NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Reinforcement Learning Assignment (AI6101 -Introduction to AI and Ethics)

BY

ONG WEN QING , G2202256K

1. Requirements of this assignment

In this project, we will learn how to implement one of the Reinforcement Learning algorithms (e.g., Q-learning, SARSA, or value iteration, policy iteration) to solve the Box Pushing grid-world game with various difficulty levels. Novel RL ideas are welcome and will receive bonus credit. In this assignment, we need to implement the code on our own and present a convincing presentation to demonstrate the implemented algorithm.

2. Introduction

In this assignment I will be implementing one of the reinforcements learning (RL) algorithms that is taught in class (Q-learning) to implement in solving of the Box-Pushing grid-world game. In this report, I will demonstrate the basic environment and algorithms and included graphic data to show the experiment results.

2.1 Grid-World Environment

The environment is a 2D grid world with 5 rooms as shown in Figure 1. The size of the environment is 6x14. In Figure 1, A indicates the agent, B stands for box, G is the goal and x mean the cliff. Hence, the objective of this assignment is to use one of the RL algorithms and train the agent to push the box to the goal position. The games will end under the following three conditions:

1. The agent or the box steps into the dangerous region (cliff)
2. The current time step attains the maximum time step of the game.
3. The box arrival at the goal position.

Figure 1: Box Pushing Game (Grid World Environment)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0							X			X				
1							X			X				
2				X			X			X			X	
3				X			X						X	
4		B		X						X			X	G
5	A			X						X			X	

2.2 MDP Formation

The MDP formulation is describe as follows:

1. States: The state consist of the position of the agent and the box. In Python, it is a tuple, for example, at time step 0, the state is ((5, 1), (4, 1)) where (5, 1) is the position of the agent and (4, 1) is the position of the box.
2. Observation: The agent has a partial observation of 3x3, and the agent is in the centre. In the code, the algorithm takes the observation as input.
3. Action: The action space is [1,2,3,4], which is corresponding to [up, down, left, right]. The agent needs to select one of them to navigate in the environment.
4. Reward: The reward is calculated based on the agent's movement. The final reward is a summation of three values at each time step. The three values are -1 for each movement, the negative value of the distance between the box and the goal as well as the negative value of the distance between the agent and the box. In addition to that, the agent will also receive a reward of -1000 if the agent or the box falls into the cliff.

   Formally, $r_t = \begin{cases} -1 - distance(box, goal) - distance(box, agent) & \text{if no falling} \\ -1 - distance(box, goal) - distance(box, agent) - 1000 & \text{if falling} \end{cases}$
5. Transition: Agent's action can change its position and the position of the box. If a collision with a boundary happens, the agent or the box would stay in the same position. The transition can be seen in the step () function in line 206-257 of environment.py.

3. Reinforcement Learning algorithms

3.1 Q-Learning

Q-Learning was developed by Watkins, 1989 and was the most important breakthroughs in reinforcement learning. Q-learning is a value-based algorithm [1]. For example, the learned action-value function, Q, is directly approximates q*, which is the optimal action-value function, independent of the policy being followed. This result in the analysis of the algorithm to be simplified since this enable early convergence proofs. This policy also determines which state-action pairs are visited and updated; however, the correct convergence required all the pairs to be continued updated [1] .

In this task, we will initialize the Q-table to [0,0,0,0] before exploration of the environment. The Q-table will record the estimate Q-value of the different actions with distinct states which indicated as (state, action) in the Q-table since Q-learning is a value-based algorithm. When the agent conduct the search in the environment, it will update Q (s,a) with every iteration. Hence, with increasing iteration, the agent will be able to explore the environment more and Q-function will be able to converge, or a certain set number of iteration has been reached. The formulae of the Q-learning as shown below. (Example is taken from the lecture notes):

- Previously:

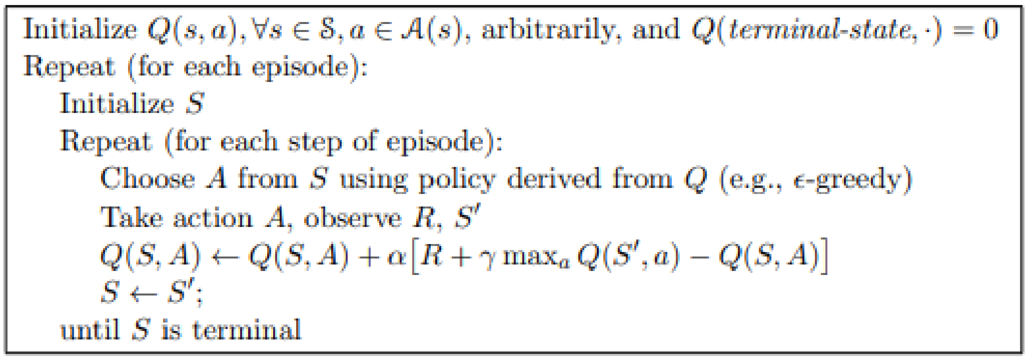
$$\tilde{Q}_\pi(s,a) = \frac{1}{N}\Sigma_{i=1}^{N} G_{i,s}$$

- Now, updating rule:

$$\underbrace{Q_{new}(S_t,A_t)}_{\text{new estimation}} \leftarrow Q_{old}(S_t,A_t) + \underbrace{\alpha}_{\text{learning rate}}(\underbrace{R_{t+1} + \gamma\max_a Q_{old}(S_{t+1},a)}_{\text{new sample}} - \underbrace{Q_{old}(S_t,A_t)}_{\text{old estimation}})$$

Some notion to take note on α is the learning rate; γ is the discount factor; t is the current time step (episodes).

Figure 2: Below Pseudo Code show the Iteration process of the Q-learning. The assignment is built based on this Pseudo Code [2].



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$;
    until $S$ is terminal
```

4. Procedure

We will be using ε-Greedy algorithm to speed up the learning rate as well as randomness in the learning process which can help to explore randomly instead of using action in the Q-table. Table 1 will illustrate the parameter that will be used for the assignment.

Table 1: Training Parameter for this assignment.

Training Parameters	
Learning Rate, α	0.5
Discount factor,γ	0.99
Number of episodes T	5000
ε-greedy ε	0.01

5. Learning Progress

5.1 Episode Rewards vs Episodes

We can notice that Q-learning algorithm able to converge to an optimized path and completing the learning progress in this assignment. The Episode rewards vs episodes curve of the learning progress is shown in Figure 3 while Figure 4 Shows the Episode Length vs Episode. The graph is smoothed 10X due to the density of the data point collected. In addition, we can observe that at the start the learning rate fluctuations/oscillation is very drastic which shows that the agent is trying to find the optimal path to push the box to the goal in the grid work environment. Towards the end of the learning, the agent successfully pushes the box to the desired destination.

Figure 3: Episode Rewards Vs Episode (Smoothed 10x)

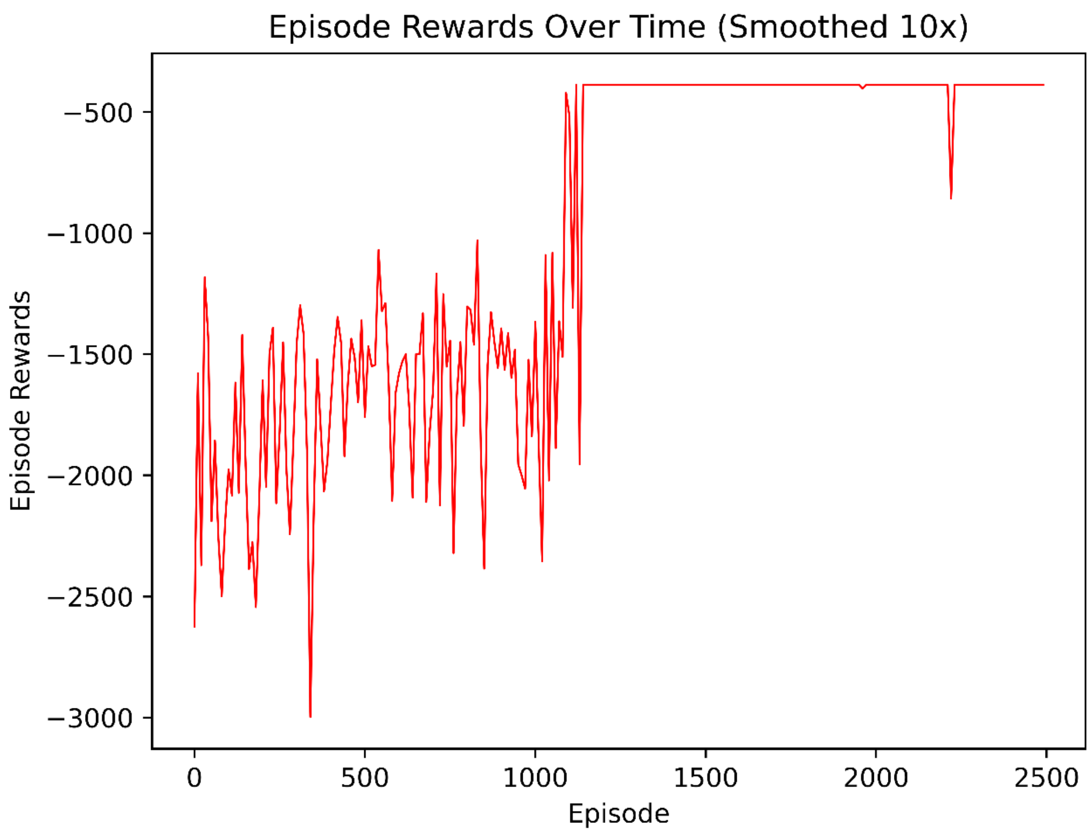
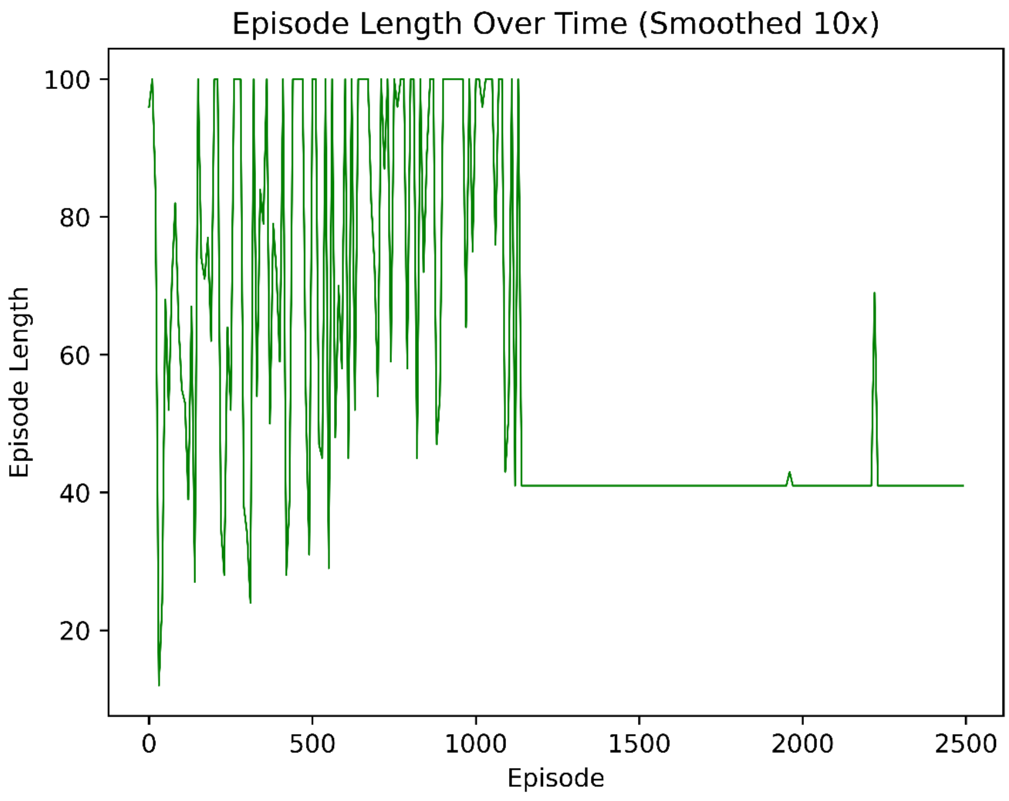


Figure 4: Episode Length Vs Episode (Smoothed 10x)



5.2 Final V-Table

Observation: For this experiment, the rewards is directly proportional to the distance between the box and the goal for each iteration. In addition, the distance between the box and goal are dependent on the 4 actions (up, down, left and right). Hence, the V-table contain only 4 actions (Up, down, left and right) of each block in each state. In the code, the function printgrid(grid) will illustrate the V-table of the Box Pushing game before implementation of Q-learning (Figure 5) and Final V-table of the Box Pushing game after the implementation of Q-learning (Figure 6).

Figure 5: V-table of the Box Pushing Game (Before implementation of Q-learning)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0							X			X				
1							X			X				
2				X			X			X			X	
3				X			X						X	
4		B		X						X			X	G
5	A			X						X			X	

Figure 6: Final V-table of the Box Pushing Game (After implementation of Q-learning)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0					>	V	X			X			>	V
1	>	>	>	>	^	V	X			X	>	>	^	V
2	^	<		X		V	X			X	^	<	X	V
3		^		X	V	<	X	>	>	>	V	^	X	A
4		^		X	>	>	>	^	<	X	>	^	X	B
5	>	^		X				>	^	X			X	

5.3 Discussion (Experiment with learning rate, α from 0.5 to 0.1)

In this experiment, we vary the learning rate from 0.5 to 0.1 to observe the Episode Reward Rate as well as the Path the Agent move in the V-Table.

Table 2: Training Parameter to demonstrate the experiment with learning rate, α =0.1

Training Parameters	
Learning Rate, α	0.5 ->0.1
Discount factor,γ	0.99
Number of episodes T	5000
ε-greedy ε	0.01

Figure 7: Shows the Episode Rewards Vs Episode for the experiment with learning rate, α =0.1

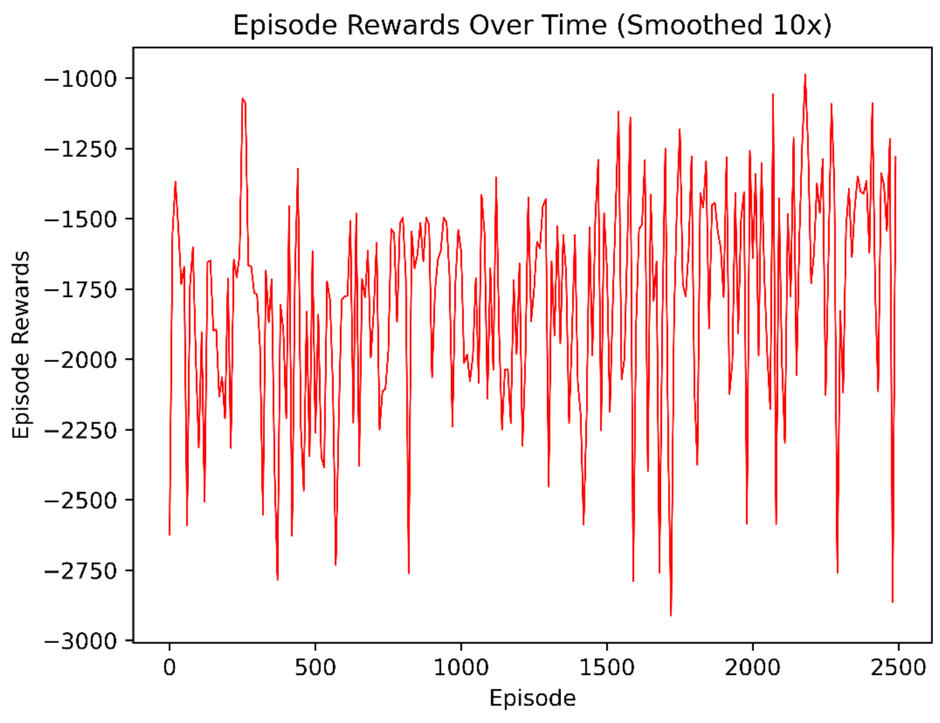


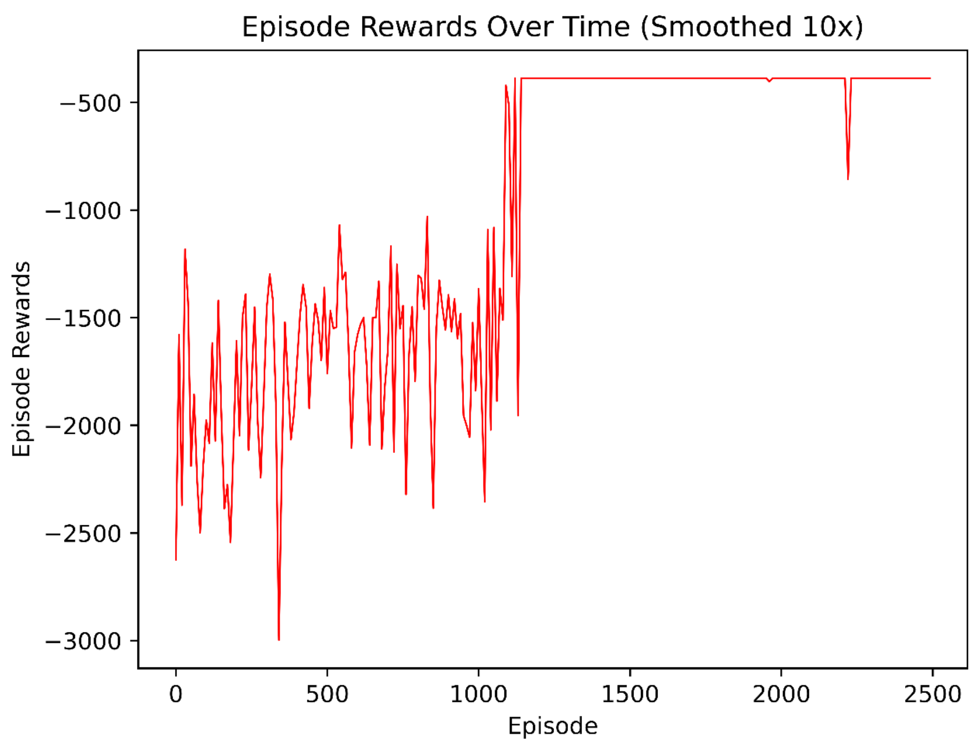
Figure 8: V-table for the Box Pushing Game (experiment with learning rate, α =0.1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0					>	V	X	>	V	X				
1	>	>	>	>	^	V	A	<	V	X	V	>	^	
2	^	<		X		V	X	^	<	X	>	v	X	
3		^		X	V	<	X	>	^	<	<	<	X	
4		^		X	>	>	>	V	^	X	^	B	X	G
5	>	^		X				>	^	X	^	<	X	

Observation: When the learning rate, α change from 0.5 to 0.1, the Q-learning algorithm not able to converge with the same number of iterations which is show in Figure 7, hence from Figure 8, the V-table we can see that the agent not able to push the box to the desired goal and in addition the agent is stuck in a loop until it hit the cliff as highlighted in Figure 8 . Therefore, for this assignment the optimal learning rate is set to be 0.5.

Conclusion: This assignment enables us to be familiar with Reinforcement algorithms such as Q-learning algorithm and understanding the parameters in the Q-learning algorithm, such as the learning rate α, which can affect the rewards given to the agent as shown in the 5.3 Discussion of the report.

Finally, the Episode Rewards vs Episode and Final V-table as shown below:



Final V-table

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0					>	V	X			X			>	V
1	>	>	>	>	^	V	X			X	>	>	^	V
2	^	<		X		V	X			X	^	<	X	V
3		^		X	V	<	X	>	>	>	V	^	X	A
4		^		X	>	>	>	^	<	X	>	^	X	B
5	>	^		X				>	^	X			X	

Reference

[1] R. Sutton, F. Bach and A. Barto, *Reinforcement Learning*. Massachusetts: MIT Press Ltd, 2018.

[2] Teaching Material by Assoc Prof Bo AN