$

$$\Rightarrow \quad Q(s_2, D) = 1 + 0.99 * (1)$$

$$\Rightarrow \quad Q(s_2, D) = 1.99$$

Case(ii) Action: Left

$$Q(s_2, L) = -1 + 0.99 * \max_a Q(s_1, a)$$

$$\Rightarrow \quad Q(s_2, L) = -1 + 0.99 * (2.97)$$

$$\Rightarrow \quad Q(s_2, L) = 1.94$$

Case(iii) Action: Up

$$Q(s_2, U) = -1 + 0.99 * \max_a Q(s_1, a)$$

$$\Rightarrow \quad Q(s_2, U) = -1 + 0.99 * (2.97)$$

$$\Rightarrow \quad Q(s_2, U) = 1.94$$

Case(iv) Action: Right

$$Q(s_2, R) = 1 + 0.99 * \max_a Q(s_3, a)$$

$$\Rightarrow \quad Q(s_2, R) = 1 + 0.99 * (1)$$

$$\Rightarrow \quad Q(s_2, R) = 1.99$$

At S_1:

Case(i) Action: Down

$$Q(s_1, D) = 1 + 0.99 * \max_a Q(s_2, a)$$

$$\Rightarrow \quad Q(s_1, D) = 1 + 0.99 * (1.99)$$

$$\Rightarrow \quad Q(s_1, D) = 2.97$$

Case(ii) Action: Left

$$Q(s_1, L) = -1 + 0.99 * \max_a Q(s_0, a)$$

$$\Rightarrow \quad Q(s_1, L) = -1 + 0.99 * (3.94)$$

$$\Rightarrow \quad Q(s_1, L) = 2.90$$

Case(iii) Action: Up

$$Q(s_1, U) = 0 + 0.99 * \max_a Q(s_1, a)$$

$$\Rightarrow \quad Q(s_1, U) = 0 + 0.99 * (2.97)$$

$$\Rightarrow \quad Q(s_1, U) = 2.94$$

Case(iv) Action: Right

$$Q(s_1, R) = 1 + 0.99 * \max_a Q(s_2, a)$$

$$\Rightarrow \quad Q(s_1, R) = 1 + 0.99 * (1.99)$$

$$\Rightarrow \quad Q(s_1, R) = 2.97$$

At S_0:

Case(i) Action: Down

$$Q(s_0, D) = 1 + 0.99 * \max_a Q(s_1, a)$$

$$\Rightarrow \quad Q(s_0, D) = 1 + 0.99 * (2.97)$$

$$\Rightarrow \quad Q(s_0, D) = 3.94$$

Case(ii) Action: Left

$$Q(s_0, L) = 0 + 0.99 * \max_a Q(s_0, a)$$

$$\Rightarrow \quad Q(s_0, L) = 0 + 0.99 * (3.94)$$

$$\Rightarrow \quad Q(s_0, L) = 3.90$$

Case(iii) Action: Up

$$Q(s_0, U) = 0 + 0.99 * \max_a Q(s_0, a)$$

$$\Rightarrow \quad Q(s_0, U) = 0 + 0.99 * (3.94)$$

$$\Rightarrow \quad Q(s_0, U) = 3.90$$

Case(iv) Action: Right

$$Q(s_0, R) = 1 + 0.99 * \max_a Q(s_1, a)$$

$$\Rightarrow \quad Q(s_0, R) = 1 + 0.99 * (2.97)$$

$$\Rightarrow \quad Q(s_0, R) = 3.94$$

Final Table of future rewards for this problem:

	ACTION			
STATE	UP	DOWN	LEFT	RIGHT
0	3.90	3.94	3.90	3.94
1	2.94	2.97	2.90	2.97
2	1.94	1.99	1.94	1.99
3	0.97	1	0.97	0.99
4	0	0	0	0

Conclusion:

In this project, we implemented reinforcement learning, and Deep-Q Learning. The hyper-parameters were tuned to evaluate the performance of the model. The mean rewards and agent training time were calculated, evaluated and tested throughout the project.

References:

[1] Tijsma, A. D., Drugan, M. M., & Wiering, M. A. (2016, December). Comparing exploration strategies for q-learning in random stochastic mazes. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on* (pp. 1- 8). IEEE.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M.G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature 518(7540), 529.

[3] Some concepts from Piazza