112-1 (Fall 2023) Semester

Reinforcement Learning

Assignment #2-2

TA: Guan-Ting Liu (劉冠廷)

Department of Electrical Engineering
National Taiwan University

Outline

- Tasks
  - MC: Every-Visit Monte-Carlo Evaluation + ϵ-Greedy Improvement
  - SARSA: Temporal-Difference Prediction TD(0) +ϵ-Greedy Improvement
  - Q-Learning +ϵ-Greedy Improvement
- Environment
- Code structure
- Grading
- Submission
- Policy
- Contact

Tasks

Task 1 – Every-Visit Monte-Carlo Prediction with ϵ-Greedy Improvement

- Problem
  - Evaluate a policy by predicting the Q value function for **each** (state, action) pair
  - Update the Q value function with the **Every-Visit** Monte-Carlo method using **constant $\boldsymbol{\alpha}$**
  - Using collected **trajectories** to update the value function **per episode**

MC Policy Evaluation + ϵ-Greedy Improvement

- Sample kth episode using π: $\{S_1, A_1, R_2, \dots, S_T\} \sim \pi$
- For each state S_t and action A_t in the episode,

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \boxed{(G_t - Q(S_t, A_t))}$$

- Improve policy based on new action-value function

$$\epsilon \leftarrow \text{constant}$$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Estimation Loss

Task 1 – Every-Visit Monte-Carlo Prediction with ϵ-Greedy Improvement

- Problem
  - Evaluate a policy by predicting the Q value function for **each** (state, action) pair
  - Update the Q value function with the **Every-Visit** Monte-Carlo method using **constant $\boldsymbol{\alpha}$**
  - Using collected **trajectories** to update the value function **per episode**

ϵ-Greedy Improvement

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Task 2 - SARSA: Temporal-difference Prediction TD(0) with ϵ-Greedy Improvement

- Problem
  - Evaluate a policy by predicting the Q value function for each (state, action)
  - Update the Q value function with TD(0) method using the collected transition per step

TD(0) Policy Evaluation + ϵ-Greedy Improvement

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
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha \underbrace{\boxed{\left[R + \gamma Q(S',A') - Q(S,A)\right]}}_{\text{Estimation Loss}}$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Task 3 - Q-Learning with ϵ-Greedy Improvement

- Problem
  - Store the collected transition $(S, A, R, S', done)$ in the replay buffer at each time step
  - Uniformly random sample transitions from the replay buffer and store them in the batch
  - Update the Q value function method using the sampled transitions in the batch

Q-Learning + ϵ-Greedy Improvement

Given update frequency m and sample batch size n
Initialize transition count $i = 0$, value function $Q(S, A)$, replay buffer ψ
Repeat (for each episode)
    Initialize S
    Repeat (for each step of the episode):
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe $R, S', done$
        Store the transition $(S, A, R, S', done)$ to ψ
        $i = i + 1$
        Initialize sampled batch transition $B = [\,]$
        If i mod $m == 0$:
            Uniformly random sample n transitions from ψ and store them to B
        For each $(S, A, R, S', done)$ in B:
            $Q(S,A) \leftarrow Q(S,A) + \alpha \underbrace{\boxed{[R + \gamma \max_{A'} Q(S', A') - Q(S,A)]}}_{\text{Estimation Loss}}$
        $S \leftarrow S'$
        Until S is terminal

Environment

Grid World

State space

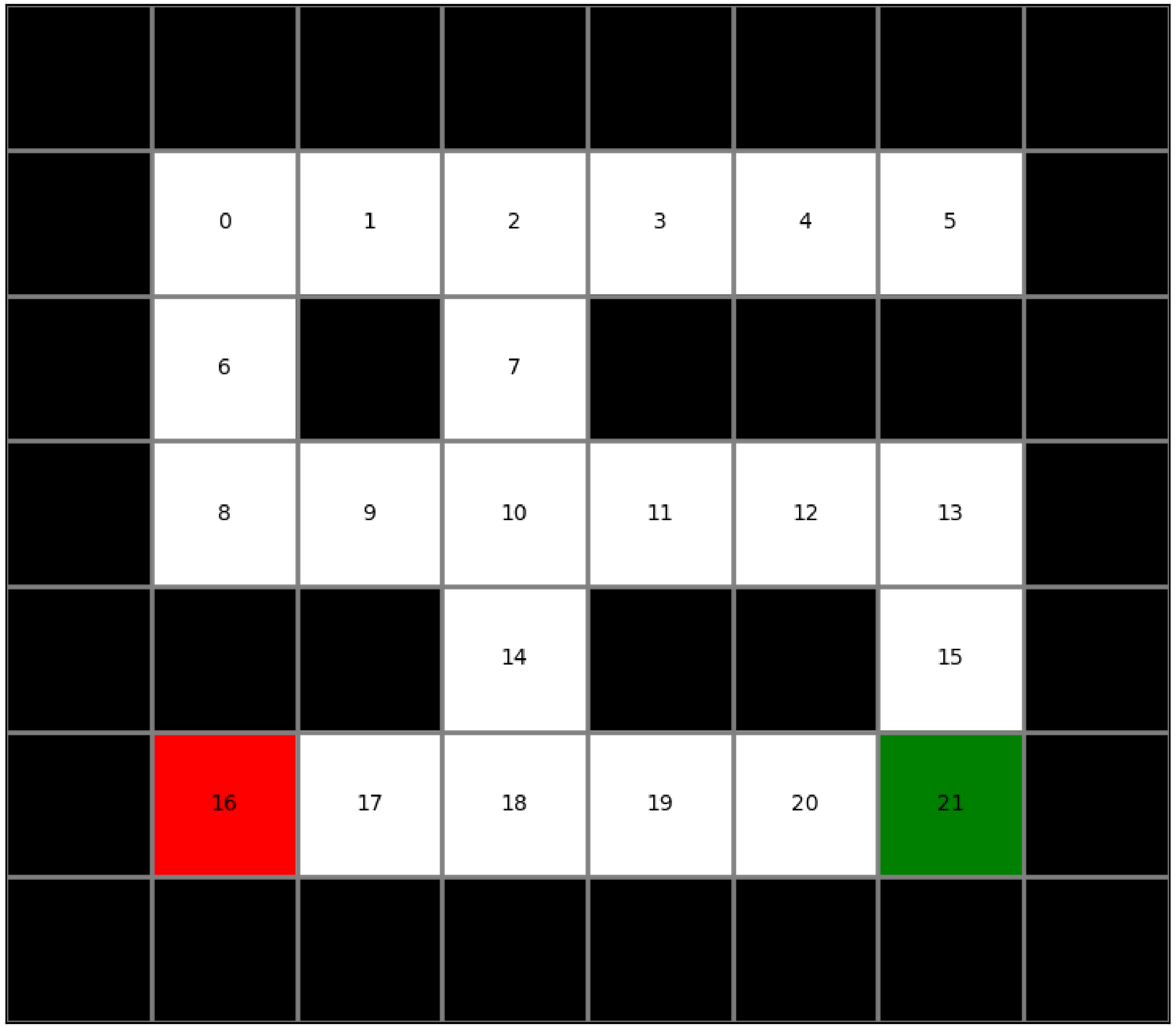
- Nonterminal states: Empty, Wall
- Terminal states: Goal, Trap
- 0-indexed

Action space

- Up, down, left, right
- Hitting the wall will remain at the same state

Reward

- Step reward given at every transition
- Goal reward given after reaching goal state
- Trap reward given after reaching trap state



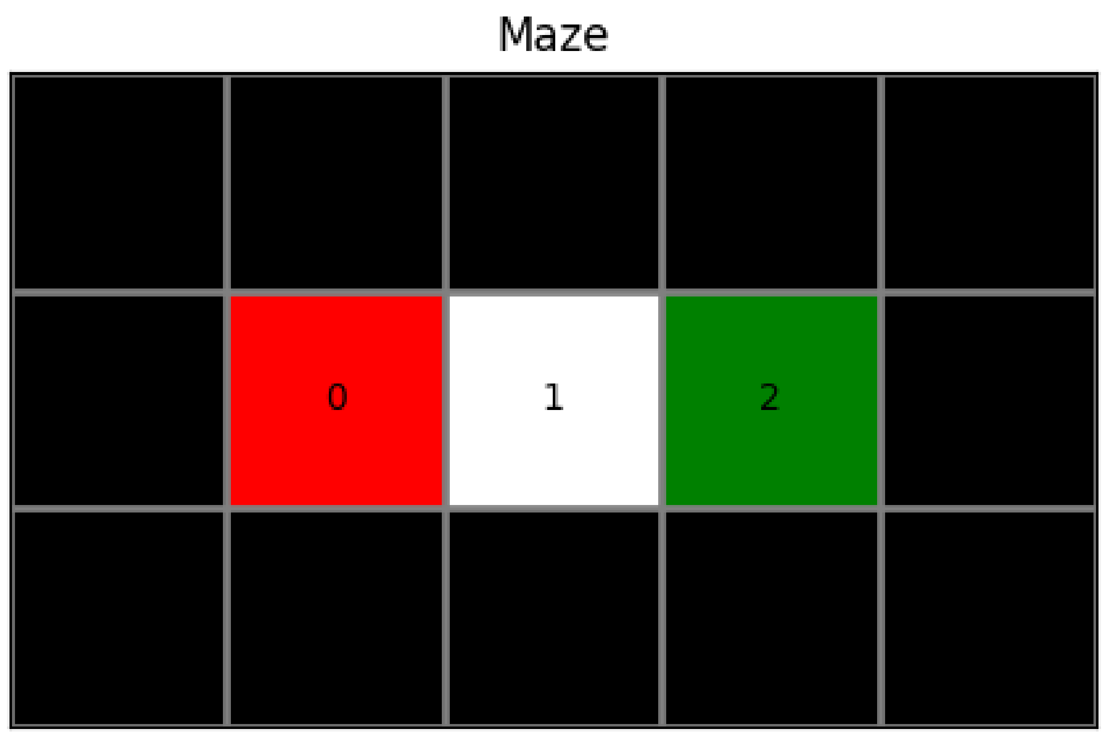
Interaction with Environments

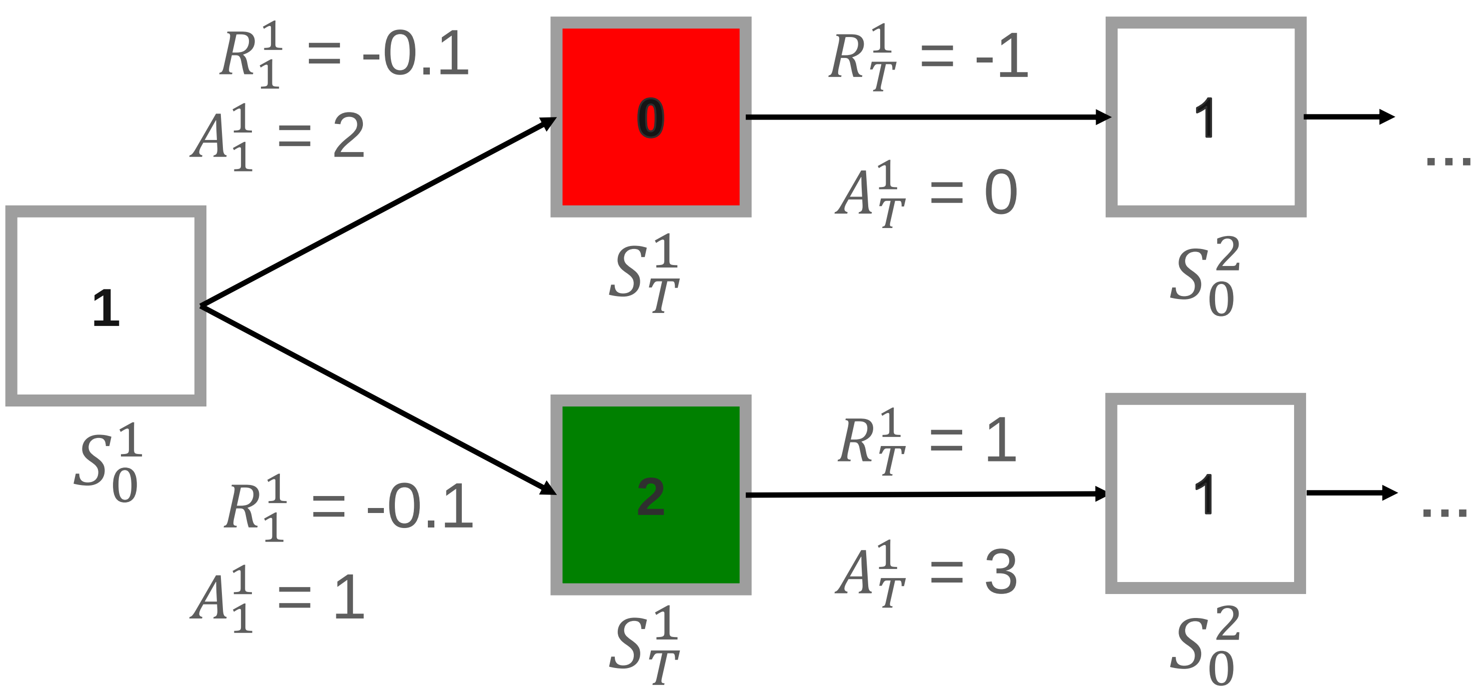
- Learn to interact with a OpenAI gym-like environment
- Grid World in this assignment is a MDP (defined by gridworld.py)
- Grid World functions:
  - step(): Interact with the environment, taking one parameter (action) (in HW 2.1, no parameter is needed to call step())
  - reset(): Reset the environment to the initial state (only used once at the first step)
  - Update the values function with states, rewards and done flags

```
def run(self, max_episode=1000) -> None:
    """Run the algorithm until convergence."""
    # TODO: Implement the Q_Learning algorithm
    iter_episode = 0
    current_state = self.grid_world.reset()
    prev_s = None
    prev_a = None
    prev_r = None
    is_done = False
    transition_count = 0
    while iter_episode < max_episode:
        # TODO: write your code here
        # hint: self.grid_world.reset() is NOT needed here

        raise NotImplementedError
```

Terminal State





- The value of the Goal and Trap state is considered in this assignment
- The final transition at the end of the i^{th} episode will be:

  $$(S_T^i, A_T^i, R_T^i, S_0^{i+1})$$

  where the first state of the $(i+1)^{st}$ episode is S_0^{i+1}

Code Structure

DP_solver_2_2.py

class **DynamicProgramming**
- Parent class for DP algorithms

class **MonteCarloPolicyIteration**
- TODO: run(), policy_evaluation(), policy_improvement()
- The implementation will be judged by run()
- policy_evaluation(), and policy_improvement() are just auxiliary functions, which will not be directly called during judgement

class SARSA
- TODO: run(), policy_eval_improve()
- The implementation will be judged by run().
- policy_eval_improve() is the auxiliary function, which will not be directly called during judgment.

class Q_learning
- TODO: run(), add_buffer(), sample_batch(), policy_eval_improve()
- The implementation will be judged by run().
- Add_buffer(), sample_batch(), and policy_eval_improve() are auxiliary functions, which will not be directly called during judgment.

Feel free to add any function if needed

Grading

Grading

- MC: Running-Average Monte-Carlo Evaluation + ϵ-Greedy Improvement (15%)
  - Hyperparameters are the same as those in main.py
  - Test cases (3% x 5 cases)
- SARSA: Temporal-difference Prediction TD(0) + ϵ-Greedy Improvement (15%)
  - Hyperparameters are the same as those in main.py
  - Test cases (3% x 5 cases)
- Q-Learning + ϵ-Greedy Improvement (15%)
  - Hyperparameters are the same as those in main.py
  - Test cases (3% x 5 cases)
- Report (25%) (Report Template: https://www.overleaf.com/read/vczdnjcvmkbh)
  - Discuss and plot learning curves under ϵ values of (0.1, 0.2, 0.3, 0.4) on MC, SARSA, and Q-Learning (4%)

    Learning curves: #episode (X-axis) vs. Average Non-Discounted Episodic Reward of last 10 episodes (Y-axis)
  - Discuss and plot loss curves under ϵ values of (0.1, 0.2, 0.3, 0.4) on MC, SARSA, and Q-Learning (4%)

    Loss curves: #episode (X-axis) vs. Average Absolute Estimation Loss over each transition of the last 10 episodes (Y-axis)
  - Discuss (and plot) different settings in terms of step reward, discount factor, learning rate, buffer size, update_frequency, and sample batch size (16%)
  - Using Weights & Bias (https://wandb.ai/site) to plot all figures in the report (1%)

Hyperparameters in main.py

```
STEP_REWARD       = -0.1
GOAL_REWARD       = 1.0
TRAP_REWARD       = -1.0
DISCOUNT_FACTOR   = 0.99
LEARNING_RATE     = 0.01
EPSILON           = 0.2
BUFFER_SIZE       = 10000
UPDATE_FREQUENCY  = 200
SAMPLE_BATCH_SIZE = 500
```

Learning Curves & Loss Curves

1. Learning curves: #episode (X-axis) vs. Average non-discounted Episodic Reward $\mathcal{R}$ of last 10 episodes (Y-axis)

   Example:

   Episode 0: step1: r_{01}, step2: r_{02}, …, step T: r_{0a}
   Episode 1: step1: r_{11}, step2: r_{12}, …, step T: r_{1b}
   ⋮
   Episode 9: step1: r_{91}, step2: r_{12}, …, step T: r_{9j}

$$\mathcal{R} = \frac{\frac{\sum_{k=1}^{a} r_{0k}}{a} + \frac{\sum_{k=1}^{b} r_{1k}}{b} + \ldots + \frac{\sum_{k=1}^{j} r_{9k}}{j}}{10}$$

2. Loss Curves: #episode (X-axis) vs. Average Absolute Estimation Loss $\mathcal{L}$ over each transition of the last 10 episodes (Y-axis)

   Example:

   Episode 0: $\text{abs}(EL_{01})$, $\text{abs}(EL_{02})$, …, $\text{abs}(EL_{0a})$
   Episode 1: $\text{abs}(EL_{11})$, $\text{abs}(EL_{12})$, …, $\text{abs}(EL_{1b})$
   ⋮
   Episode 9: $\text{abs}(EL_{91})$, $\text{abs}(EL_{92})$, …, $\text{abs}(EL_{9j})$

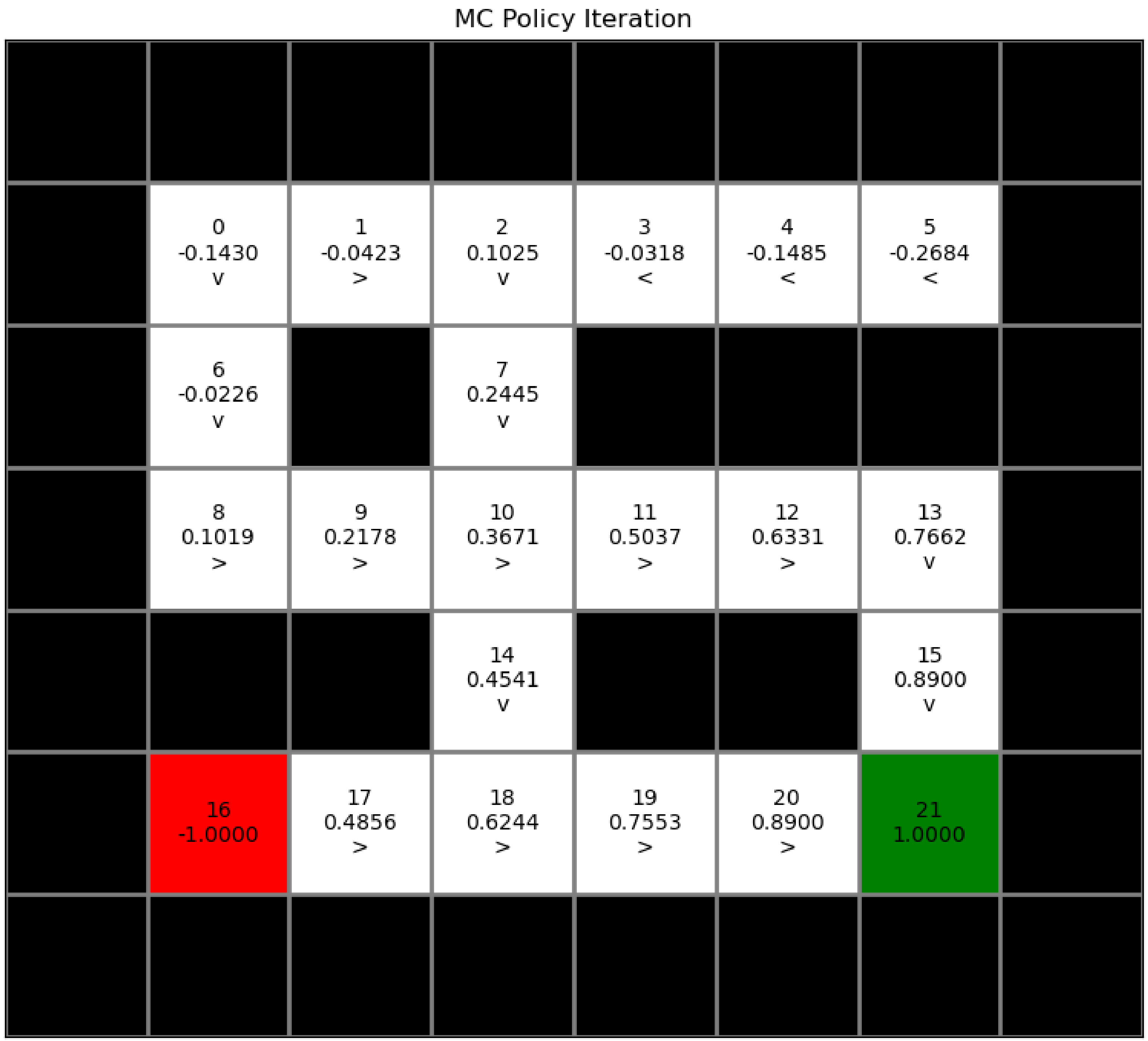
$$\mathcal{L} = \frac{\frac{\sum_{k=1}^{a} \text{abs}(EL_{0k})}{a} + \frac{\sum_{k=1}^{b} \text{abs}(EL_{1k})}{b} + \ldots + \frac{\sum_{k=1}^{j} \text{abs}(EL_{9k})}{j}}{10}$$

Criteria

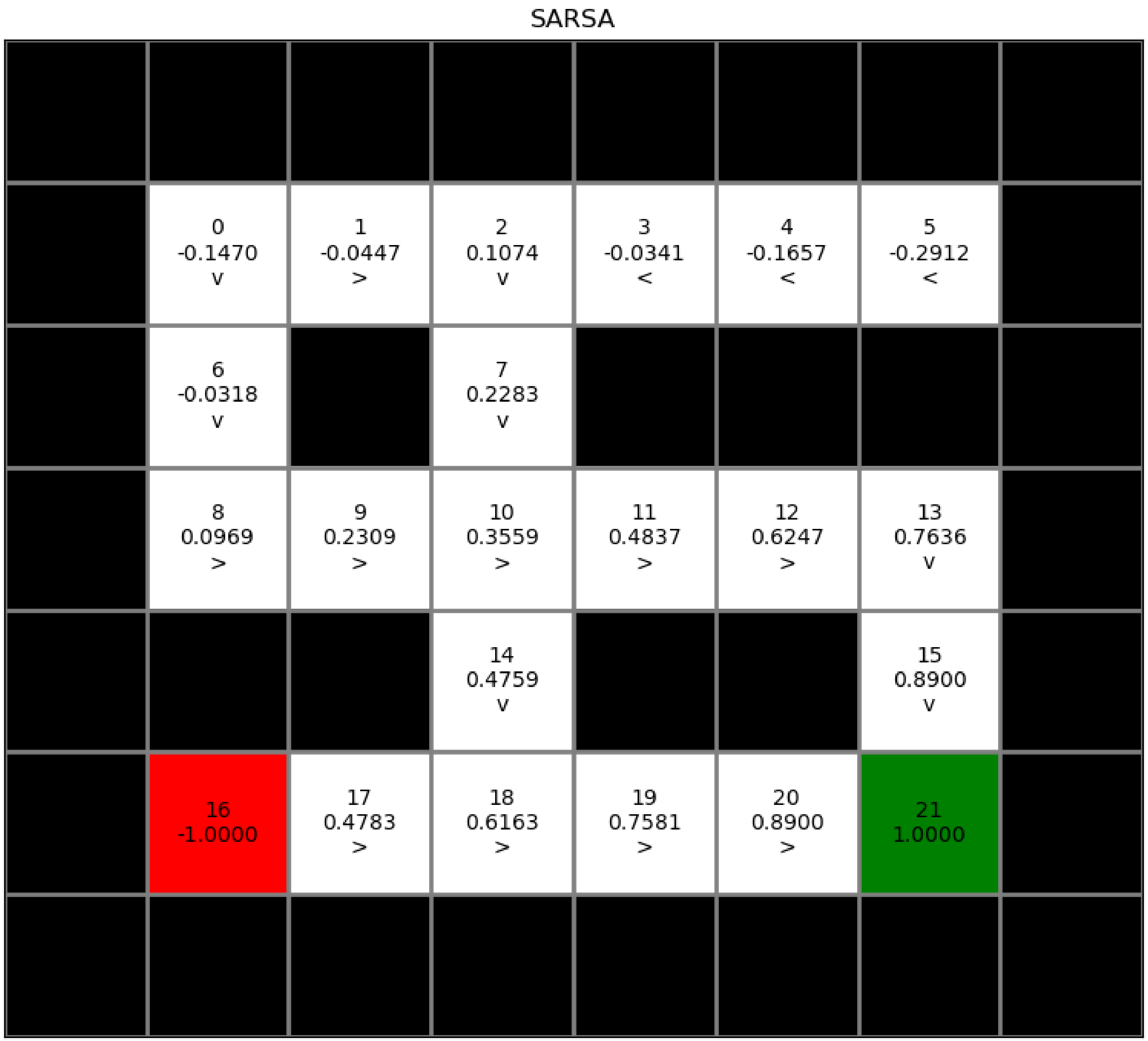
- Test cases:
  - Call run() and check the final output
  - Task 1, 2: Check the resulting *max_state_values* on every state (tolerance: 0.1)
  - Task 3: Check the resulting *max_state_values* on every state (tolerance: 0.001)
  - Tolerance: abs(estimated – *result_from_TAs*)
  - Programs implemented in Task 1, 2 will be reviewed by TAs
  - Testing hyperparameters are listed in main.py
  - The implementation of *max_state_values* can be found in DP_solver.py
  - Run time limit of **5 minutes** for each case to avoid infinite loops
  - Up to 512000 episodes will be conducted in Task1 and Task 2 on private test cases
  - Up to 50000 episodes will be conducted in Task3 on private test cases

- Sample solutions are provided for reference
  - Optimal policy may not be unique
  - Policy in sample solutions are obtained after running 512000 episodes on Task 1,2 and 50000 episode on Task 3

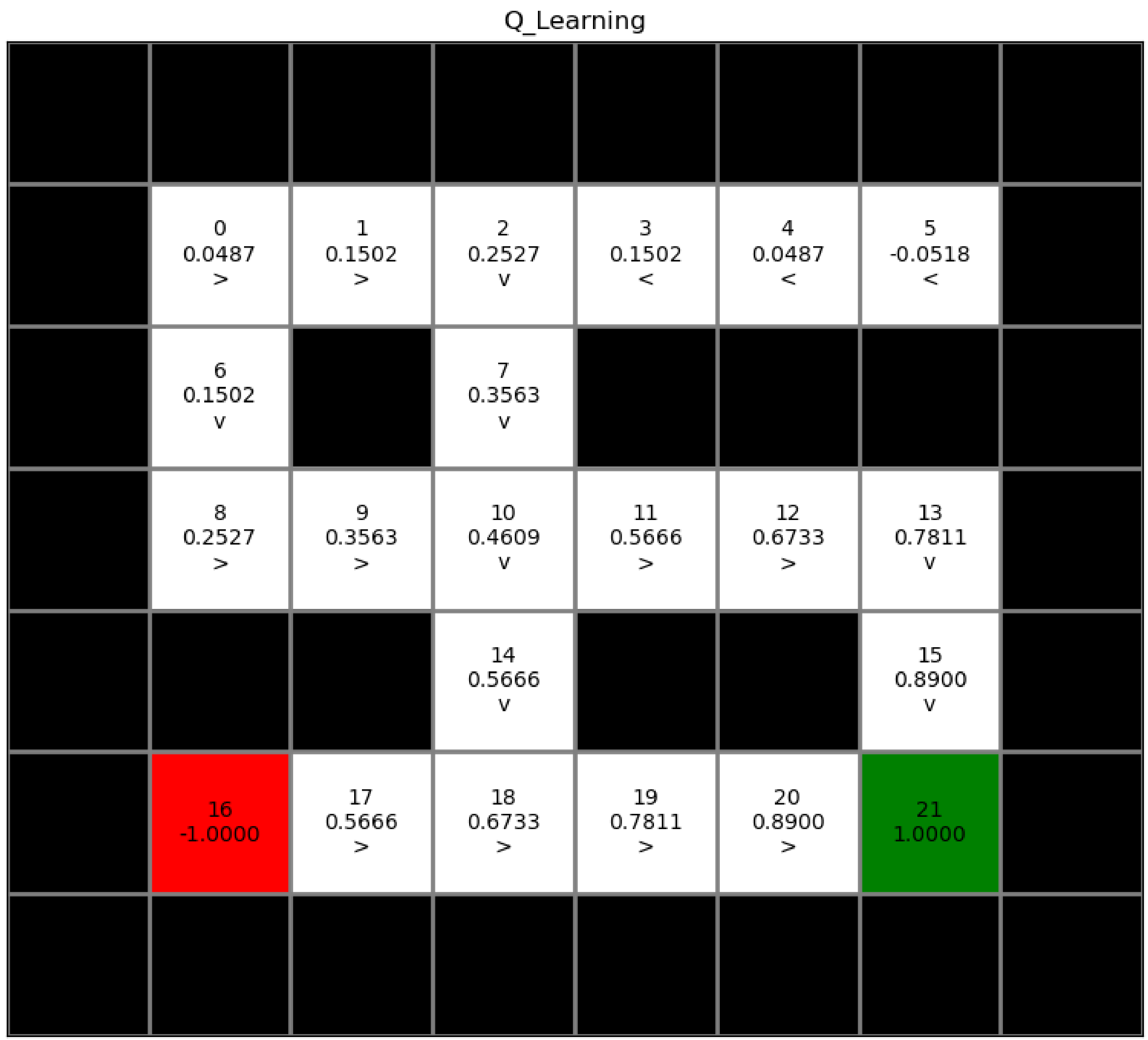
Sample Solutions: MC Policy Iteration



Sample Solutions: SARSA



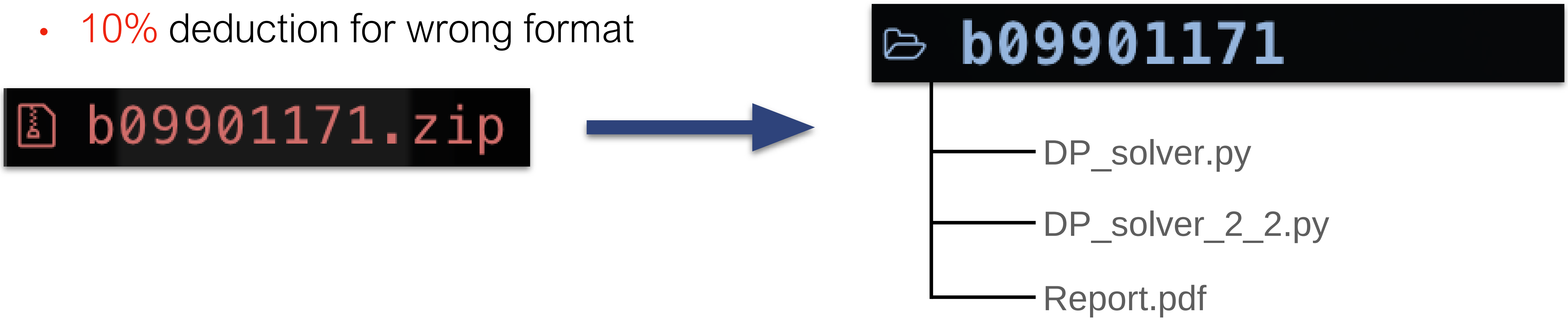
Sample Solutions: Q-Learning



Submission

Submission

- Submit on NTU COOL with following zip file structure
  - DP_solver.py: containing your implementation for HW 2.1
  - DP_solver_2_2.py: containing your code for HW 2.2
  - Get rid of pycache, DS_Store, etc.
  - Student ID with lower case
  - 10% deduction for wrong format



- **Deadline: 2023/10/19 Thu 09:30am**
- **No late submission is allowed**

Policy

Policy

Package

- You can use any Python standard library (e.g., heap, queue…)
- System level packages are prohibited (e.g., sys, os, multiprocess, subprocess…) for security concern

Collaboration

- Discussions are encouraged
- Write your own codes

Plagiarism & cheating

- All assignment submissions will be subject to duplication checking (e.g., MOSS)
- Cheater will receive an **F** grade for this course

Grade appeal

- Assignment grades are considered finalized two weeks after release

Contact

Questions?

- General questions
  - Use channel **<u>#assignment 2</u>** in slack as first option
  - Reply in thread to avoid spamming other people
- Personal questions
  - DM us on Slack: **<u>TA 劉冠廷 Guan-Ting Liu</u>**

    <u>TA 陳尚甫 Shang-Fu Chen</u>