Reinforcement Learning Assignment

Instructor: Bo An, Professor, SCSE, NTU

Teaching Assistant:
Zheng Longtao, longtao001@e.ntu.edu.sg

Plagiarism Policy: Your solution must be the result of your own individual effort. While you are allowed to discuss problems with your classmates, but you must not blatantly copy others' solutions.

1. Requirements

This project requires you to implement and evaluate one of the Reinforcement Learning (RL) algorithms (e.g., Q-learning, SARSA, etc.) to solve the **CliffBoxPushing** grid-world game. Novel ideas are welcome and will receive bonus credit. In this assignment, you need to implement the code on your own and present a convincing presentation to demonstrate the implemented algorithm.

The following links can help you to get to know more about current RL algorithms:

- OpenAI Spinning Up: https://spinningup.openai.com/en/latest/index.html

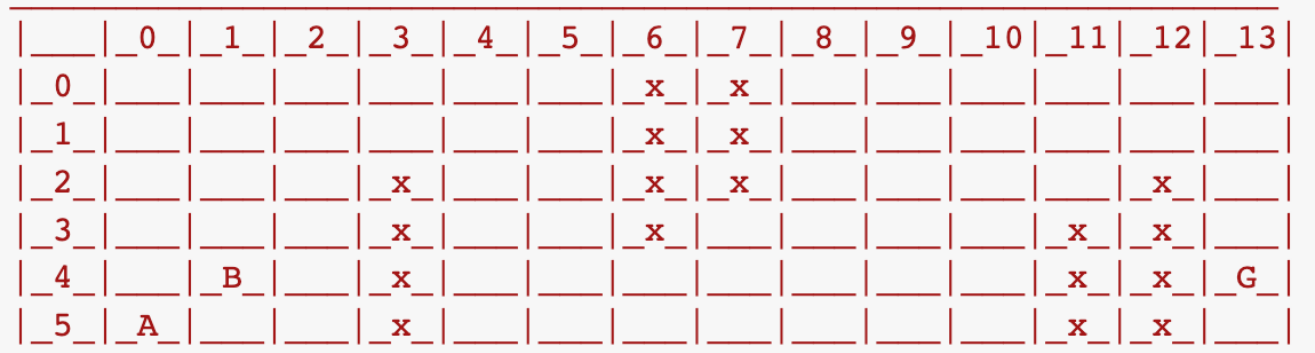

Figure 1. The Cliff Box Pushing Grid World.

2. The Environment

The environment is a 2D grid world as shown in Fig. 1. The size of the environment is 6×14. In Fig. 1, **A** indicates the agent, **B** stands for the box, **G** is the goal, and **x** means cliff. You need to write code to implement one of the RL algorithms and train the agent to push the box to the goal position. The game ends under 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 arrives at the goal.

The MDP formulation is described as follows:

- **State:** The state consists 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, 0), (4, 1))` where `(5, 0)` is the position of the agent and `(4, 1)` is the position of the box.
- **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.
- **Reward:** The reward consists of
  1. the agent will receive a reward of -1 at each timestep
  2. the negative value of the distance between the box and the goal
  3. the negative value of the distance between the agent and the box
  4. the agent will receive a reward of -1000 if the agent or the box falls into the cliff.
  5. the agent will receive a reward of 1000 if the box reaches the goal position.
- **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 of the environment.

You can check the code for further details.

3. Complete the Code

We provide the code template **RLAgent.ipynb**, which includes the implementation of the BoxPushing environment. You can visualize the grid world and manually control your agent to push the box, as well as training agent within the Jupyter Notebook (or Google Colab). We provide a random agent and an example agent class in the code for you to add your own code to train an RL agent and compare their performance. The reset() of environment initializes the positions of the agent and the box.

You need to complete the code of (marked with TODO)

1. The RLAgent class.
2. Visualization of learned V-table (the average Q-values of each agent position, hint: consider all the box position and calculate average values) and policy.

There is no limit on the number of iterations. You can also do reward engineering or change other part of the code, but do not modify the game setting (i.e., the map is fixed, and grading will be based on the provided map). If you do this, please clarify your modification in the report, and provide the reasons by ablation experiments (e.g., better sample efficiency).

Here is one example of visualization of learned V-table and policy (this example is not correct).

```
V table:
0 [-4.71, -3.99, -2.43, -2.72, -2.99, -4.04, -2.87, -2.25, -1.18, -1.1]
1 [-2.64, -3.69, -2.86, -2.19, -2.75, -4.4, -3.54, -2.63, -1.57, -0.18]
2 [-0.58, -1.78, -1.88, -2.15, -2.68, -3.41, -2.48, -1.89, -1.43, 40.62]
3 [0.0, 0.0, -0.43, -2.0, -2.14, -2.69, 1.3, 72.66, 101.76, 274.03]
4 [-0.48, 0.0, -2.28, -2.29, -2.44, -1.81, 46.09, 202.0, 403.47, 769.05]
5 [-1.9, -0.38, -1.0, -1.19, -1.38, -1.58, 10.59, 163.12, 633.52, 870.23]
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'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'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
```

Note: The policy visualization ought to follow a step-by-step approach, while the V-table can be meaningless. To verify your solution, it is recommended to review the learned policy.

4. Deliverables

You can use the code to get the state and finish your own RL code as well as the report.

Submission due: 11:59pm October 22 (Sunday)

Please submit through NTULearn Website.

Submission files:

- The ipynb file that includes your code implementation. You need to add text block or comments to your implementation and visualize the learning progress: episode rewards vs. episodes, and the final V-table and policy.
- A pdf file that includes your report, which contains the description of your chosen algorithm and the plots such as learning progress, V-table, policy, etc. from the ipynb file.
- Your final submission should be compressed into a zip file.

The report filename format:

- AI6101_Report_YourName_YourMatriculationNumber.pdf

The code filename format:

- AI6101_Code_YourName_YourMatriculationNumber.ipynb

Please put the two files into a zip file with the filename format:

- AI6101_Project_YourName_YourMatriculationNumber.zip

5. Marking Criteria

The grading criterion are as follows (total 100 marks):

Item	Marks
Bug-free: correctly implement the code of your chosen RL algorithms and visualization	50%
Plot the learning progress: episode rewards vs. episodes	25%
Final V-table (shown in the gird) and the **policy**.	25%

A bonus of **10** mark is awarded for each of the following task:

- Compare different exploration techniques (such as UCB) with the default epsilon-greedy and non-exploration strategy. Come up with your analysis.
- use only sparse reward (the agent only receives a reward of 1000 if the box reaches the goal position) and implement your ideas to solve this problem (hint: exploration technique, hierarchical RL, etc.).

Nevertheless, your final project mark will still be capped at 100% even if the total mark plus the bonus mark exceeds 100%. You can refer to option-critic as your HRL solution.