114-1 (Fall 2025) Semester

Reinforcement Learning

Assignment #3

TA: Huai-Chih Wang(王懷志)

Department of Electrical Engineering
National Taiwan University

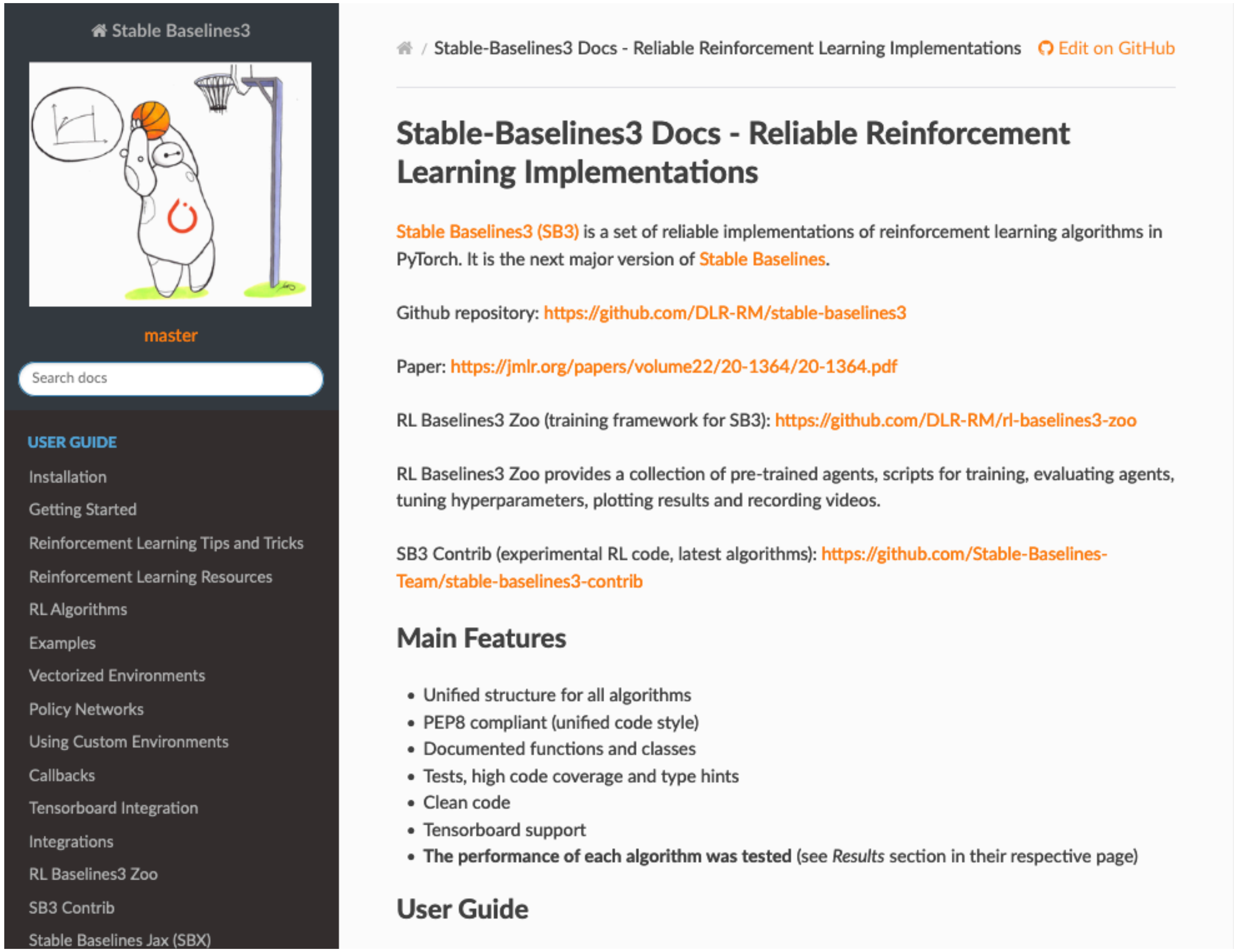
Outline

- Stable baseline 3 & Gymnasium
- 2048 The Game
- Environment
- Code structure
- Grading
- Report
- Policy
- Submission
- Contact

Stable Baselines 3 & Gymnasium

Stable Baselines 3

- Stable Baselines is a package that provides training APIs for RL training.
- Useful for training Gym environment on basic RL algorithms.
- Documentation



Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations

Stable Baselines3 (SB3) is a set of reliable implementations of reinforcement learning algorithms in PyTorch. It is the next major version of Stable Baselines.

Github repository: https://github.com/DLR-RM/stable-baselines3

Paper: https://jmlr.org/papers/volume22/20-1364/20-1364.pdf

RL Baselines3 Zoo (training framework for SB3): https://github.com/DLR-RM/rl-baselines3-zoo

RL Baselines3 Zoo provides a collection of pre-trained agents, scripts for training, evaluating agents, tuning hyperparameters, plotting results and recording videos.

SB3 Contrib (experimental RL code, latest algorithms): https://github.com/Stable-Baselines-Team/stable-baselines3-contrib

Main Features

- Unified structure for all algorithms
- PEP8 compliant (unified code style)
- Documented functions and classes
- Tests, high code coverage and type hints
- Clean code
- Tensorboard support
- **The performance of each algorithm was tested** (see *Results* section in their respective page)

User Guide

Supported Algorithms

- A2C (Asynchronous Advantage Actor Critic)
- DDPG (Deep Deterministic Policy Gradient)
- DQN (Deep Q Network)
- PPO (Proximal Policy Optimization)
- SAC (Soft Actor Critic)
- TD3 (Twin Delayed DDPG)

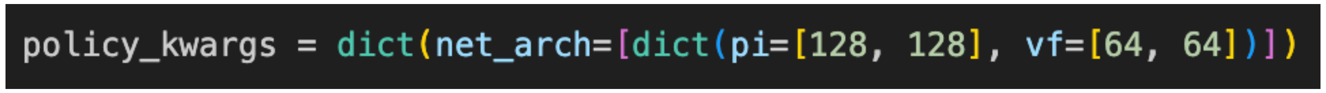
- Note that some algorithms don't support every action and observation space type.
- Make sure that the algorithm you pick fits the environment for this assignment.
- Read RL Algorithms and the documentation of each algorithm for more detail.

Policy Networks

- Each algorithm (PPO, DQN, A2C, etc.) has a built-in policy class.
  - 64 units (per layer) for PPO/A2C/DQN
  - 256 units for SAC

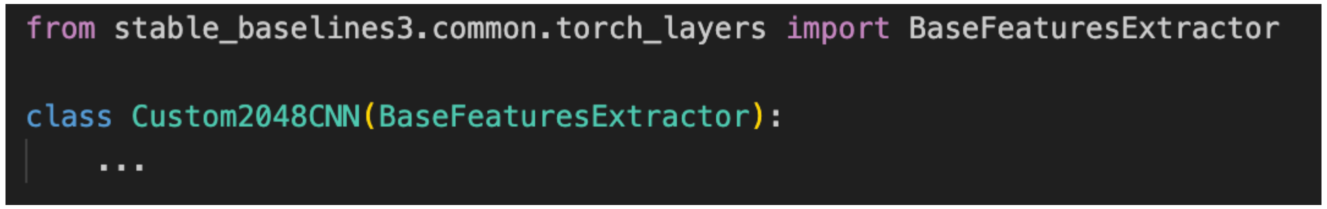
- Types of Policies
  - MlpPolicy: For vector inputs (e.g., numbers)
  - CnnPolicy: For image inputs
  - MultiInputPolicy: For environments with mixed inputs (e.g., images + vectors)

- Customization:
  - Policy network:



```
policy_kwargs = dict(net_arch=[dict(pi=[128, 128], vf=[64, 64])])
```

  - Features extractor:



```
from stable_baselines3.common.torch_layers import BaseFeaturesExtractor

class Custom2048CNN(BaseFeaturesExtractor):
    ...
```

Vectorized Environments

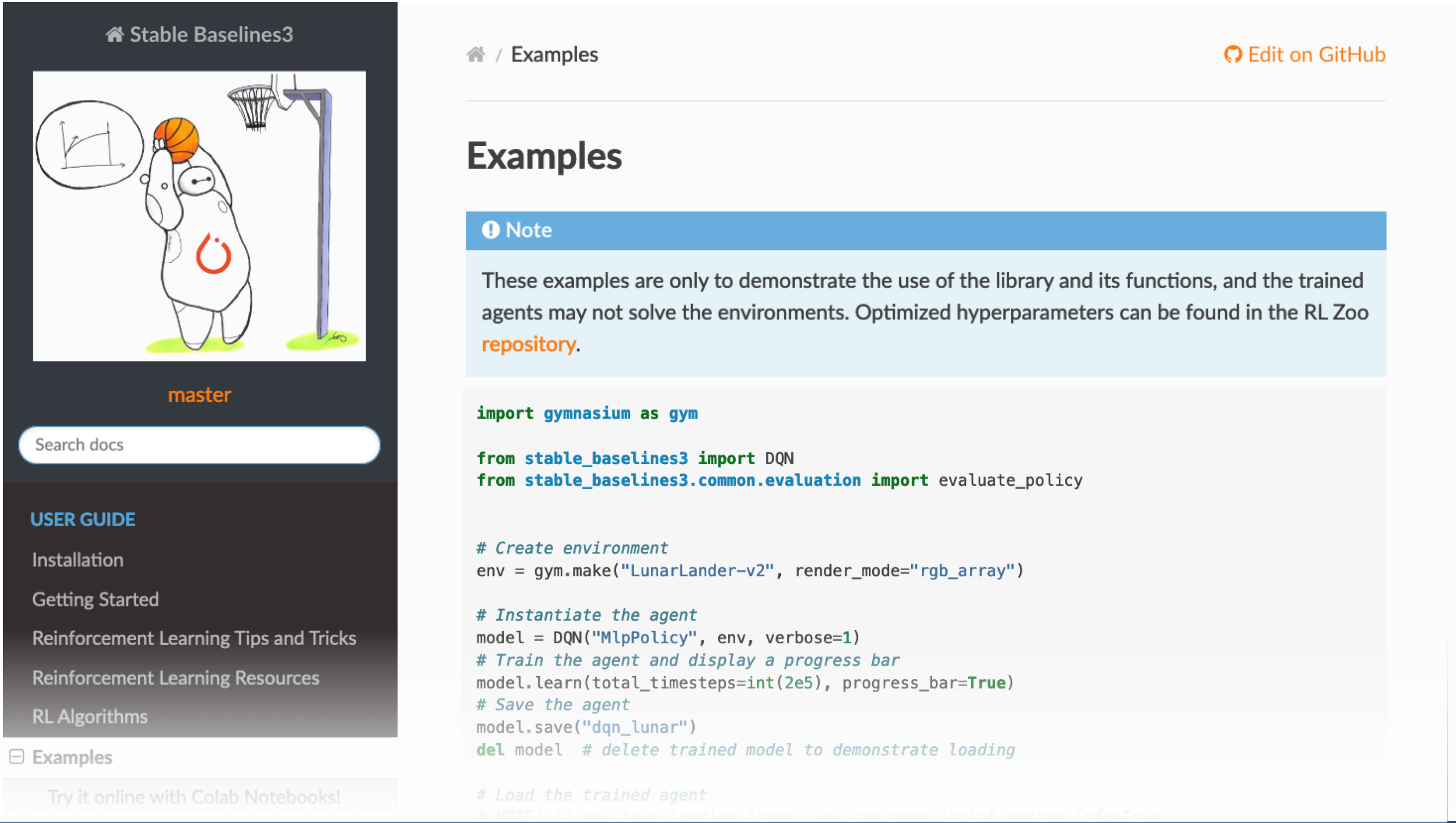
- Vectorized Environments are used to stack multiple independent environments into a single environment.
- Training the agent on a parallel environment helps speed up the training process.
- You can use VecEnv for training and still use the original Env for evaluation.
- Example: SubprocVecEnv can create separate envs that agents interact with in parallel.

- Note: VecEnv wraps the environment with Gym 0.21 API logic, so:
  - Output of VecEnv.step() would be obs, reward, termination, info instead.
  - Output of VecEnv.reset() would be obs only.

    (See VecEnv API vs Gym API)

Tutorials

- Stable Baselines 3 Examples



Examples

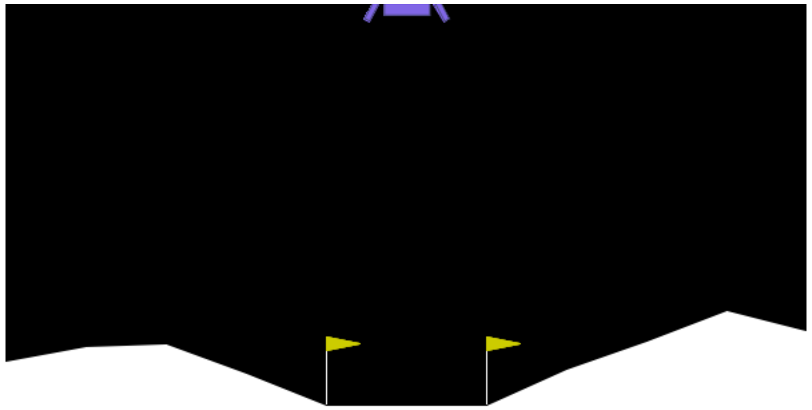
Note

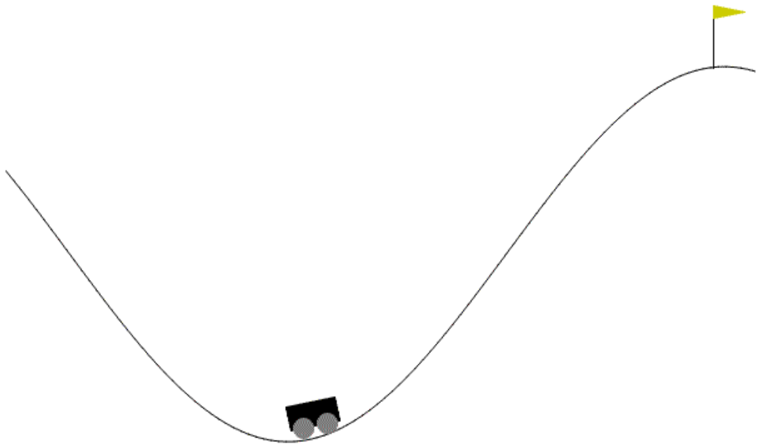
These examples are only to demonstrate the use of the library and its functions, and the trained agents may not solve the environments. Optimized hyperparameters can be found in the RL Zoo repository.

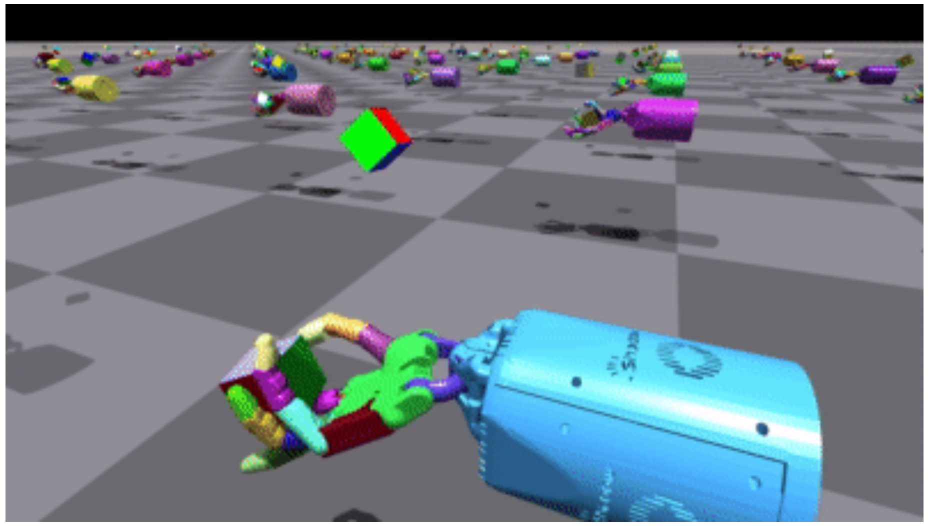
```
import gymnasium as gym

from stable_baselines3 import DQN
from stable_baselines3.common.evaluation import evaluate_policy


# Create environment
env = gym.make("LunarLander-v2", render_mode="rgb_array")

# Instantiate the agent
model = DQN("MlpPolicy", env, verbose=1)
# Train the agent and display a progress bar
model.learn(total_timesteps=int(2e5), progress_bar=True)
# Save the agent
model.save("dqn_lunar")
del model  # delete trained model to demonstrate loading

# Load the trained agent
# NOTE: if you have loading issue, you can pass `print_system_info=True`
```

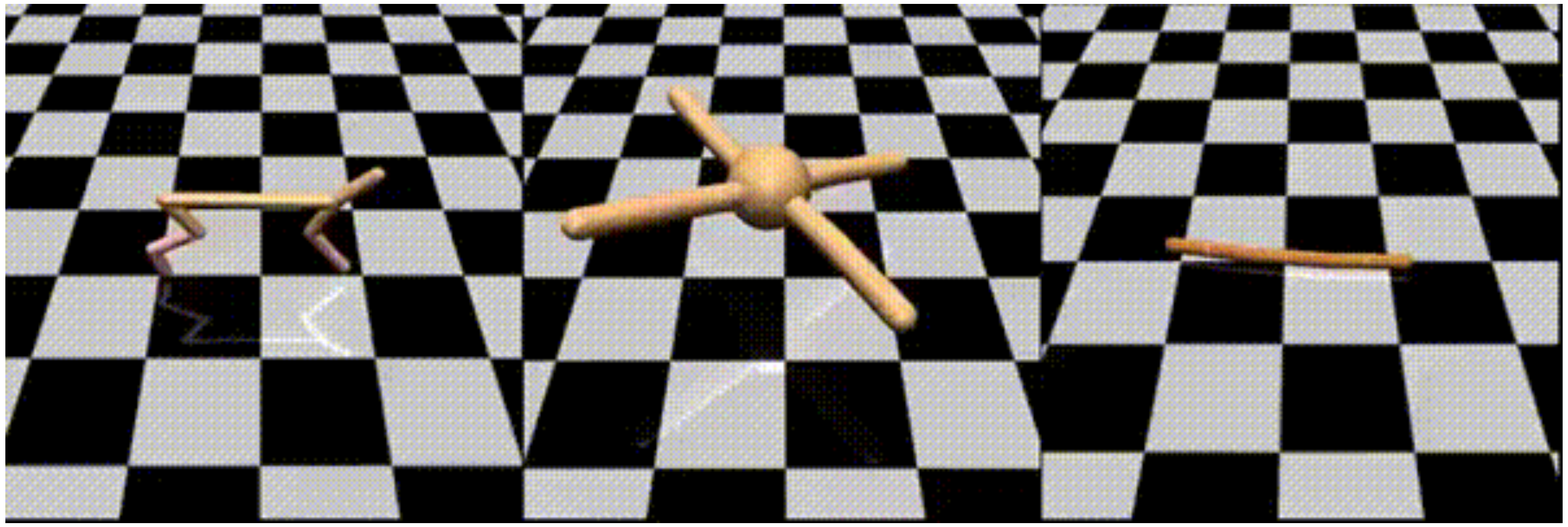
Environment Packages

- There are many open-source environment packages for RL training.
- Examples: “OpenAI Gym”, “Deepmind Control”, “Nvidia Isaac Gym”, etc.
- For this assignment, we will mainly focus on “OpenAI Gym / Gymnasium” for its compatibility with “Stable Baselines 3”.



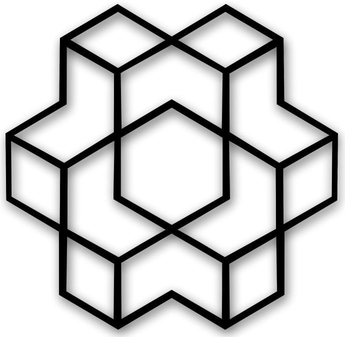






OpenAI Gym / Gymnasium

- Gym is an open source package developed by OpenAI to provide a standard API to communicate between learning algorithms and environments.
- The Farama Foundation announced the release of Gymnasium by October 2022 and took place in the future maintenance of OpenAI Gym. (Announcing The Farama Foundation)
- For this assignment, we will be using Gymnasium.
- OpenAI Gym / Gymnasium



Environment Methods and Tips

- Methods: (For Gymnasium or Gym version after v0.26.0)

  - step(action): Interacts with the environment. Returns obs, rewards, termination, truncation, info.

    "True" if episode ends because it reaches max episode length.

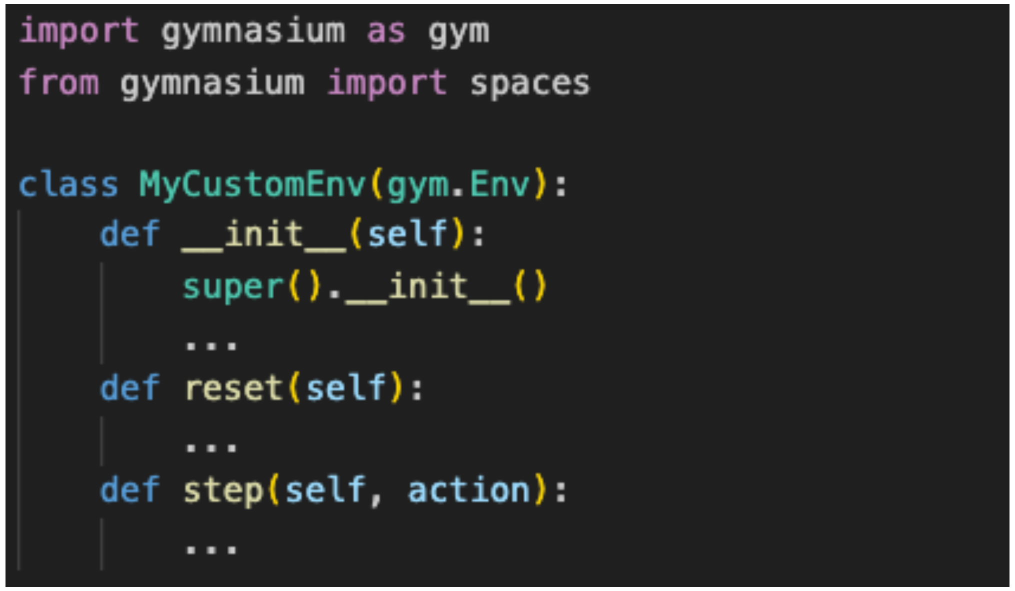
    "True" if the episode ends due to termination conditions. (ex: Agent made illegal moves or finished a task)

  - reset(): Resets the environment. Returns obs, info.

    A dictionary of information that might be useful. (ex: {"success": False, "score": 42})

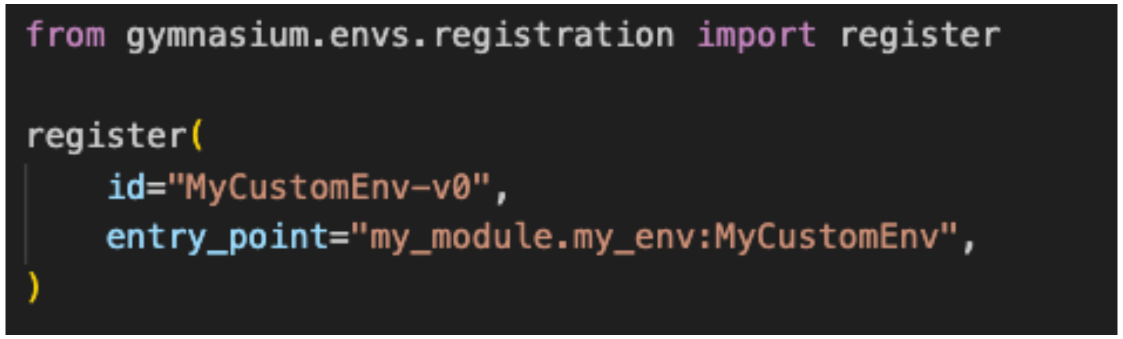
Custom Environments

- Step 1: Create Your Environment Class
  - Save it in “my_module/my_env/“



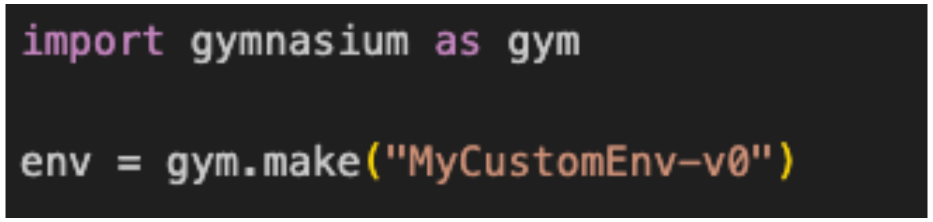
```python
import gymnasium as gym
from gymnasium import spaces

class MyCustomEnv(gym.Env):
    def __init__(self):
        super().__init__()
        ...
    def reset(self):
        ...
    def step(self, action):
        ...
```

- Step 2: Register the Environment
  - Register in your training script



```python
from gymnasium.envs.registration import register

register(
    id="MyCustomEnv-v0",
    entry_point="my_module.my_env:MyCustomEnv",
)
```

- Step 3: Make the Environment
  - Make the env before training or eval



```python
import gymnasium as gym

env = gym.make("MyCustomEnv-v0")
```

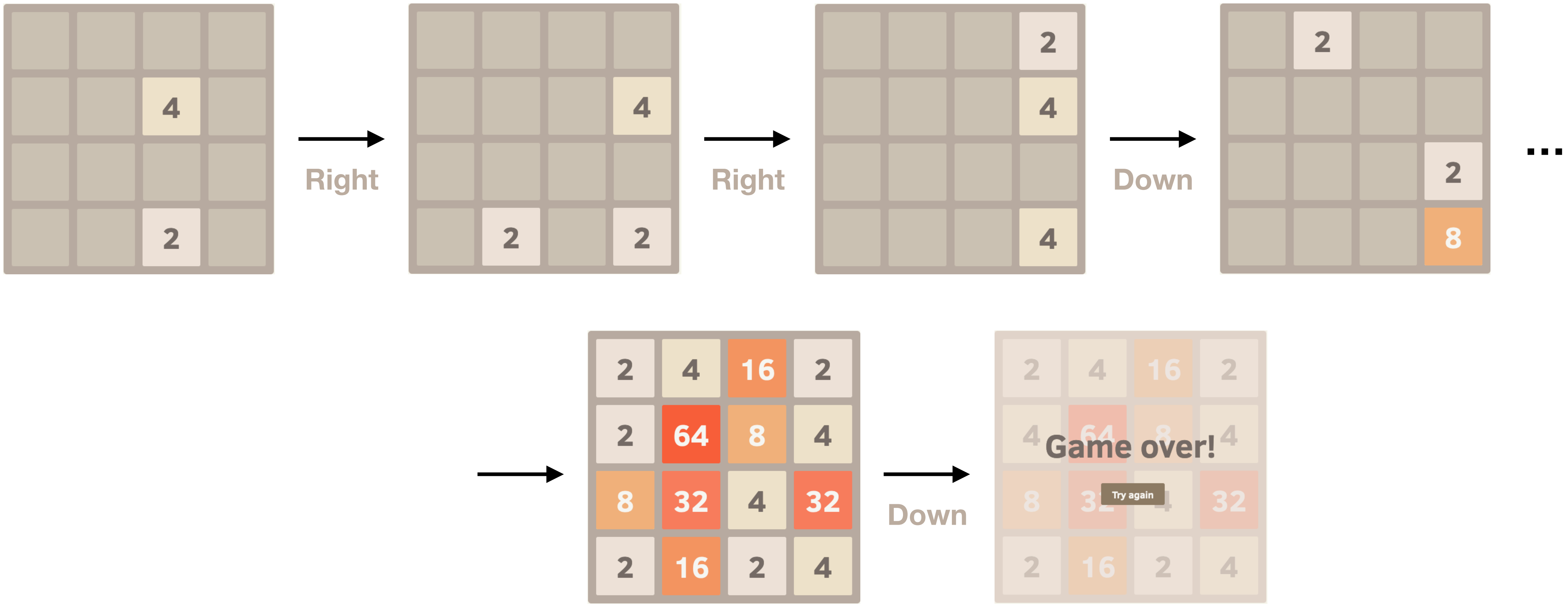
Notes

- There was a major update for the Gym API after version 0.26.0
- If you're using environments from older versions of Gym, you might want to watch out for compatibility issues.
- Release notes for v0.26.0

2048 The Game

2048

- Use ↑, ↓, → and ← to move number tiles on the board.
- When the same number collide, they will merge into a tile with the total value of two tiles.
- You will also get scores for each tiles combined.
- A new tile (with value 2 or 4) will generate in a random empty space after each move.
- The game ends when there is no legal move left.
- Play 2048



Tasks

- You need to train an agent to play 2048 using the Stable Baselines 3 package.

- A simulated 2048 environment will be provided, you will:
  - Choose and tune hyper parameters with algorithms from SB3.
  - Modify the environment (Terminal conditions, reward shaping…).
  - Work on other techniques that can help agent perform better.

- Write a report to let TA know what you have done.

Environment

Termination

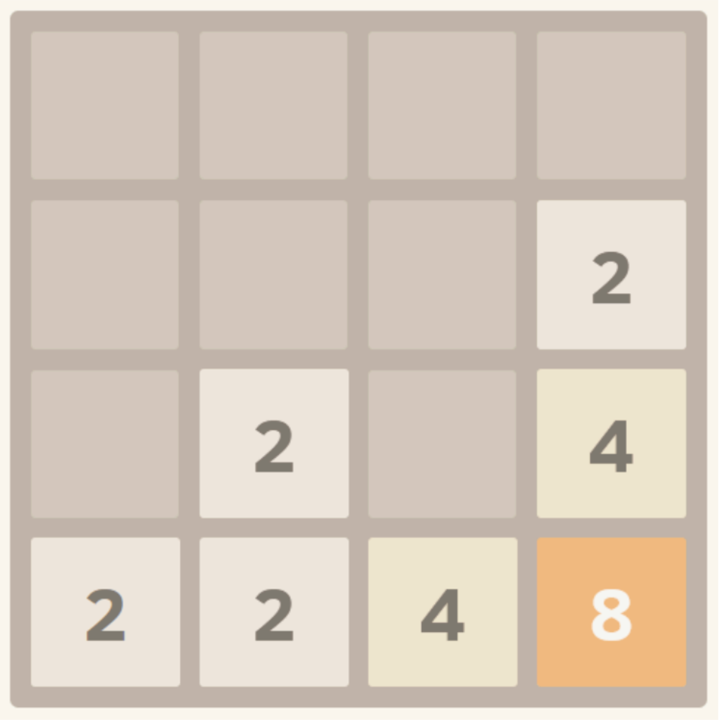
- For this environment, there will be 2 scenarios causing the step function to return True for termination.
  1. Termination for game end:
     - The episode terminates when there is no legal move left.
  2. Termination for illegal moves:
     - To avoid infinite loops, the environment will consider action that causes no state change an “illegal move”.
     - In evaluation, the episode terminates when the agent executes an illegal move.
     - You can modify the training code to let agent explore other actions even after illegal moves.

State and Action Space

- State: Box(0,1, (16, 4, 4))
- Action: Discrete(4)

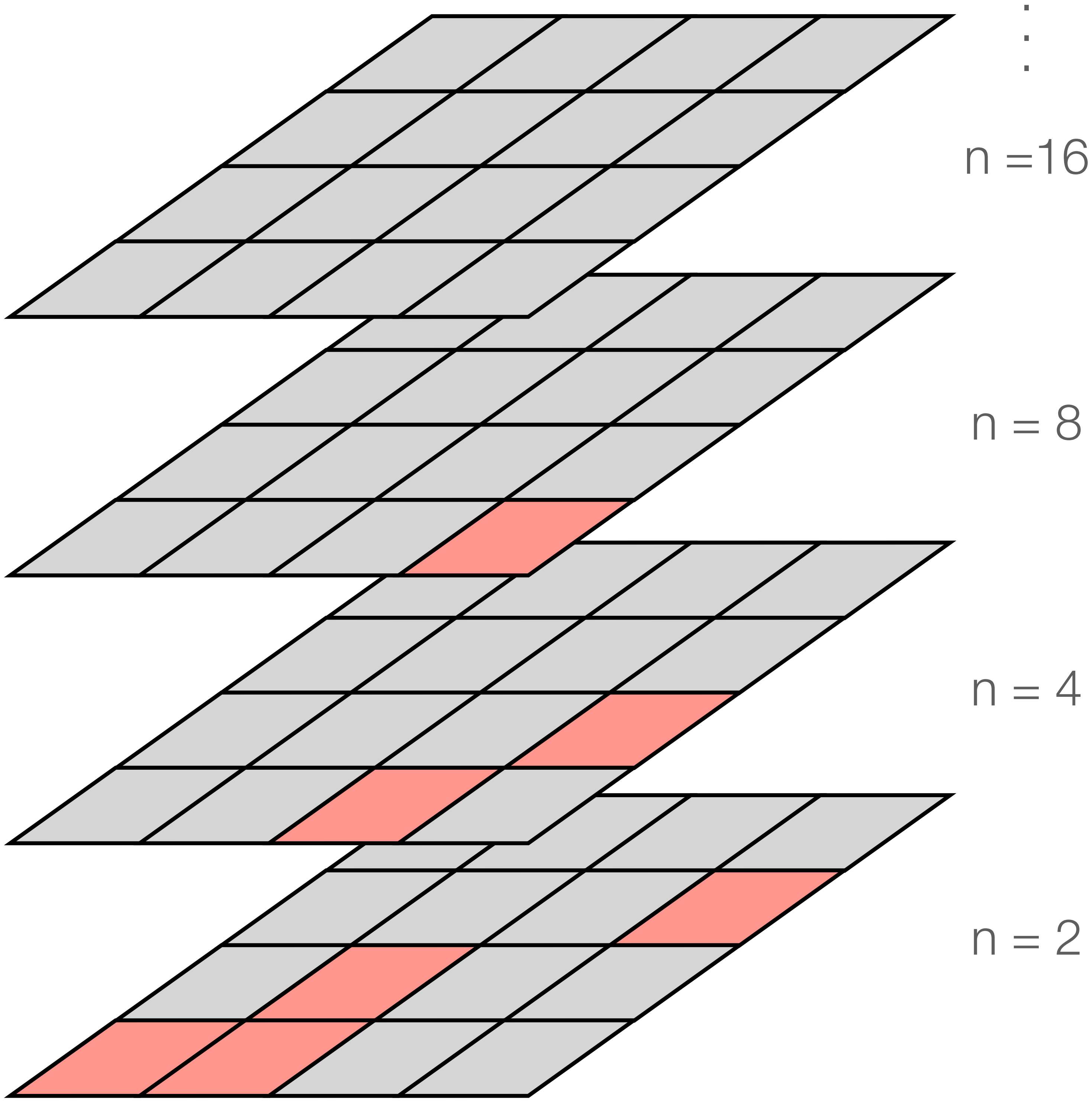
  (↑, →, ↓, ←)

State visualization:



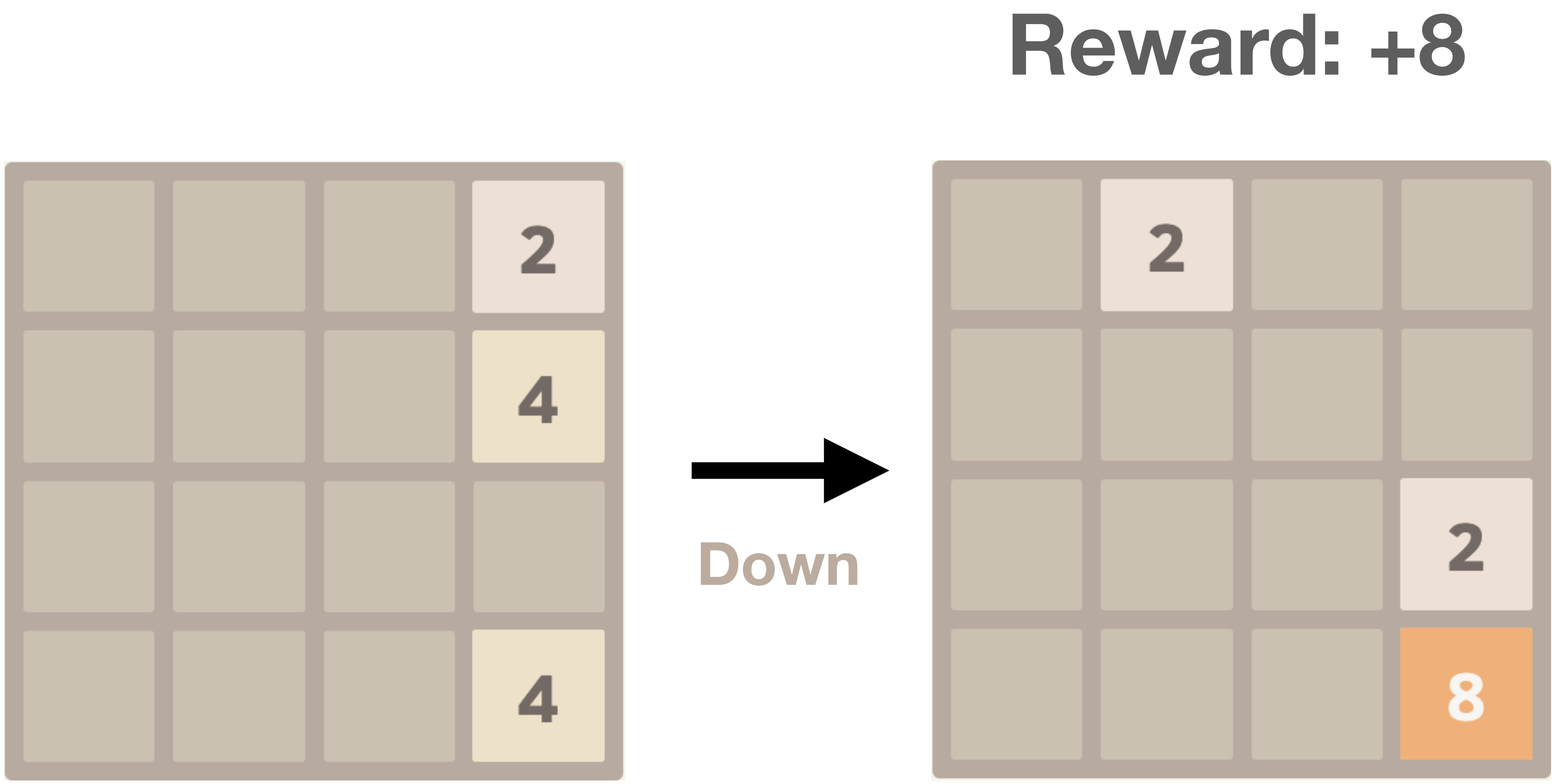
			2
	2		4
2	2	4	8

=



Reward

- The default reward will be the added score for each step.
- Grades will only consider the “highest tile value”, not the reward.
- You can modify the reward during the training stage.



Code Structure

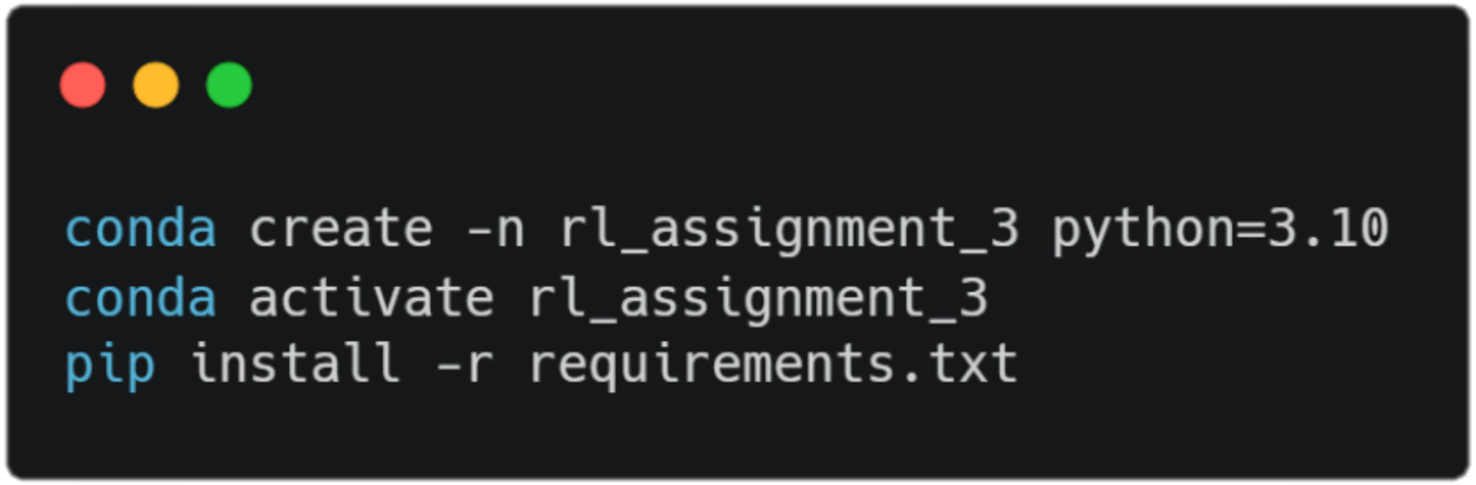
Root directory

- rl_hw3_2048game/
  - The main directory for task “2048 The Game”.

env_2048game

requirement.txt

- Conda



```
conda create -n rl_assignment_3 python=3.10
conda activate rl_assignment_3
pip install -r requirements.txt
```

- venv
  - Sorry for virtual environment users. Stable Baseline 3 is hard to install using venv with pip. Do not use venv in PA3.

Python Files

- train.py
  - Sample code for training your model with SB3.
  - Modify my_config to set hyper parameters and configurations.
- my2048_env.py
  - Environment used for training your agent.
  - Feel free to modify any parts of this file to help agent learn better.
- eval.py
  - Load saved model and run 100 rollouts for evaluation.
- eval2048_env.py
  - Environment that will be used for evaluate agent’s performance.
  - TA will use “eval.py” and “eval2048_env.py” for evaluation, so don’t modify them.

- Note: You are encouraged to modify “train.py” and “my2048_env.py” as much as you like. Still, do keep in mind that the model will be put in our setting for evaluation.

Other folders

- models/
  - Folder where sample models are saved.
- envs/
  - Folder for __init__.py, my2048_env.py and eval2048_env.py.

Tips for 2048

Tips

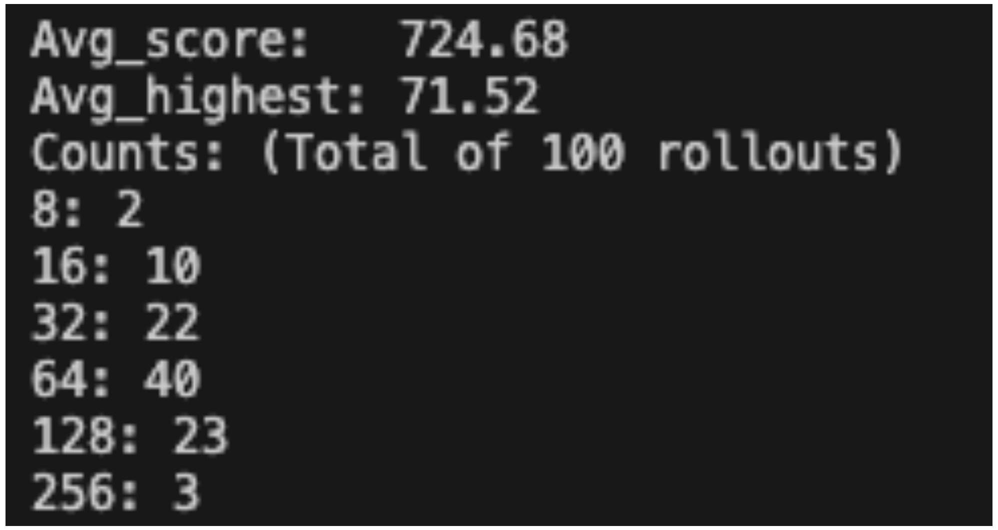
- train.py
  - Try different algorithms from SB3 and see how each of them performs. (Note that not all algorithms are compatible due to state and action space type.)
  - Try different policy networks. You can also build and use your custom network architecture and feature extractor. (See SB3 Policy Network)
  - Try set different learning rates or total timestep numbers.
- my2048_env.py
  - Try giving the agent multiple tries before terminating the episode when the agent executes an illegal move during training.
  - Try different sets of weight to give additional rewards for certain board patterns that helps the agent learn different strategies.

Grading

Grading

- Baselines (60%)
  - Simple baseline: Reach 128 for at least one rollout. (Public: 6%, Private: 6%)
  - Medium baseline: Reach 256 for at least one rollout. (Public: 6%, Private: 6%)
  - Strong baseline: Reach 512 for at least one rollout. (Public: 6%, Private: 6%)
  - Master baseline: Reach 1024 for at least one rollout. (Public: 6%, Private: 6%)
  - Nightmare baseline: Reach 2048 for at least one rollout. (Public: 4%, Private: 4%)
  - Insane baseline: Reach 4096 or over for at least one rollout. (Public: 2%, Private: 2%)

- Report (40%)
  - See "Report" slides for details



```
Avg_score:    724.68
Avg_highest: 71.52
Counts: (Total of 100 rollouts)
8: 2
16: 10
32: 22
64: 40
128: 23
256: 3
```

Criteria

- Test cases:
  - We will run your model on seeded environments and check the best tile value ("Highest") for baseline evaluations.
  - Public test cases: 100 rollouts, using seed 0 to 99.
  - Private test cases: 100 rollouts, using another 100 seeds selected by TA.
  - Public and private will be tested separately.
  - Run time for each case is limited to **10 minute** to prevent infinite loops.

    (We will give extra run time to those whose highest scores exceed 2048)

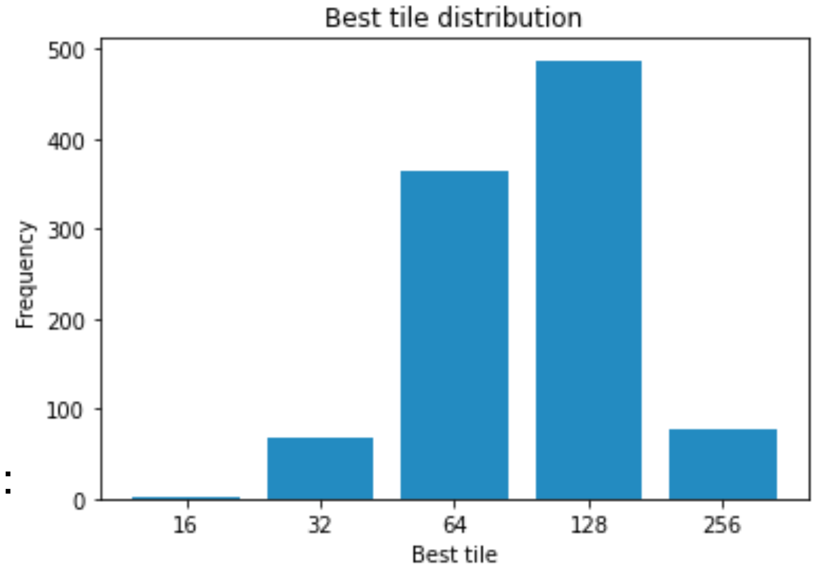
- Only 1 model will be loaded and tested for both test cases.
- For grading, we will use "eval.py" and "eval2048_env.py" to evaluate your performance, please make sure that **"eval.py" can load your model** without modifying it.
- Model should be trained with algorithms from SB3.
- Please make sure that we can briefly reproduce your results.

Report

Report

- Q1. Try using at least two different algorithms to train your 2048 model. Present your results and discuss possible reasons why some algorithms perform better than others in this task. (12%)
  - Show your results (avg_highest, avg_score.) and simply discuss
- Q2. Show your best tile distribution in 2048 (see Figure1) (8%)
- Q3. Describe what you have done to train your best model? (10%)
- Q4. Choose an environment (cannot be 2048) from the Gymnasium library and train an agent. (10%)
  - Show your results as many as possible (videos of the results, learning curve, …)
  - Write down anything you find interesting or difficult using pen and paper, and include a photo or a scan of it at the end of the report. (at least 50 words)
- Use Overleaf template and save in PDF format.
  - Overleaf template

Figure 1:



Policy

Policy

Package

- You can use any Python standard library (e.g., heap, queue…).
- Don't print anything or you'll get 10% deduction for doing it.
- System level packages are prohibited (e.g., sys, os, shutil…), importing any of them will result in 0 score.

Collaboration

- Discussions are encouraged.

Plagiarism & cheating

- All assignment submissions will be subject to duplication checking (e.g., MOSS).
- Cheater will receive an **F** grade for this course.
- Using LLM for coding is not prohibited. However, we will not accept using LLM as an excuse for plagiarism.

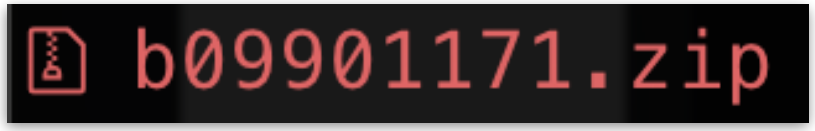
Grade appeal

- Assignment grades are considered finalized two weeks after release.

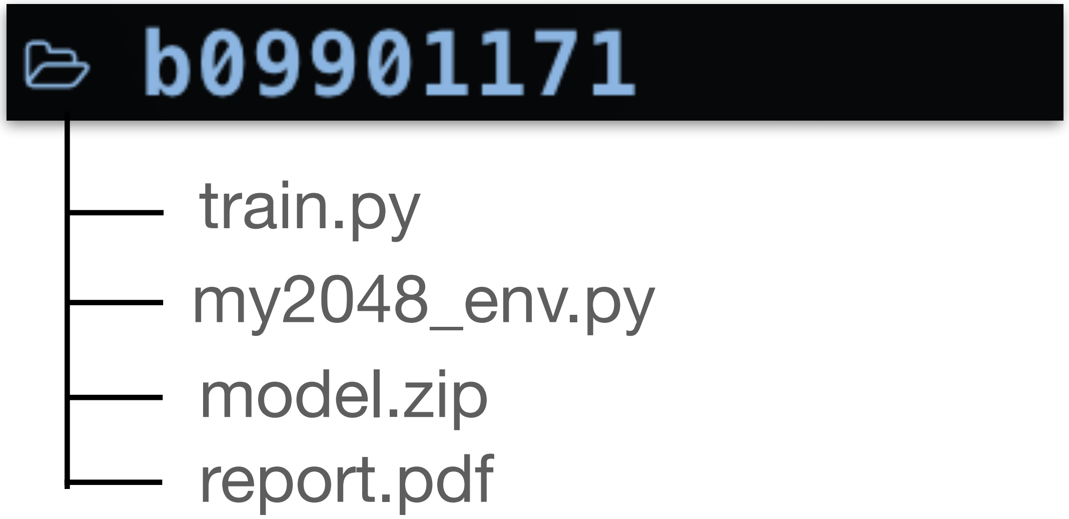
Submission

Submission

- Submit on NTU COOL with following zip file structure:
  - train.py, my2048_env.py: Containing your implementation for 2048.
  - model.zip: Saved 2048 model that can be loaded with eval.py.
  - Get rid of pycache, DS_Store, etc.
  - Student ID with lower case
  - 10% deduction for wrong format & printing
  - 0 score for importing system level packages



b09901171.zip



b09901171

- train.py
- my2048_env.py
- model.zip
- report.pdf

- Deadline 2025/11/05 Wed 11:59pm
- No late submission is allowed

Contact

Questions?

- General questions
  - Use channel **#assignment** in slack as first option
  - Reply in thread to avoid spamming other people
- Personal questions
  - DM us on Slack: **TA 王懷志 d14942005**

TA 王懷志 d14942005 (you)