**AY2024/25 SEMESTER 1**

**SC3000 Aritificial Intelligence**

**Reinforcement Learning Assignment**

Tutor: **Bo An**

Lab group: **TEL6**

Date of Submission: **10 October 2024**

Author: **Randy Tan Yu Hong (U2222570E)**

## Table of Contents

# 1. Background Information

This report is part of the Lab Assignment 1 for SC3000 Artificial Intelligence for AY24/25 Semester 1. The project requires us to implement a Reinforcement Learning (RL) Algorithm to solve the **CliffBoxPushing** grid-world game.

Our task is to implement the code for:

1. **The RLAgent class**
2. **Visualization of learned V-table**

# 2. Reinforcement Learning Algorithm

In the course of SC3000, we were taught 3 RL algorithms - Monte Carlo, Q-learning, and Deep Q-Network. Below is a brief analysis of the suitability of each RL algorithm for the assignment:

1. **Monte Carlo (MC):**
   The main idea behind MC is to use randomness to solve a problem - generating suitable random numbers and observing the fraction of numbers obeying some properties. MC methods require episodic tasks and only update values at the end of each episode. In this assignment, waiting until the end of the episode for updates could be inefficient and lead to poor performance during training.

2. **Q-Learning:**
   Q-learning is an example of bootstrapping, leveraging the estimates of future rewards to update the Q-values of the current state-action pair. Q-learning is a good fit for this assignment because it updates the Q-value at each step, allowing the agent to learn to avoid dangerous cliffs faster. It can also handle the reward structure, learning to push the box while avoiding the cliff by continuously updating values during each step.

3. **Depp Q-Network (DQN):**
   DQN uses a neural network the approximate the Q-values, and is generally used for larger or continuous state spaces. Since the grid is relatively small (6x14), the extra power that comes with the complexity of DQN may not be necessary

## Final Selection

I have decided to use Q-learning for this specific problem due to the manageable size of the grid-world and the need for step-by-step updates, which helps the agent learn to avoid cliffs and push the box to the goal efficiently. DQN would introduce unnecessary complexities, while MC might be inefficient due to delayed updates.

The Q-learning formula is as shown below (adapted from Lecture Notes).

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

**Fig.1 Q-learning formula**

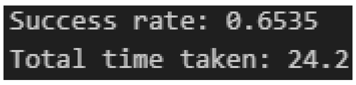
In this assignment, the learning rate (**alpha value**) has been set to 0.1, the discount factor (**gamma value**) has been set to 0.99.

The learning rate determines how quickly the agent learns from new experiences, while the discount factor discounts the future rewards compared to immediate rewards (money is worth more today than tomorrow). **Alpha=0.1** means that only 10% of the new experiences is based on the Q-value, and 90% is based on the old Q-value. **Gamma=0.99** means that the agent places more emphasis on future rewards, considering long-term rewards when making decisions.

Q-learning helps the agent to improve his decision-making by continuously adjusting its Q-values based on the maximum expected reward it can achieve in each given state by following the optimal policy.

## 3. Results and Analysis

### Learning Curve



```
Success rate: 0.6535
Total time taken: 24.2
```

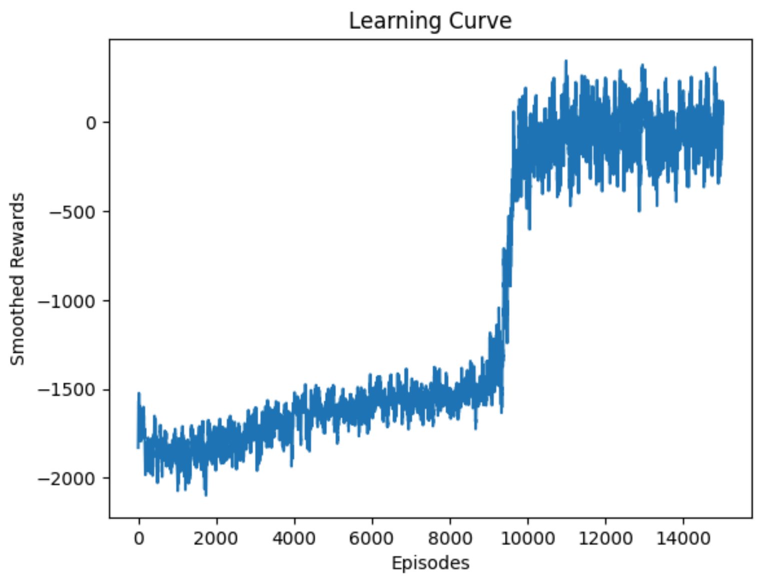
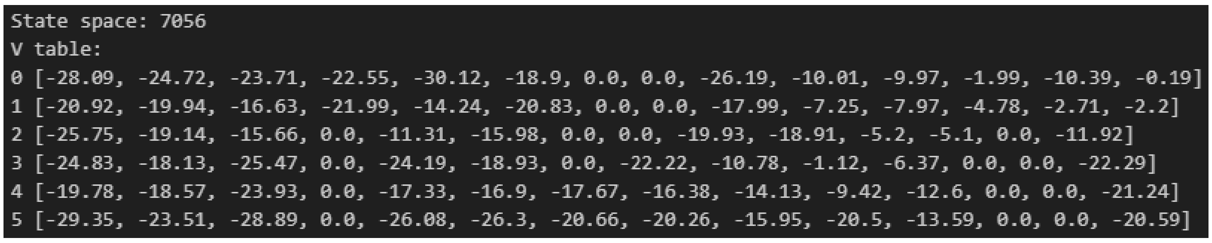


**Fig. 2: Learning Curve of Smoothed Rewards against Episodes**

The model converges at around **10,000 episodes** in a 15,000 episode run. Taking **24.2s** to complete the training. And a success rate of **0.6535** (calculated as number of episodes where final_rewards > 0 / total number of episodes)

## V-table



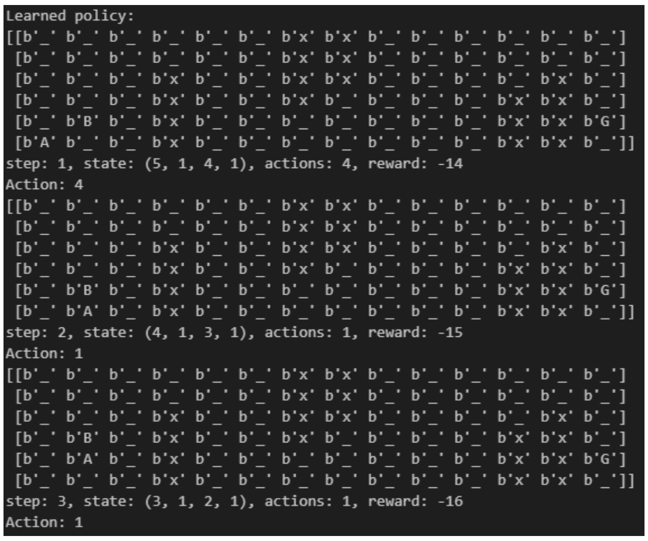
```
State space: 7056
V table:
0 [-28.09, -24.72, -23.71, -22.55, -30.12, -18.9, 0.0, 0.0, -26.19, -10.01, -9.97, -1.99, -10.39, -0.19]
1 [-20.92, -19.94, -16.63, -21.99, -14.24, -20.83, 0.0, 0.0, -17.99, -7.25, -7.97, -4.78, -2.71, -2.2]
2 [-25.75, -19.14, -15.66, 0.0, -11.31, -15.98, 0.0, 0.0, -19.93, -18.91, -5.2, -5.1, 0.0, -11.92]
3 [-24.83, -18.13, -25.47, 0.0, -24.19, -18.93, 0.0, -22.22, -10.78, -1.12, -6.37, 0.0, 0.0, -22.29]
4 [-19.78, -18.57, -23.93, 0.0, -17.33, -16.9, -17.67, -16.38, -14.13, -9.42, -12.6, 0.0, 0.0, -21.24]
5 [-29.35, -23.51, -28.89, 0.0, -26.08, -26.3, -20.66, -20.26, -15.95, -20.5, -13.59, 0.0, 0.0, -20.59]
```

**Fig. 3: State space and V-table**

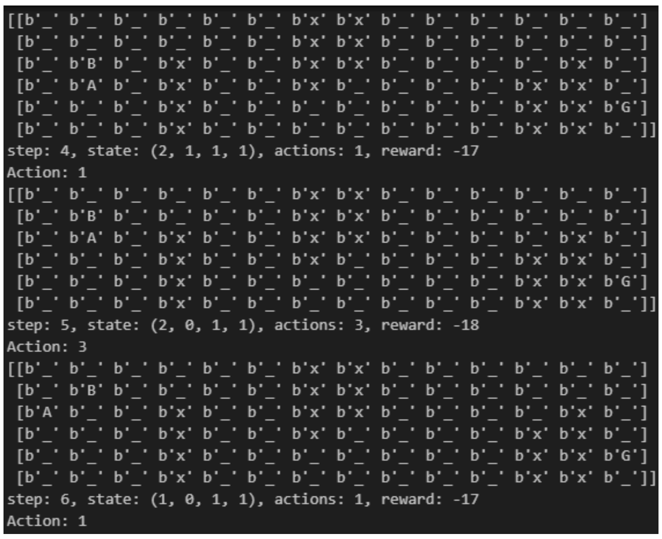
There are **7,056** possible states the agent can be in. The updated V-table containing the average Q-values of each agent position is shown above as well.

## Policy

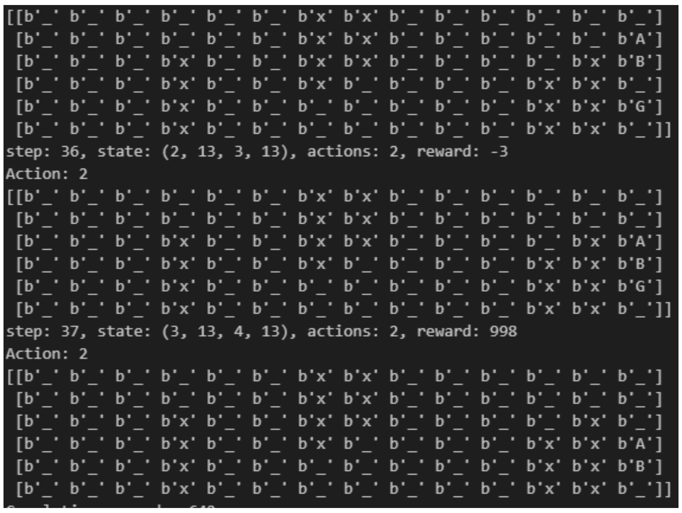
Due to space constraints, only the first 6 steps and last 2 steps with end goal state are shown.A total of **37 steps** were taken by the agent. A cumulative reward of **642**. The action history and optimal state transition are shown below as well.



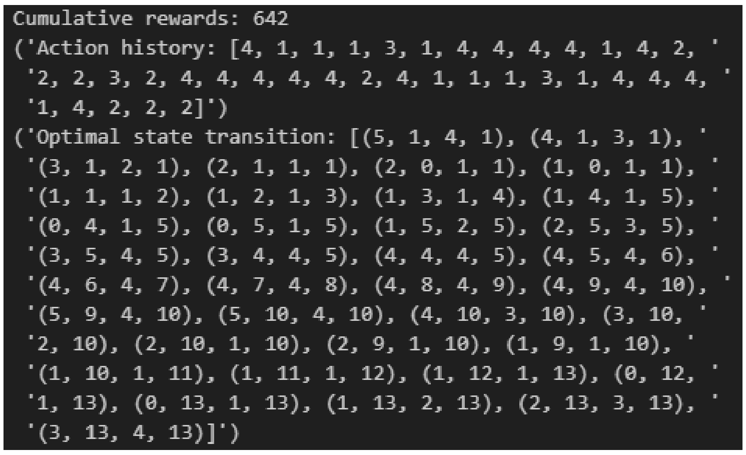
```
Learned policy:
[[b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'x' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'x' b'x' b'_']
 [b'_' b'B' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'G']
 [b'A' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_']]
step: 1, state: (5, 1, 4, 1), actions: 4, reward: -14
Action: 4
[[b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'x' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'x' b'x' b'_']
 [b'_' b'B' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'G']
 [b'_' b'A' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_']]
step: 2, state: (4, 1, 3, 1), actions: 1, reward: -15
Action: 1
[[b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'x' b'_']
 [b'_' b'B' b'_' b'x' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'x' b'x' b'_']
 [b'_' b'A' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'G']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_']]
step: 3, state: (3, 1, 2, 1), actions: 1, reward: -16
Action: 1
```



```
[[b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'B' b'_' b'x' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'x' b'_']
 [b'_' b'A' b'_' b'x' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'x' b'x' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'G']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_']]
step: 4, state: (2, 1, 1, 1), actions: 1, reward: -17
Action: 1
[[b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'B' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'A' b'_' b'x' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'x' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'x' b'x' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'G']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_']]
step: 5, state: (2, 0, 1, 1), actions: 3, reward: -18
Action: 3
[[b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'B' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'A' b'_' b'_' b'x' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'x' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'x' b'x' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'G']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_']]
step: 6, state: (1, 0, 1, 1), actions: 1, reward: -17
Action: 1
```



```
[[b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'A']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'x' b'B']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'x' b'x' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'G']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_']]
step: 36, state: (2, 13, 3, 13), actions: 2, reward: -3
Action: 2
[[b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'x' b'A']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'x' b'x' b'B']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'G']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_']]
step: 37, state: (3, 13, 4, 13), actions: 2, reward: 998
Action: 2
[[b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'_' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'x' b'_' b'_' b'_' b'_' b'x' b'_']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'x' b'x' b'A']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'B']
 [b'_' b'_' b'_' b'x' b'_' b'_' b'_' b'_' b'_' b'_' b'_' b'x' b'x' b'_']]
```



```
Cumulative rewards: 642
('Action history: [4, 1, 1, 1, 3, 1, 4, 4, 4, 4, 1, 4, 2, '
 '2, 2, 3, 2, 4, 4, 4, 4, 4, 2, 4, 1, 1, 1, 3, 1, 4, 4, 4, '
 '1, 4, 2, 2, 2]')
('Optimal state transition: [(5, 1, 4, 1), (4, 1, 3, 1), '
 '(3, 1, 2, 1), (2, 1, 1, 1), (2, 0, 1, 1), (1, 0, 1, 1), '
 '(1, 1, 1, 2), (1, 2, 1, 3), (1, 3, 1, 4), (1, 4, 1, 5), '
 '(0, 4, 1, 5), (0, 5, 1, 5), (1, 5, 2, 5), (2, 5, 3, 5), '
 '(3, 5, 4, 5), (3, 4, 4, 5), (4, 4, 4, 5), (4, 5, 4, 6), '
 '(4, 6, 4, 7), (4, 7, 4, 8), (4, 8, 4, 9), (4, 9, 4, 10), '
 '(5, 9, 4, 10), (5, 10, 4, 10), (4, 10, 3, 10), (3, 10, '
 '2, 10), (2, 10, 1, 10), (2, 9, 1, 10), (1, 9, 1, 10), '
 '(1, 10, 1, 11), (1, 11, 1, 12), (1, 12, 1, 13), (0, 12, '
 '1, 13), (0, 13, 1, 13), (1, 13, 2, 13), (2, 13, 3, 13), '
 '(3, 13, 4, 13)]')
```

## 4. Learning Outcomes

Throughout this Lab assignment project, I learnt how to implement the Q-Learning algorithm taught during lectures to train an agent. Updating the Q-values after each step taken by the agent, and finally to visualize the V-table and reviewing the learned policy to verify the solution.

## 5. References

I referred to various lecture notes and online materials throughout the course of this project. As stated below:

- AY24/25S1 SC3000 Lecture Notes for Module 4_Markov Decision Prorcess and Module 5_Reinforcement Learning
- basu, Priyam. "RL Part 5- Implementing an Iterable Q-Table in Python." Medium, May 24, 2020. https://medium.com/iecse-hashtag/rl-part-5-implementing-an-iterable-q-table-in-python-a9b515c2c1a