

113-1 (Fall 2024) Semester

# Reinforcement Learning

## Assignment #2

Model-Free Prediction & Control

TA: Co Yong (楊可)

---

Department of Electrical Engineering  
National Taiwan University

# Outline

---

- Environment
- Tasks
- Code structure
- Grading
- Submission
- Policy
- Contact

# Environment

# Grid World

## State space

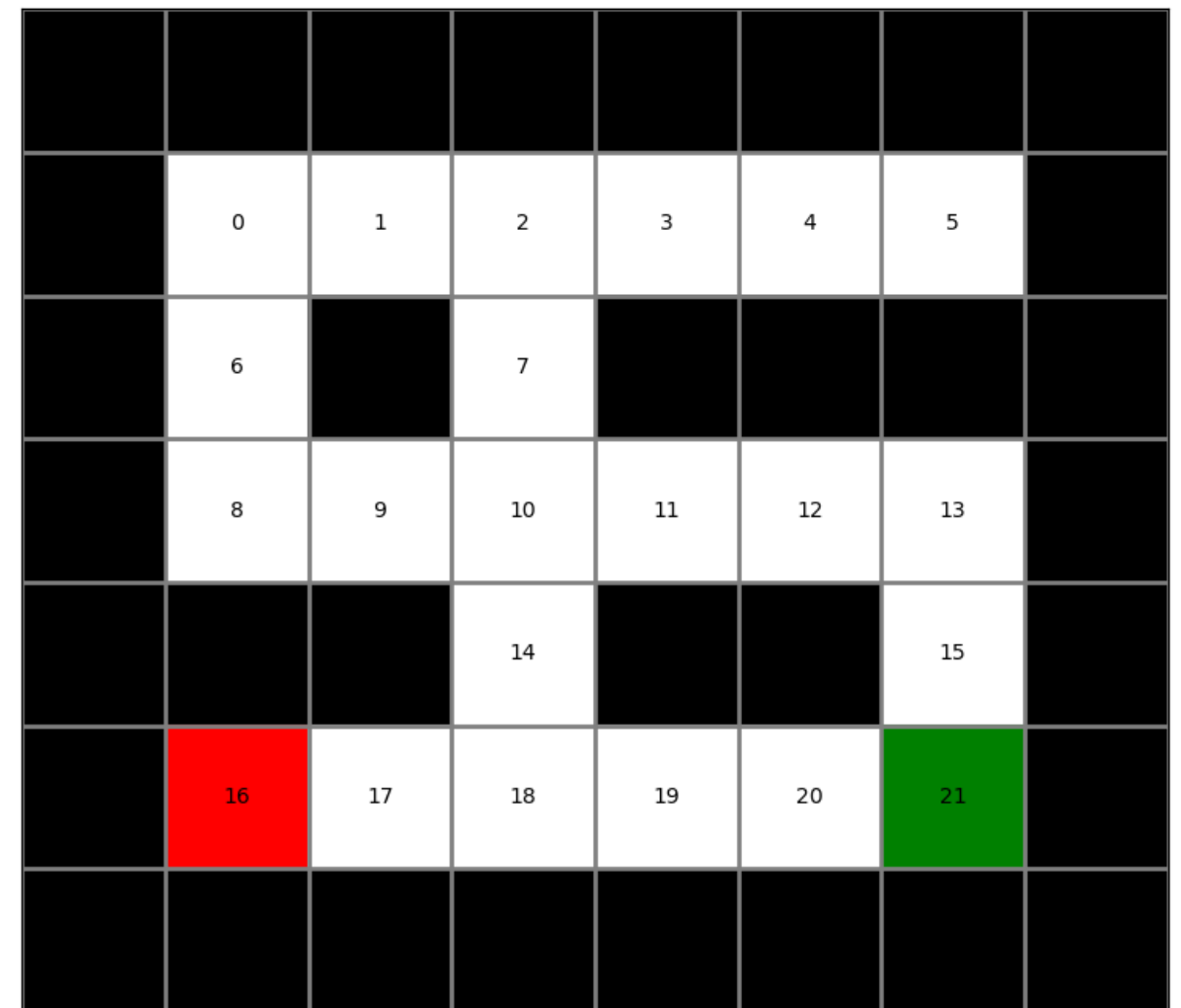
- Nonterminal states: Empty
- Terminal states: Goal (**Green**), Trap (**Red**)
- 0-indexed

## 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 **leaving** goal state
- Trap reward given after **leaving** trap state

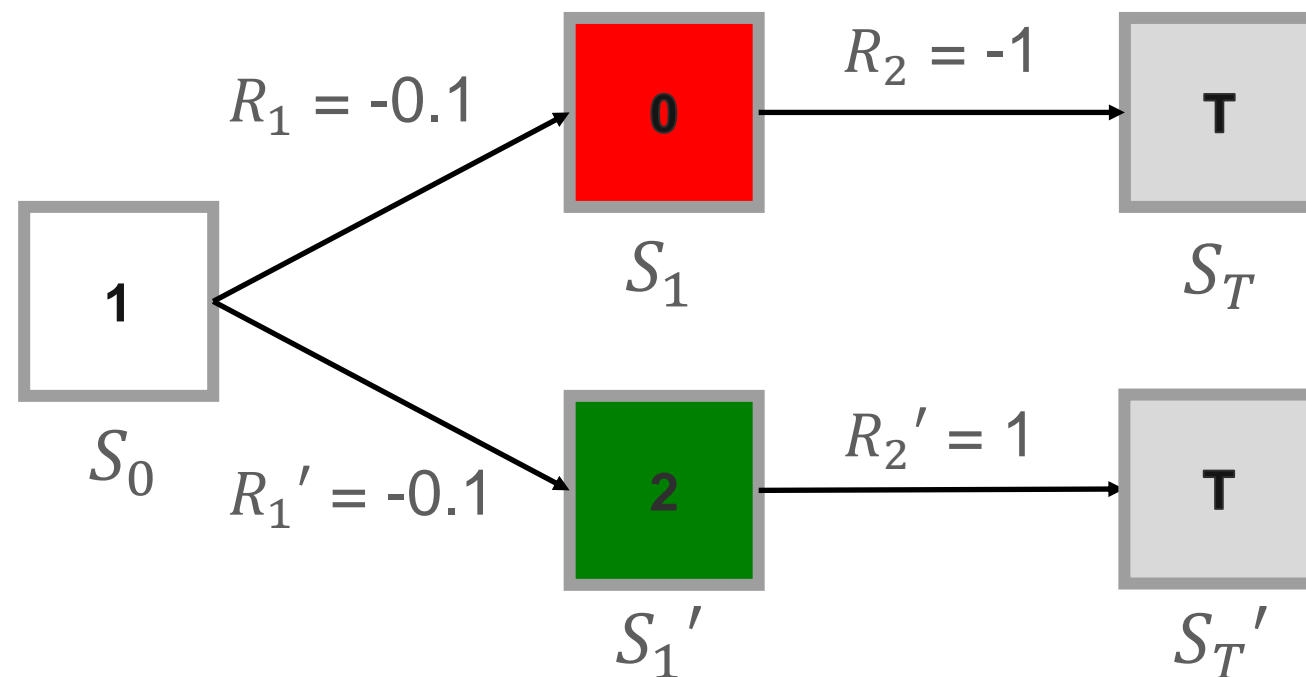
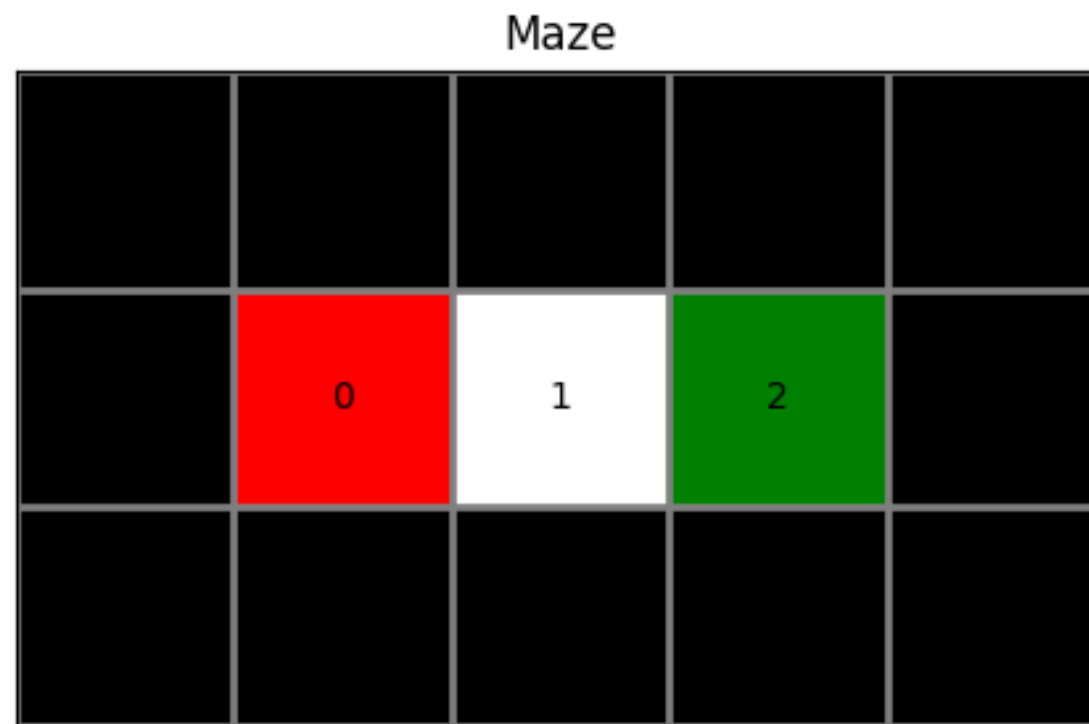


# Interaction with Environments

---

- Learn to interact with a OpenAI gym-like environment
- Grid World in this assignment is a **MDP** (defined by maze.txt, do not modified)
- Grid World functions:
  - step(**action**): Interact with the environment, **taking one parameter (action)**
  - reset(): Reset the environment to the initial state (only used once at the first step)
  - Update the **value/q-value** function with **states, rewards and done flags**

# Terminal State



- The final transition at the end of the  $i^{th}$  episode will be:  
 $(S_T^i, A_T^i, R_{T+1}^i, S_0^{i+1})$   
 where the first state of the  $(i + 1)^{st}$  episode is  $S_0^{i+1}$

# Tasks

## Model-Free Prediction & Control

# Prediction 1 - First-Visit Monte-Carlo Prediction

- Evaluate a policy by predicting the value function for each state
- Update the value function with First-visit Monte-Carlo method using **state, reward, and done** from the `collect_data()` function.
- Update the value function **per episode**

## First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy  $\pi$  to be evaluated

Initialize:

$V(s) \in \mathbb{R}$ , arbitrarily, for all  $s \in \mathcal{S}$

$Returns(s) \leftarrow$  an empty list, for all  $s \in \mathcal{S}$

Be careful with the index of  $S$  and  $R$  !  
( $S_{T-1}$  is goal or trap in our case)

Loop forever (for each episode):

Generate an episode following  $\pi$ :  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode,  $t = T-1, T-2, \dots, 0$ :

$G \leftarrow \gamma G + R_{t+1}$

Unless  $S_t$  appears in  $S_0, S_1, \dots, S_{t-1}$ :

Append  $G$  to  $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$



# Prediction 2 - TD(0)

- Evaluate a policy by predicting the value function for each state
- Update the value function with TD(0) method using `collect_data()` function
- Update the value function **per step**

## Tabular TD(0) for estimating $v_\pi$

Input: the policy  $\pi$  to be evaluated

Algorithm parameter: step size  $\alpha \in (0, 1]$

Initialize  $V(s)$ , for all  $s \in \mathcal{S}^+$ , arbitrarily except that  $V(\text{terminal}) = 0$

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

$A \leftarrow$  action given by  $\pi$  for  $S$

        Take action  $A$ , observe  $R, S'$

$V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$     **Be careful with the done flag**

$S \leftarrow S'$

    until  $S$  is terminal

# Prediction 3 - N-step TD

- Evaluate a policy by predicting the value function for each state
- Update the value function with n-step TD method using the `collect_data()` function
- Update the value function **per step** expect steps that out of range of the n-step TD

## *n*-step TD for estimating $V \approx v_\pi$

Input: a policy  $\pi$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , a positive integer  $n$

Initialize  $V(s)$  arbitrarily, for all  $s \in \mathcal{S}$

All store and access operations (for  $S_t$  and  $R_t$ ) can take their index mod  $n + 1$

Loop for each episode:

    Initialize and store  $S_0 \neq$  terminal

$T \leftarrow \infty$

    Loop for  $t = 0, 1, 2, \dots$ :

        If  $t < T$ , then:

            Take an action according to  $\pi(\cdot|S_t)$

            Observe and store the next reward as  $R_{t+1}$  and the next state as  $S_{t+1}$

            If  $S_{t+1}$  is terminal, then  $T \leftarrow t + 1$

$\tau \leftarrow t - n + 1$  ( $\tau$  is the time whose state's estimate is being updated)

            If  $\tau \geq 0$ : **Skip n-1 step**

$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$  **Be care of the index R !**

                If  $\tau + n < T$ , then:  $G \leftarrow G + \gamma^n V(S_{\tau+n})$  **Skip n-1 step** ( $G_{\tau:\tau+n}$ )

$V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$

    Until  $\tau = T - 1$

[Text book p.144](#)

# Control 1 – Every-Visit Monte-Carlo Prediction with $\epsilon$ -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  $\alpha$**
  - Using collected **trajectories** to update the value function **per episode**

## MC Policy Evaluation + $\epsilon$ -Greedy Improvement

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

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

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

- Improve policy based on new action-value function

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

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

Estimation Loss

# Control 1 – Every-Visit Monte-Carlo Prediction with $\epsilon$ -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  $\alpha$**
  - Using collected **trajectories** to update the value function **per episode**

## $\epsilon$ -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  $\epsilon$  choose an action at random

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

# Control 2 - SARSA: Temporal-difference Prediction

## TD(0) with $\epsilon$ -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 + $\epsilon$ -Greedy Improvement

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

Initialize  $S$

Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

Repeat (for each step of episode):

Take action  $A$ , observe  $R, S'$

Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

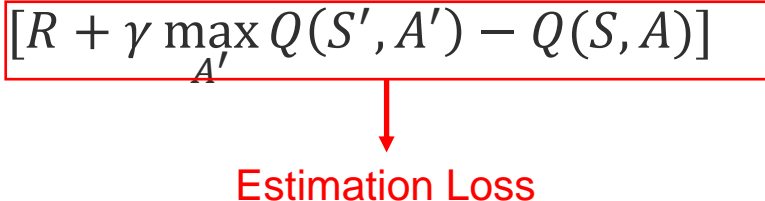
until  $S$  is terminal

Estimation Loss

# Control 3 - Q-Learning with $\epsilon$ -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 + $\epsilon$ -Greedy Improvement

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



# Code Structure

# algorithms.py

---

class **ModelFreePrediction()**

- Parent class for prediction algorithms
- Collect\_data(): use the policy to interact with the env

class **MonteCarloPrediction()**

- TODO: run()

class **TDZeroPrediction()**

- TODO: run()

class **TDNstepPrediction()**

- TODO: run()

class **ModelFreeControl()**

- Parent class for control algorithms

class **MonteCarloPolicyIteration()**

- TODO: run(),
- Auxiliary functions:  
policy\_evaluation(), policy\_improvement()

class **SARSA()**

- TODO: run()
- Auxiliary functions:  
policy\_eval\_improve()

class **Q\_learning()**

- TODO: run()
- Auxiliary functions:  
add\_buffer(), sample\_batch(), policy\_eval\_improve()

Feel free to add any function if needed

The implementation will be judged by run()



# Grading

# Grading (100%)

---

- Monte-Carlo prediction (10%)
  - Test cases (2% x 5 cases)
- TD(0) prediction (10%)
  - Test cases (2% x 5 cases)
- N-step TD prediction (10%)
  - Test cases (2% x 5 cases)
- Monte-Carlo Control (15%)
  - Test cases (3% x 5 cases)
- SARSA (15%)
  - Test cases (3% x 5 cases)
- Q-Learning (15%)
  - Test cases (3% x 5 cases)
- Report (25%) (Report Template: <https://www.overleaf.com/read/gqfnzvjlwcps#939ed3>)
  - Run Monte-Carlo prediction and TD(0) prediction for 50 seeds. Compare the resulting values with the GT values. Discuss the variance and bias. (16%)  
#prediction\_GT.npy is provided (calculated using iterative policy evaluation)
  - Discuss and plot learning curves under  $\epsilon$  values of (0.1, 0.2, 0.3, 0.4) on MC, SARSA, and Q-Learning (4%)
  - Discuss and plot loss curves under  $\epsilon$  values of (0.1, 0.2, 0.3, 0.4) on MC, SARSA, and Q-Learning (4%)
  - Using Weights & Bias (<https://wandb.ai/site>) to plot all figures in the report (1%)

# 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:  $r_{1b}$

$\vdots$

Episode 9: step1:  $r_{91}$ , step2:  $r_{92}$ , ..., step T:  $r_{9j}$

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

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:  $\text{abs}(EL_{11})$ ,  $\text{abs}(EL_{12})$ , ...,  $\text{abs}(EL_{1b})$

$\vdots$

Episode 9:  $\text{abs}(EL_{91})$ ,  $\text{abs}(EL_{92})$ , ...,  $\text{abs}(EL_{9j})$

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

# Prediction Bias & Variance

---

- Let  $V_{GT}$  be the ground truth value for a state.
- Let  $\hat{V}_i$  be the estimated value from the  $i$ -th run.
- Let  $\hat{V}_{avg}$  be the average predicted value over all runs for that state:

$$\hat{V}_{avg} = \frac{1}{n} \sum_{i=1}^n \hat{V}_i$$

- The *Bias* measures how far the average estimate is from the true value. For a single state, the *Bias* is:

$$Bias = \hat{V}_{avg} - V_{GT}$$

- The *Variance* measures the variability of the predicted values across different runs. For a single state, the *Variance* is:

$$Variance = \frac{1}{n} \sum_{i=1}^n (\hat{V}_i - \hat{V}_{avg})^2$$

# Criteria

---

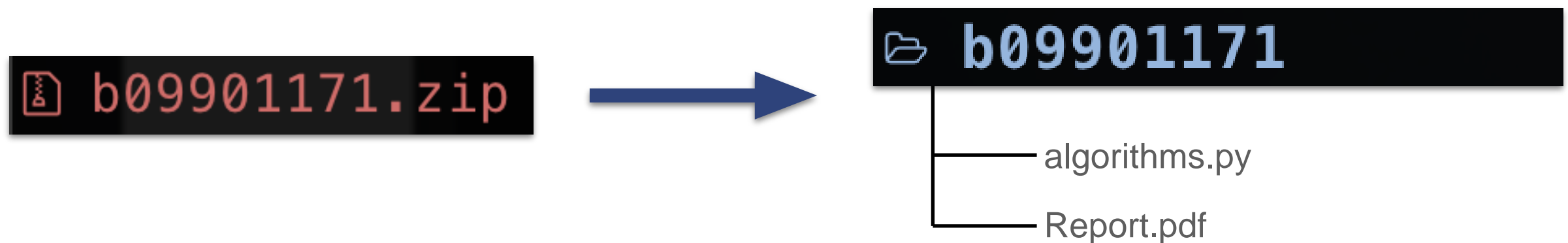
- Test cases:
  - Call `run()` and check the final output
  - Prediction Tasks: Check the **state values** after evaluation (**tolerance: 0.005**)
  - Control 1, 2: Check the resulting *max\_state\_values* on every state (**tolerance: 0.1**)
  - Control 3: Check the resulting *max\_state\_values* on every state (**tolerance: 0.001**)
  - Run time limit **3 minute** for Prediction cases to avoid infinite loops
  - Run time limit **5 minute** for Control cases to avoid infinite loops
- Sample solutions are provided for reference
  - Optimal policy may not be unique

# Submission

# Submission

---

- Submit on NTU COOL with following **zip** file structure
  - algorithms.py: containing your implementation for HW2
  - Get rid of pycache, DS\_Store, etc.
  - Student ID with lower case
  - **10%** deduction for wrong format



- **Deadline: 2024/10/17 Thu 09:30am**
- **No late submission is allowed**

# Policy



# Policy

---

## Package

- You can use any Python standard library (e.g., heap, queue...)
- Don't print anything out
- System level packages are prohibited (e.g., sys, os, multiprocessing, subprocess, shutil, pathlib, ...) for security concern, **import any one of them will result in 0 score (even if you did not call it)**

## Collaboration

- Discussions are encouraged
- Write your own codes

## Plagiarism & cheating

- All assignment submissions will be subject to duplication checking (e.g., MOSS)
- Plagiarism will receive an **F** grade for this course
- LLM for code writing is not prohibited. However, it's best to just write the code yourself, without the help of any LLM. **(We will not accept using LLM as an excuse for plagiarism.)**

## Grade appeal

- Assignment grades are considered finalized two weeks after release

# Contact

# Questions?

---

- General questions
  - Use channel **#assignment** in slack as first option
  - Reply in thread to avoid spamming other people
- Personal questions
  - DM me on Slack: **TA 楊可 r12946014**

