Reinforcement Learning Assignment

Instructor: Bo An, Professor, SCSE, NTU

Teaching Assistant:
Zheng Longtao, longtao001@e.ntu.edu.sg
Rundong Wang, rundong001@e.ntu.edu.sg

1. Requirements

In this project, you will learn how to implement one of the Reinforcement Learning algorithms (e.g., Q-learning, SARSA, or value iteration, policy iteration) to solve the **BoxPushing** grid-world game with various difficulty levels. Novel RL 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
- OpenAI Baselines: https://github.com/openai/baselines
- Google Dopamine: https://github.com/google/dopamine

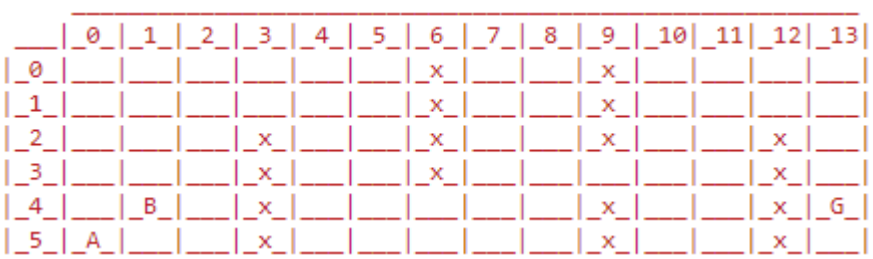

Figure 1. The BoxPushing Game.

2. The Environment

The environment is a 2D grid world with 5 rooms 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. Generally, we 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, 1), (4, 1))` where `(5, 1)` is the position of the agent and `(4, 1)` is the position of the box.
- **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.
- **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 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, $r_t = \begin{cases} -1 - distance(box, goal) - distance(box, agent) & \text{if no falling} \\ -1 - distance(box, goal) - distance(box, agent) - 1000 & \text{if falling} \end{cases}$

- **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. Code Example

We provide the environment code **environment.py**, two example codes **test.py** and **Q-Learning.py**. The **environment.py** includes the implementation of the BoxPushing environment. In the **test.py**, we provide a game interface so that you can manually control your agent to push the box. In the **Q-Learning.py**, we provide the Q-learning framework for you to add your own code to train a Q-learning agent.

Hint: you can run **test.py** to estimate the optimal accumulated reward.

4. Deliverables

We have provided the environment and example code. 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 16 (Sunday)

Please submit through NTULearn Website.

Submission files:

- A report of your project, which contains the description of your chosen algorithm, the learning progress: episode rewards vs. episodes, and the final value table or Q-table.
- Your final code implementation should be compressed into a zip file.

The report filename format:

- AI6101_Report_YourName_YourMatriculationNumber.pdf

The code zip filename format:

- AI6101_Code_YourName_YourMatriculationNumber.zip

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	50%
Show or plot the learning progress: episode rewards vs. episodes	25%
Show the **correct final V table** (show the values in a 6×14 gird) and its corresponding policy (show the arrows in a 6×14 grid).	25%

A bonus of **10** mark may be awarded if you set **reward_box_goal_distance=False** in the environment initialization and use hierarchical reinforcement learning (HRL) to solve this problem. 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.

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.