113-1 (Fall 2024) Semester
Reinforcement Learning

Assignment #3

TA: Huai-Chih Wang(王懷志)

———————————

Department of Electrical Engineering
National Taiwan University

Outline

- Task1-GridWorld

- Environment1-GridWorld

- Task2-2048game

- Environment2-2048game

- Stable baseline 3 & Gymnasium

- Code structure

- Report

- Grading

- Policy

- Submission

- Contact

Task1-GridWorld

TODO

- For this part of assignment, we will add six new states to the GridWorld environment.

- Follow the rules for these states and modify step() and reset() function in gridworld.py.

- We hope this will help you learn how to design custom env using Gymnasium.

All states

Environment1-GridWorld

Terminal states

- Original terminal states

  - Trap and Goal: Terminate after taking one step at the Trap or Goal state

- New terminal states

  - Exit: Terminate after taking one step at the Exit state with a much less positive exit reward.

  - Lava: Terminate the trajectory when entering the Lava with only the step reward.

    (Note: Termination occurs upon entering, not exiting)

Non-terminal state

- Original non-terminal states

  - Empty: The settings of Empty states remain the same as HW1 and HW2

- New non-terminal states

  - Bait: Entering bait state can get a bait reward but will have bait penalty afterward. (Note: after biting the bait the bait state become empty)

  - Door and Key: The Door will be opened when entering the Key state.

  - Portal: Teleport to another portal once you leave the Portal state and hit a wall. Portal states must be adjacent to wall states (Note: There are only two portals in our env)

Reward

- Step reward is given at every transition

- Goal reward is given after leaving Goal state

- Trap reward is given after leaving Trap state

- Exit reward is given after leaving Exit state

- Enter Key state will get step reward and open the door

- Enter Door state will only get step reward

- Enter Lava state will only get step reward

- Enter Portal state will only get step reward

- Enter Bait state will get bait reward, and every step reward afterward is the sum of the step reward and bait step penalty.

All states

| | Empty | | Goal | Trap | Lava | Exit | Key | Door | Bait | Portal | |

Initial state and reset

- Initial (Do not modify)

  - Agent will not initialize at Door, Key, Bait, Lava states.

  - In assignment_3, the tasks are left-to-right. The initial state will be at the left of the Lava and Door, and the Goal state will be at the right of the Lava and Door.

  - Door and Key will exist at the same time.

  - There will only be a single pair of portal states.

- Reset (TODO)

  - Select one valid initial state.

  - Close the Door if the Door is opened.

 - Place the bait back in the Bait state if it is bitten.

Inner State and Outer State

- The outer state is returned by the environment for Deep RL policy.

- The inner state indicates the current agent position in GridWorld.

- The outer state is initially the same as the inner state but can differ when door is opened.
  If the door is opened, the outer state is *inner_state + len(self._state_list)* in order to avoid state conflict.
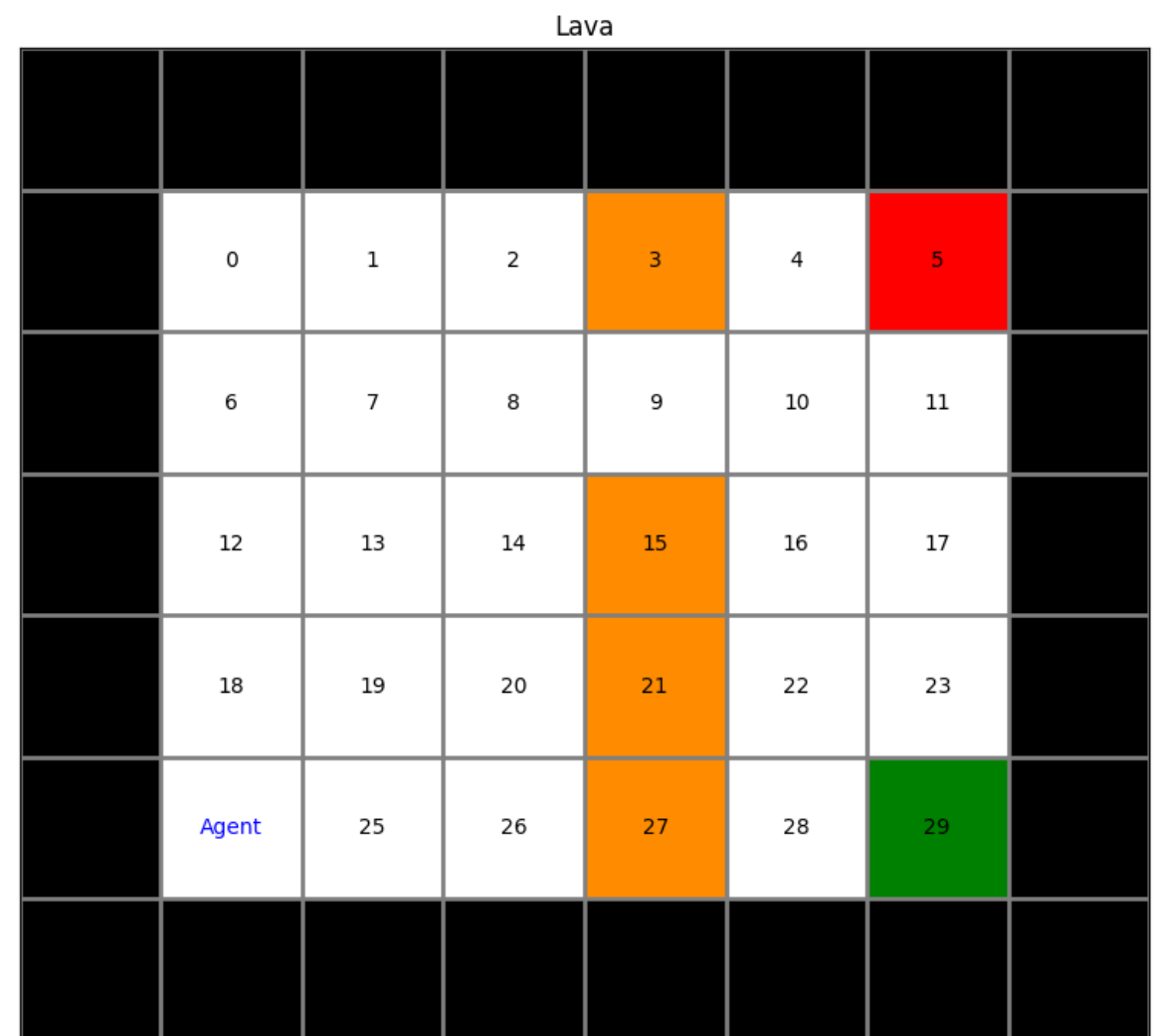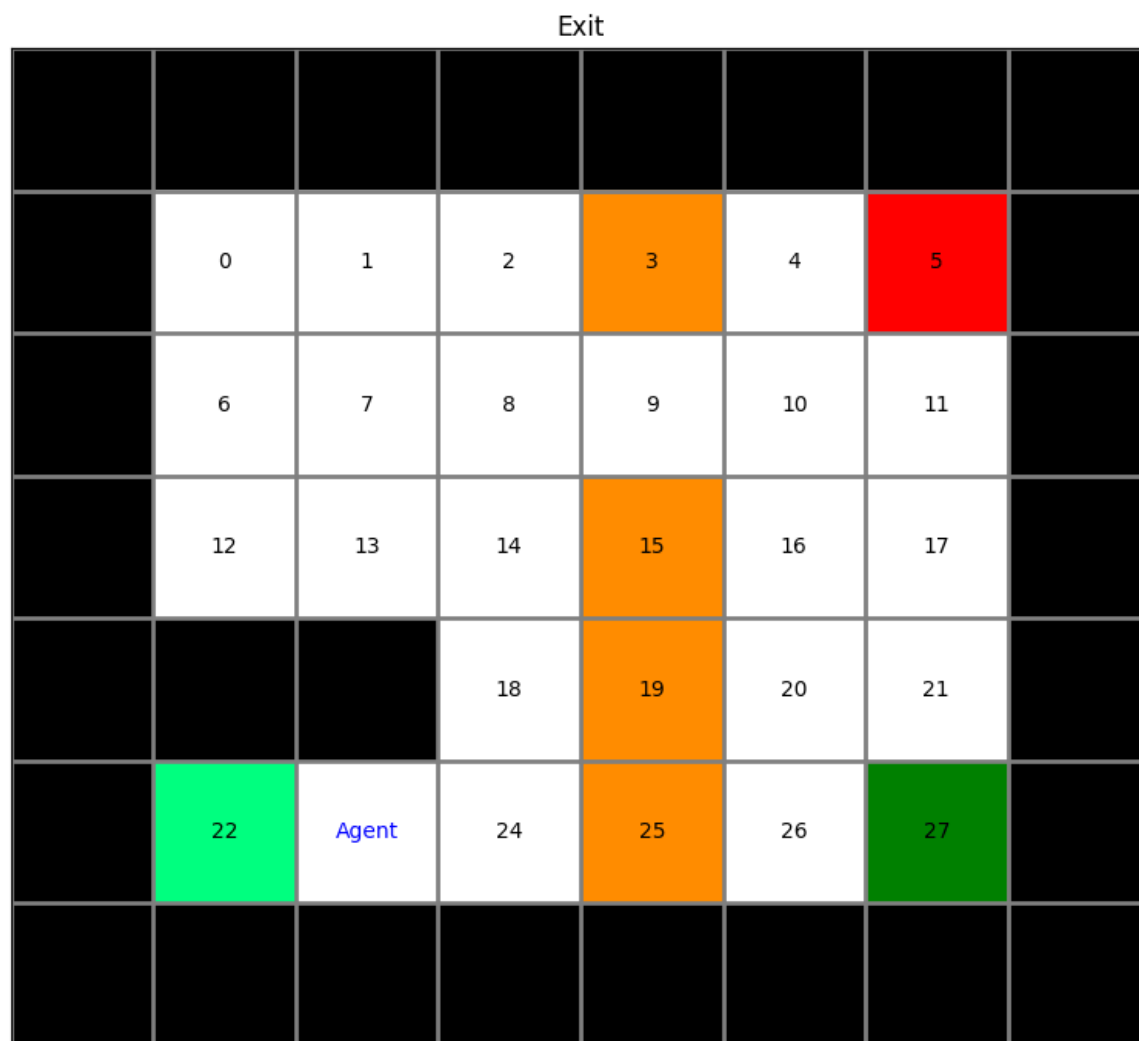
```python
next_state = get_next_state(self._current_state)
if door_is_open:
    next_state += len(self._state_list)
```
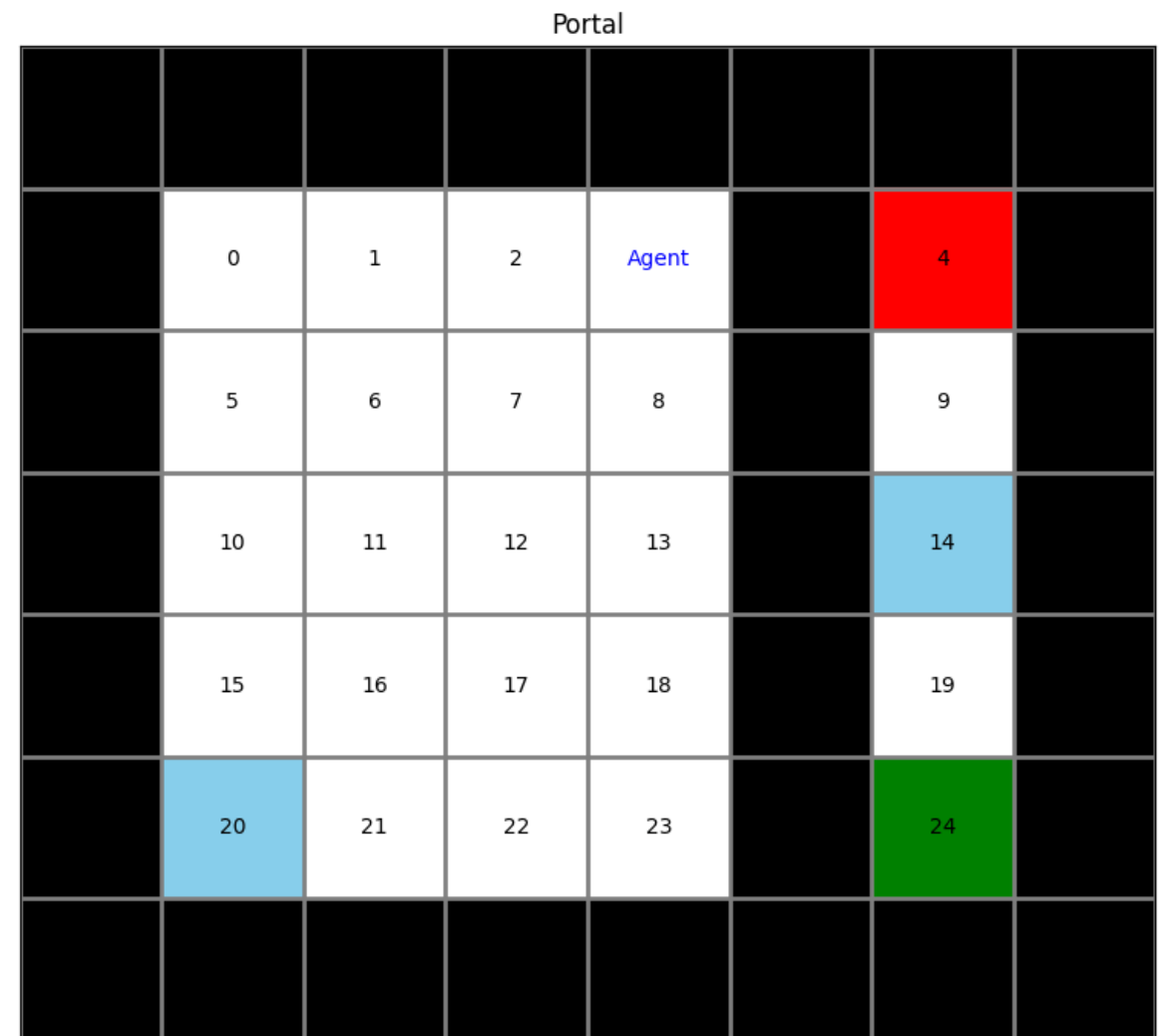
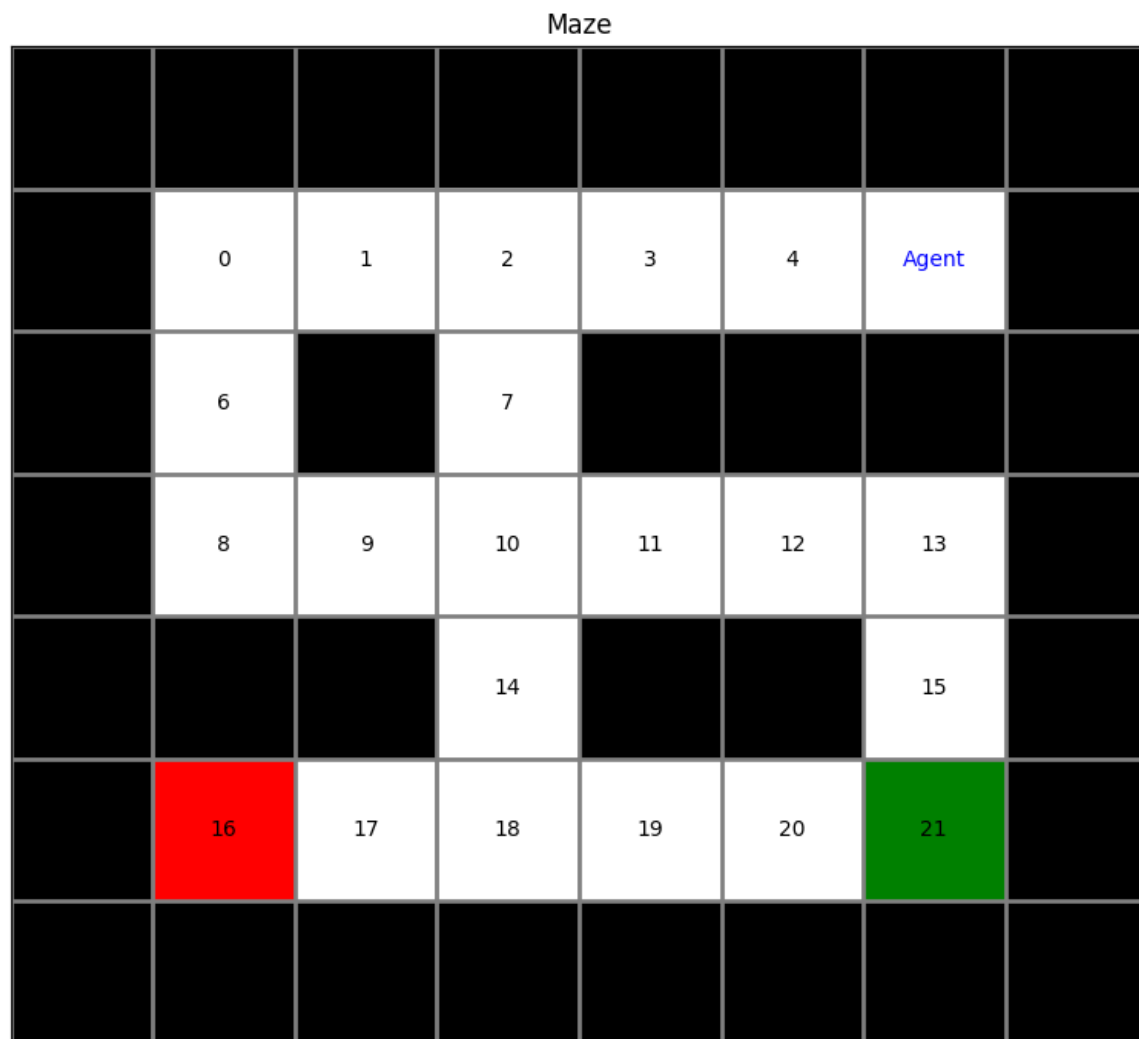# Target task: Bait and Door&Key
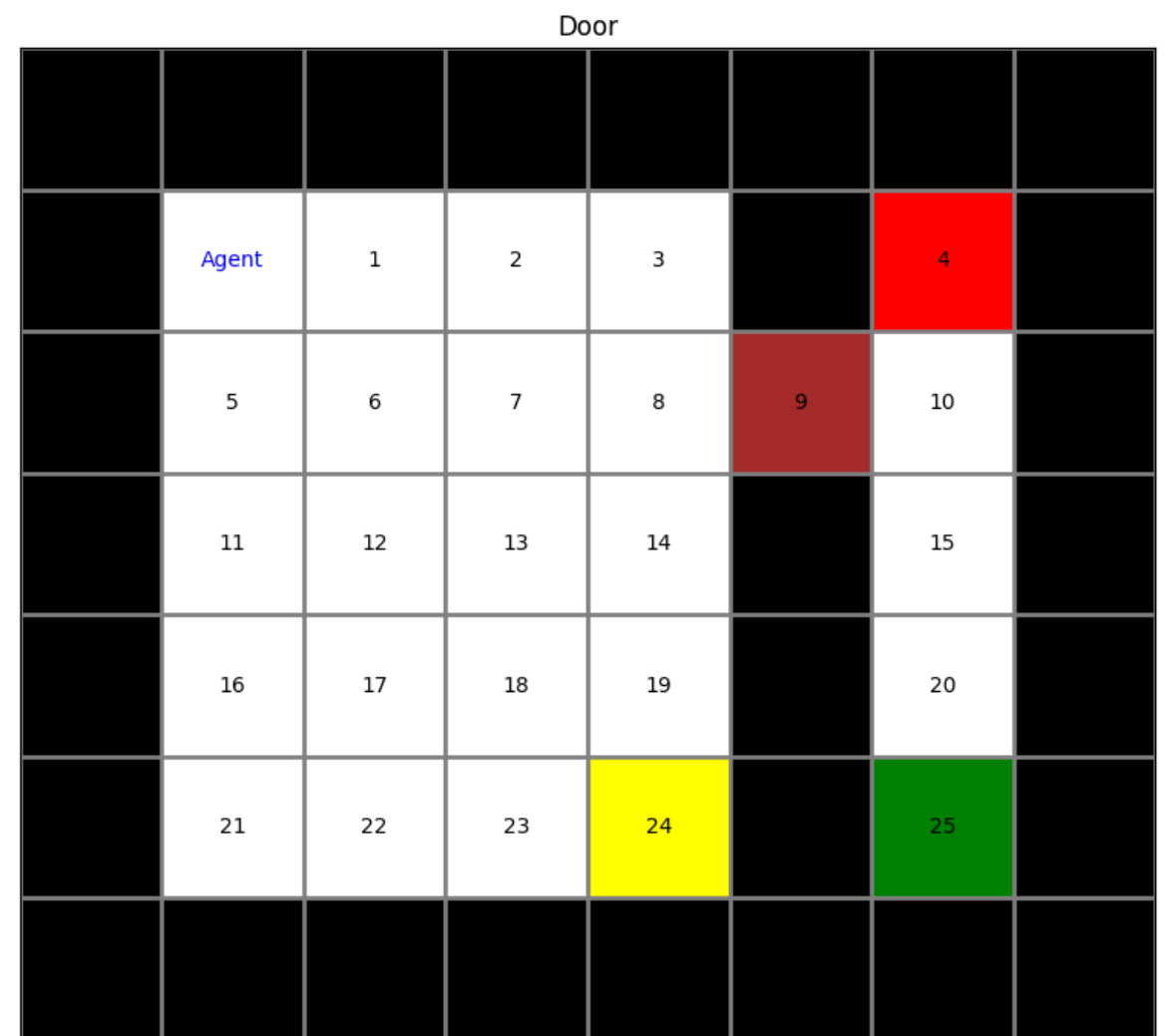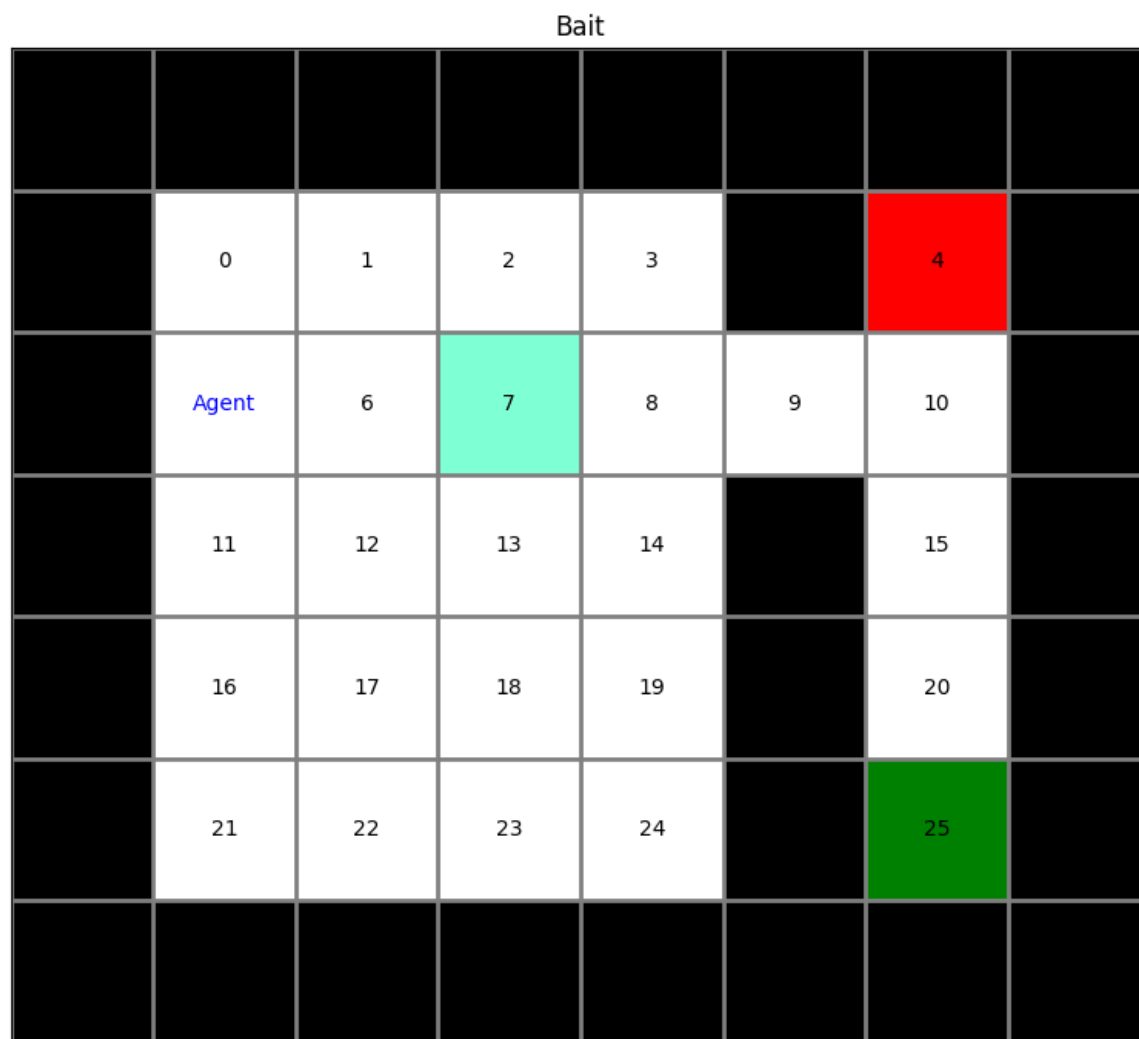
- Entering the Bait state can get a bait reward but will be penalized for every step afterward.
- Going to the Key state can only get a step reward, but the environment will changes.

Target task: Exit and Lava

- The episode ends when entering Lava states.
- You should initial the agent left to the Lava.

Target task: Maze and Portal

- **Hit** the wall at portal can go to the other portal with step reward.

Task2-2048game

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 valid move left.

- [Play 2048](#)
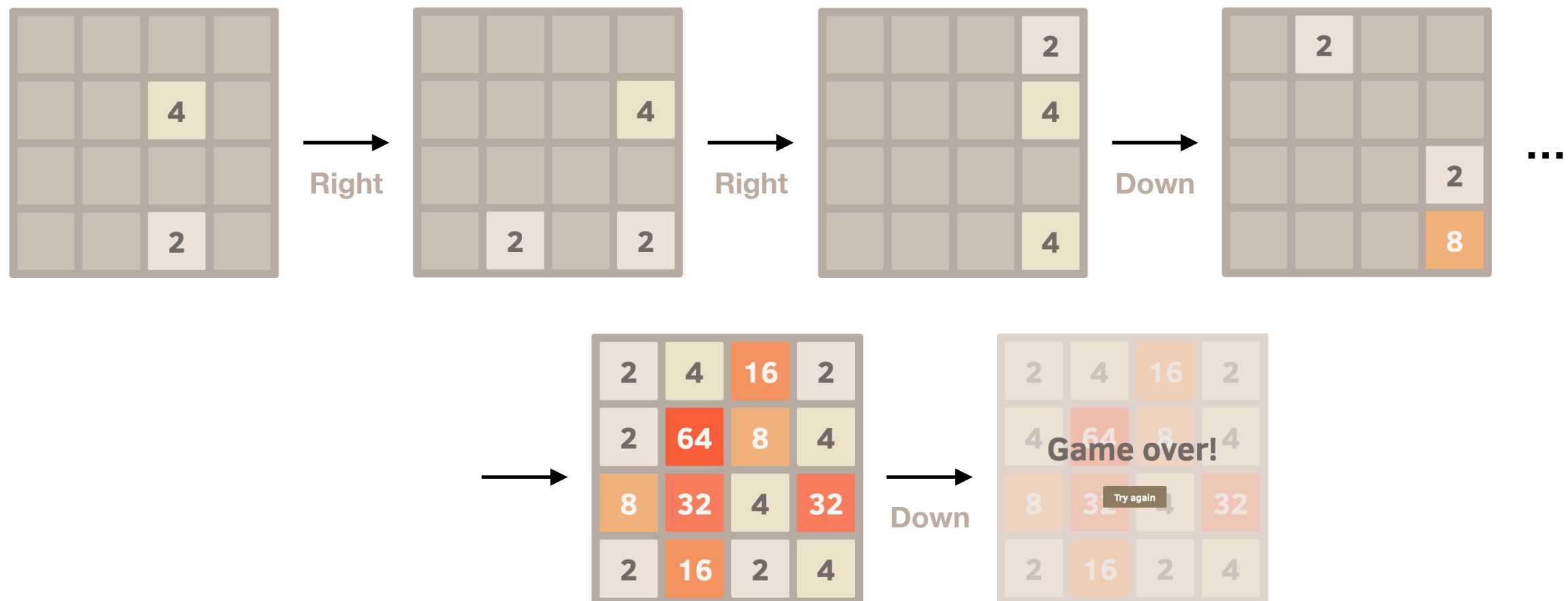
TODO

- For this part of the assignment, you need to train an agent to play 2048 with the Stable Baselines3 package.

- A simulated 2048 environment will be provided, you should:

  - Choose and tune hyperparameters with algorithms from SB3.

  - Modify the environment (Terminal conditions, reward shaping...)

  - Work on other techniques that can help policy perform better.

- We hope this part of HW3 can help you learn how to train and improve your model.

Environment2-2048game

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 valid 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 our evaluation, the episode terminates when the agent executes an illegal move.
     - You can still modify the code to let agent explore other action after illegal moves.

State and Action Space

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

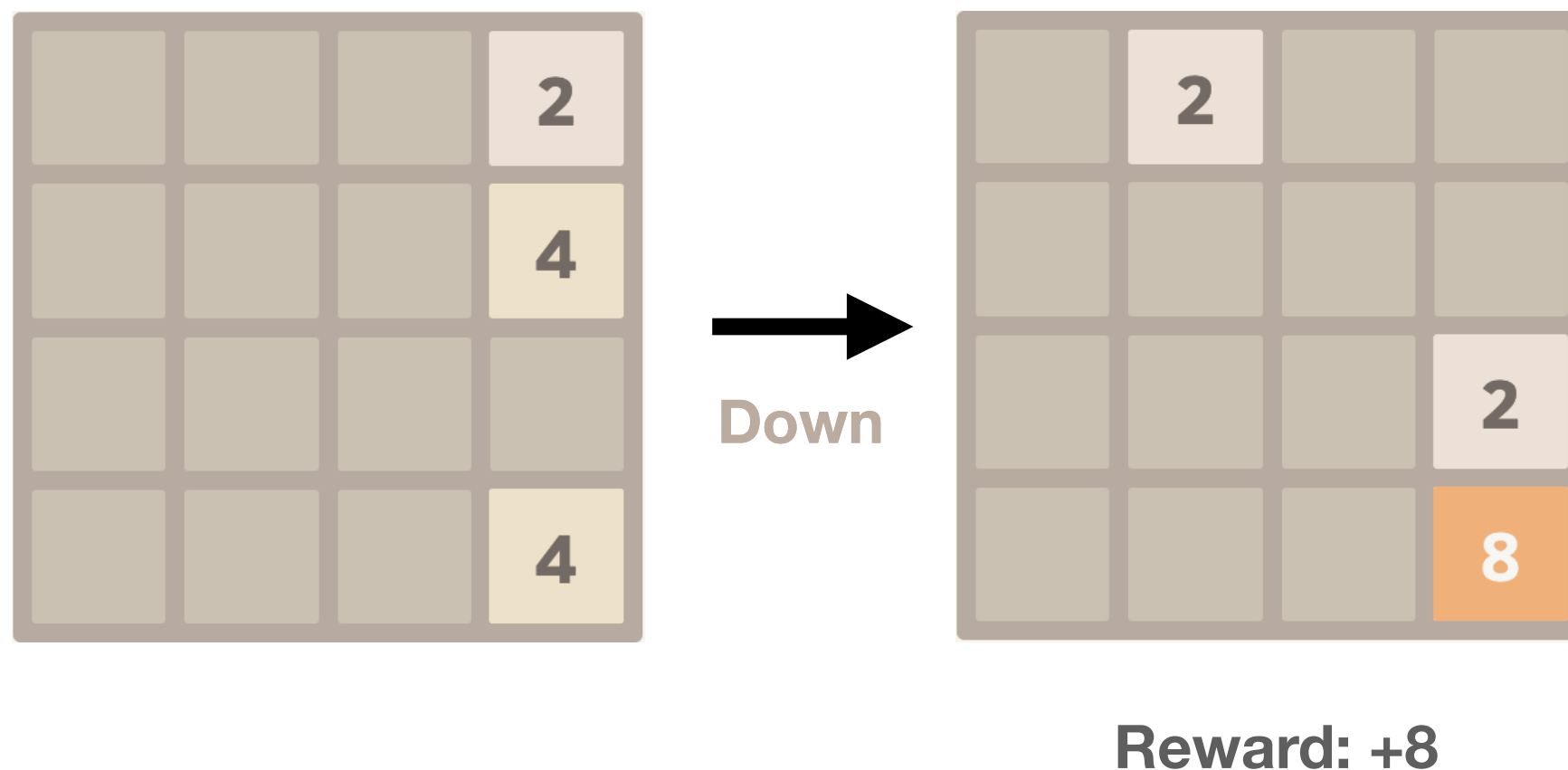- Action: Discrete(4)

Example:
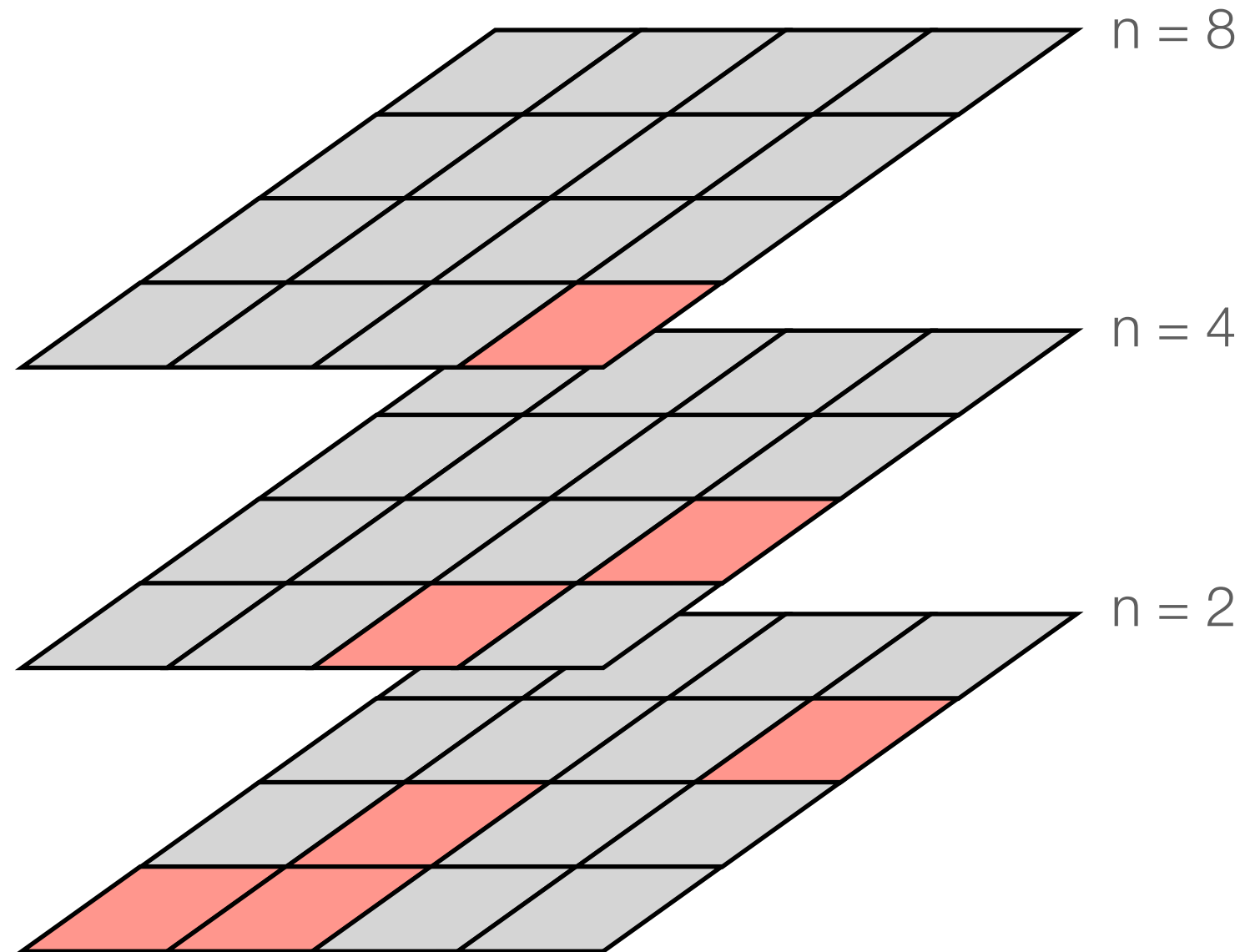
Reward

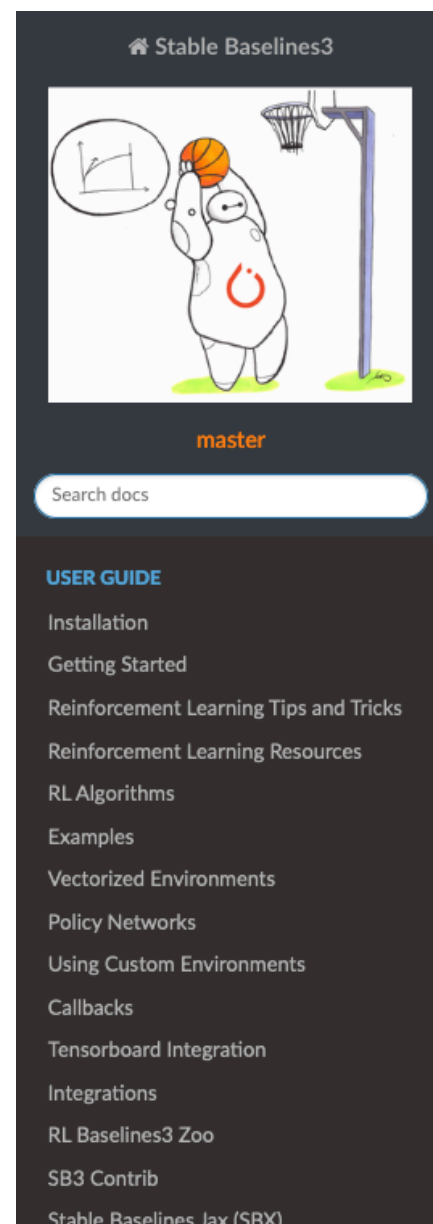- The default reward will be the added score for each step.

- Grades will only consider the highest tile value, so it's okay to modify the reward

Down

Reward: +8

Stable Baselines 3 & Gymnasium

Stable Baselines 3

- Stable Baselines is a package that provides training APIs for RL training.

- Useful for training Gym environment with basic RL algorithms.

- Documentation

- Examples

SB3 - Supported Algorithms

- [DQN](#) (Deep Q Network)

- [DDPG](#) (Deep Deterministic Policy Gradient)

- [TD3](#) (Twin Delayed DDPG)

- [PPO](#) (Proximal Policy Optimization)

- [A2C](#) (Asynchronous Advantage Actor Critic)

- [SAC](#) (Soft Actor Critic)

- 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.

SB3 - Callbacks

- Callback is a set of functions that allows you to execute custom code during training (e.g., logging, saving models, monitoring progress).

- Types of [Callbacks](#)

  - BaseCallback: Custom callbacks should inherit this callback.

  - EvalCallback: Evaluates the model at eval_freq and save the best model.

  - CheckpointCallback: Saves model at save_freq.

  - EventCallback: When an event is triggered, then a child callback is called

- CustomCallback events

  - _on_training_start(): called before first rollout

  - _on_step(): called after each env.step()

  - _on_rollout_end: called before updating the policy

SB3 - Policy Networks
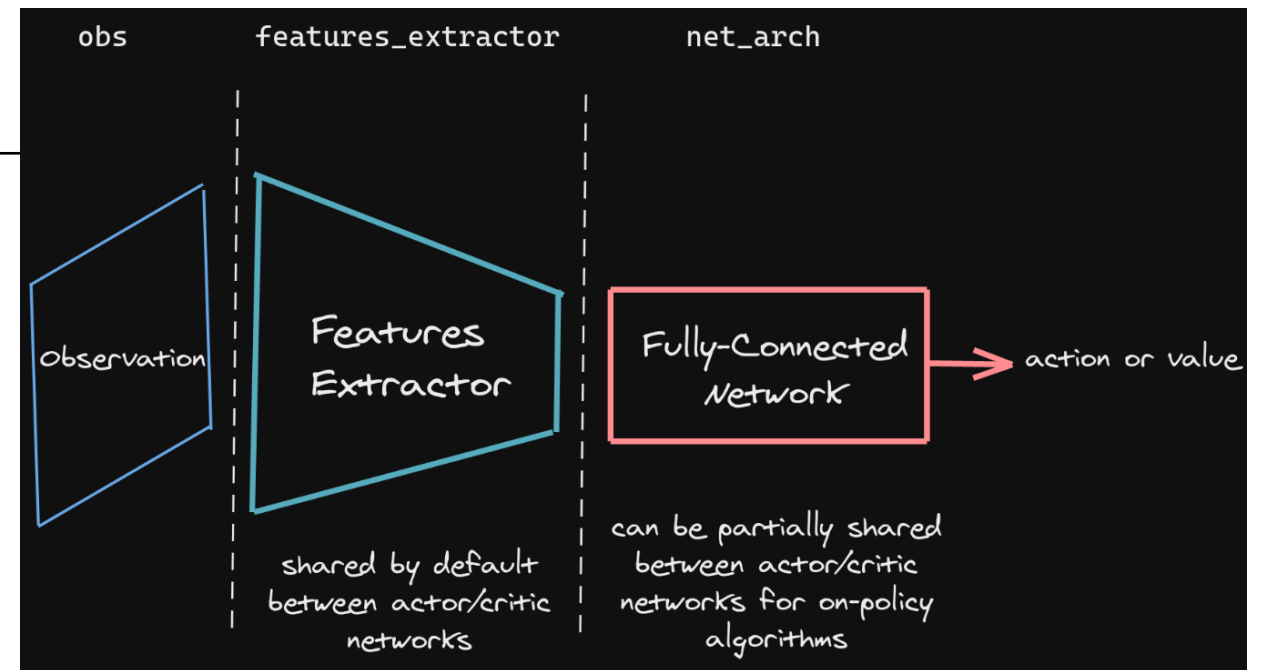
- Types of [policy networks](#)

    - CnnPolicy: For image inputs.

    - MlpPolicy: For other type of inputs.

- Structure of policy networks

    - Feature extractor: you can change the default parameters by passing `features_extractor_kwargs`

      or create custom feature extractor by inherit BaseFeatureExtractor.

    - Full-Connected network: you can change the default parameters by passing `net_arch`

    - Different architectures for actor and critic: `net_arch=dict(pi=[32, 32], vf=[64, 64])`

    - Same architecture for actor and critic: `net_arch=[256, 256]`

SB3 - Vectorized Environments

- [Vectorized Environments](#) are used to stack multiple independent environments into a single environment.

- You can use VecEnv for training and still use the original Env for evaluation.

    - DummyVecEnv: It can stack separate envs and run these envs in sequence. It will combine obs, rewards and dones into a batched output. Best for lightweight env.

    - SubprocVecEnv: It can stack separate envs and run these envs in parallel. Require multiple CPU cores and process management, which will takes extra memory.

Name	Box	Discrete	Dict	Tuple	Multi Processing
DummyVecEnv	✓	✓	✓	✓	✗
SubprocVecEnv	✓	✓	✓	✓	✓

- 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](#))

Environment Packages

- There are many <mark>open-source environment packages</mark> for RL training.

- Examples: "OpenAI Gym", "Deepmind Control", "Nvidia Isaac Gym", etc.

- For this assignment, we will mainly focus on "OpenAI Gym" for its compatibility with "Stable Baselines 3".
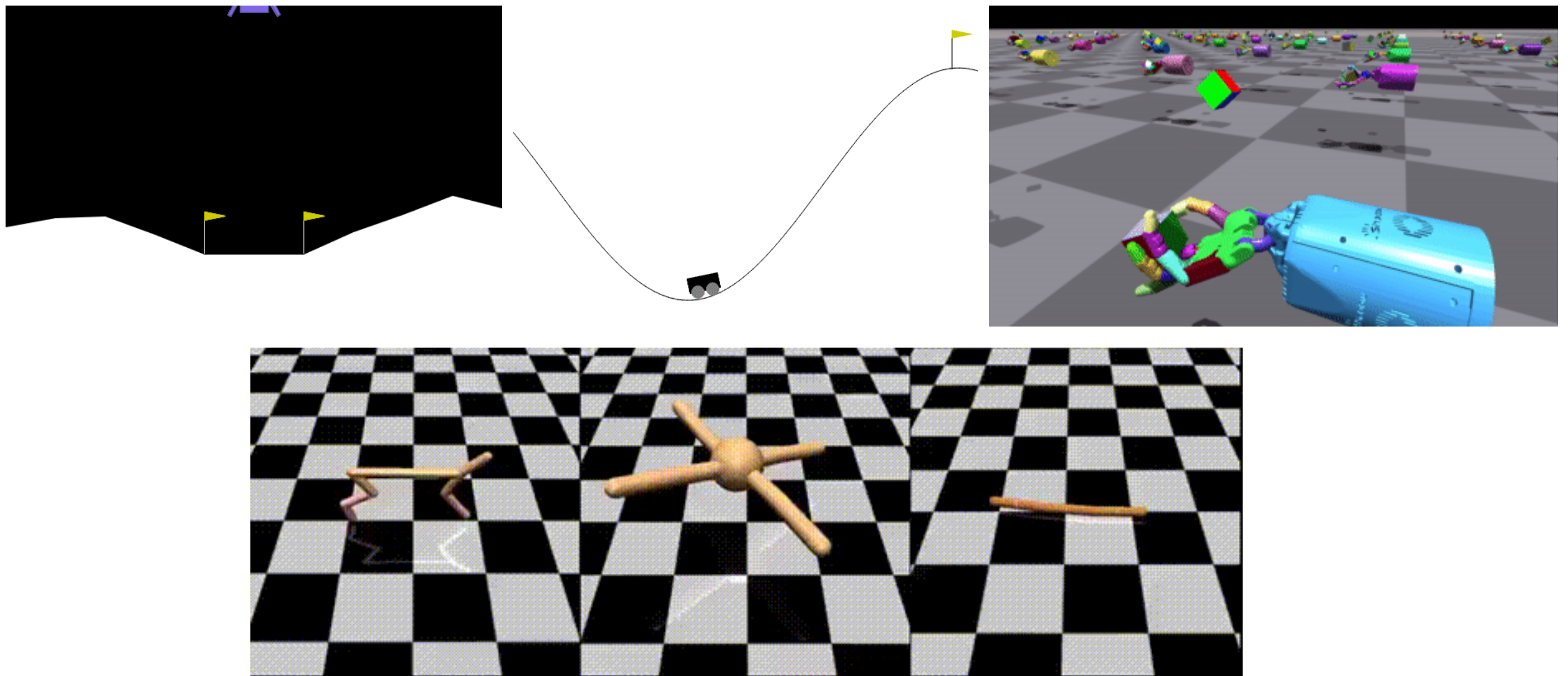
OpenAI Gym / Gymnasium

- 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.
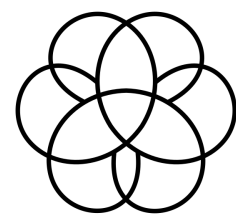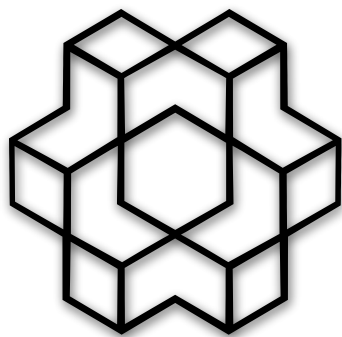
- OpenAI Gym / Gymnasium
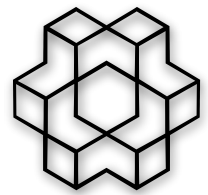
Gymnasium - Spaces

- Box: n dim tensors, range of values. E.g. spaces.Box(0, 1, shape=(3,3))

- Discrete: set of integers from 0 to n-1. E.g. spaces.Discrete(3)

- MultiDiscrete: multiple discrete values. E.g. spaces.MultiDiscrete([5,2,2])

- MultiBinary: N binary values. E.g. spaces.MultiBinary(5)

- Tuple: A combination of different spaces. E.g. spaces.Tuple((spaces.Discrete(2), spaces.Box(0, 1, shape=(2,))))

- Dict: A dictionary of named spaces. E.g. spaces.Dict({"position": spaces.Discrete(3), "velocity": spaces.Box(0, 1, shape=(2,))})

Gymnasium - Spaces
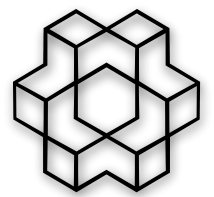
- SB3 algorithms support <mark>different action spaces</mark>.

- Tuple is not supported by any algorithms.

- You can always check Algorithms webpages for more details

Name	Box	Discrete	MultiDiscrete	MultiBinary	Multi Processing
ARS [1]	✓	✓	✗	✗	✓
A2C	✓	✓	✓	✓	✓
DDPG	✓	✗	✗	✗	✓
DQN	✗	✓	✗	✗	✓
HER	✓	✓	✗	✗	✓
PPO	✓	✓	✓	✓	✓
QR-DQN [1]	✗	✓	✗	✗	✓
RecurrentPPO [1]	✓	✓	✓	✓	✓
SAC	✓	✗	✗	✗	✓
TD3	✓	✗	✗	✗	✓

A2C

A synchronous, deterministic variant of Asynchronous Advantage Actor Critic (A3C). It uses multiple workers to avoid the use of a replay buffer.

Notes

- Original paper: https://arxiv.org/abs/1602.01783
- OpenAI blog post: https://openai.com/blog/baselines-acktr-a2c/

Can I use?

- Recurrent policies: ✗
- Multi processing: ✓
- Gym spaces:

Space	Action	Observation
Discrete	✓	✓
Box	✓	✓
MultiDiscrete	✓	✓
MultiBinary	✓	✓
Dict	✗	✓

Gymnasium - Env

- Methods:

  - Env.step(action): Interact with env and return <u>obs, reward, terminated, truncated, info</u> .

  - Env.reset(): Reset env and return <u>obs(init_state), info</u>.

  - Env.render(): Compute render frames as render_mode, like "rgb_array" or "human".

  - Env.close(): Close the env

- Attributes

  - Env.action_space

  - Env.observation_space

  - Env.spec: Environment ID or other metadata like max_episode_steps or entry_point
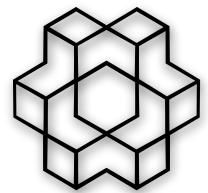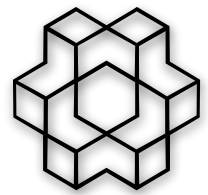
"True" if the episode ends due to termination conditions.

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

Gymnasium - Make Custom Env

- Define the custom Env class

```python
# Step 1: Define the custom environment class
class SimpleGridEnv(gym.Env):
    """Custom Environment that follows gymnasium interface"""

    def __init__(self):
        super(SimpleGridEnv, self).__init__()
```

- Register the custom Env

```python
# Step 2: Register the custom environment
gym.envs.registration.register(
    id='SimpleGrid-v0',  # Unique identifier for your environment
    entry_point=SimpleGridEnv,  # Pointing to the custom environment class
    max_episode_steps=100,  # Optional, maximum number of steps per episode
)
```

- Make Env

```python
# Step 3: Create and test the custom environment
env = gym.make('SimpleGrid-v0')

# Reset the environment and print the initial state
state, _ = env.reset()
print(f"Initial State: {state}")
```

Gymnasium - Tips & Tricks

- Normalize action space and make it symmetric if it is continuous.

- Debug with random actions to check if your env works.

- Done and Truncation should be handle separately

- Try to find existing implementation before coding from scratch

- Validate your method on easier task or env first

```python
from gym import spaces

# Unnormalized actions spaces only works with algorithms
# that don't really directly on a Gaussian to define the policy
# (e.g. DDPG or SAC, where their output is rescaled to fit the action space limits)

# LIMITS TOO BIG: In this case, the sampled actions will only have values around 0
# far away from the limits of the space
action_space = spaces.Box(low=-1000, high=1000, shape=(n_actions,), dtype="float32")

# LIMITS TOO SMALL: In that case, the sampled actions will almost always saturate
# (be greater than the limits)
action_space = spaces.Box(low=-0.02, high=0.02, shape=(n_actions,), dtype="float32")

# BEST PRACTICE: Action space is normalized, symmetric and has an interval range of 2,
# which is usually the same magnitude as the initial standard deviation
# of the Gaussian used to define the policy (e.g. unit initial std in Stable-Baselines)
action_space = spaces.Box(low=-1, high=1, shape=(n_actions,), dtype="float32")
```

Code Structure
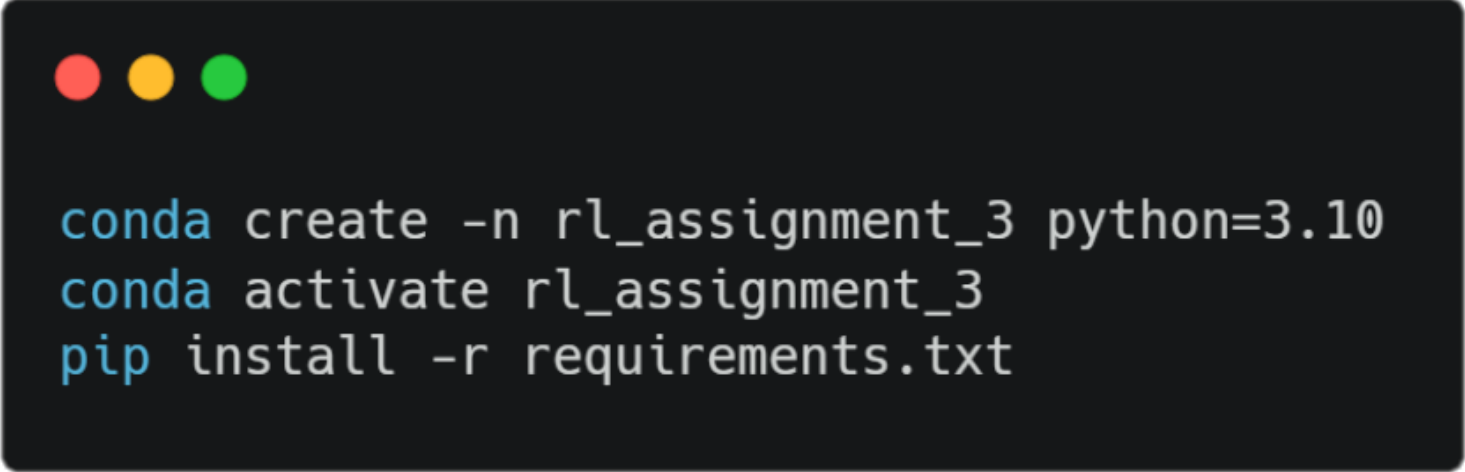
Root directory

- env1_gridworld/

  - The main directory for task1 "GridWorld"

- env2_2048game/

  - The main directory for task2 "2048 the game"

env1_gridworld

requirement.txt

- [Conda](#) / [Miniconda](#)

```
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 assignment_3.

- Packages
  - Packages in requirement.txt will be used in both env1_gridworld and env2_2048game

CONDA

gridworld.py

- class GridWorld

  - TODO: step(action).

  - TODO: reset(). Unlike HW1 and HW2, reset() will not be called inside the class when terminated. We will call reset() from outside when testing correctness.

  - ONLY the return values of step(action) and reset() are used for grading. Feel free to change any other parts except the arguments in __init__().

  - Note: Be careful with similar variables like self.step_reward and self._step_reward.

- class GridWorldEnv (Do Not Modify)

  - It is the wrapper function for the gymnasium. Do not change any part of the class.

main.py

- print_state_action(obs, action, env)

  - Print out state-action pair

- test_correctness(filename)

  - Test the correctness of your environment based on CSV files in grading/

- write_gif(filename, algorithm)

  - Write out a trajectory and save it as gif.

Other folders

- assets/

  - The assets directory contains trained Deep RL algorithms for each task.

- tasks/

  - The tasks directory contains six maps. Our public test cases are trajectories collected in these six maps. Our private test cases will be trajectories collected in maps different from these six maps.

- grading/

  - The grading directory has a list of CSV files. The CSV files contain trajectories collected from six maps in tasks/ directory. Do not modify the CSV files, since they will be used for grading.

- gif/

  - Save the gif generated from write_gif()

env2_2048game

Python Files

- train.py (TODO)

  - Sample code for training your model with SB3.

  - Modify my_config to set hyperparameters and configurations.

- my2048_env.py (TODO)

  - Environment used for training your agent.

  - Feel free to modify any parts of this file and train.py to help you train your agent.

- 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: Feel free to employ any methods to improve your agent, but make sure that we can replicate your results using the train.py and my2048_env.py you provide.

Other folders

- models/

  - Folder where sample models are saved.

- envs/

  - Folder for my2048_env.py, eval2048_env.py and __init__.py for environment registration.

Tips and Hints for 2048

Tips and Hints

- 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)
    - Set learning rate and/or total time step num.
- my2048_env.py
    - Set penalty for illegal moves.
    - Give the agent multiple tries before terminating the episode when the agent executes an illegal move during training.
    - Set weights to give additional reward for certain board patterns that helps the agent learn a certain strategy.

Grading

Grading env1-gridworld

- Correctness (24%)

  - Public Test cases (2% x 6 public cases)

  - Private Test cases (2% x 6 private cases)

  - Correctness less than 100% will not get any credit

```
The correctness of task lava: 100.0 %
The correctness of task exit: 100.0 %
The correctness of task bait: 100.0 %
The correctness of task door: 100.0 %
The correctness of task portal: 100.0 %
The correctness of task maze: 100.0 %
```

Criteria env1-gridworld

- Test cases

  - Only check the output of step and reward function.

  - **5 minutes** for each test case to avoid infinite loops.

  - The private tasks might be more complex than the sample tasks.

  - The testing sequence of correctness might be pretty long.

  - The truncation will not be graded for correctness.

- Sample agents trained on correct dynamics are provided for reference.

  - The trained policy may not be optimal.

Grading env2-2048game

- Baselines (40%)

  - Simple baseline: Reach 128 for at least one rollout. (Public: 5%, Private: 5%)

  - Medium baseline: Reach 256 for at least one rollout. (Public: 5%, Private: 5%)

  - Strong baseline: Reach 512 for at least one rollout. (Public: 5%, Private: 5%)

  - Boss baseline: Reach 1024 or higher for at least one rollout. (Public: 5%, Private: 5%)

- Report (36%)

  - 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 env2-2048game

- Test cases:

  - We will run your model on seeded environments and check the best tile value ("<u>Highest</u>") for baseline evaluations.

  - Public test cases: 100 rollouts, resetting the environment with 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 limit **10 minute** for each case to avoid infinite loops

- 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 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) (4%)

- Q3. Describe what you have done to train your best model? (10%)

- Q4. Choose an environment from the Gymnasium library and train an agent. (10%)

  - Show your results (Screenshot of environment, learning curve…)

  - Share anything you find interesting or difficult.

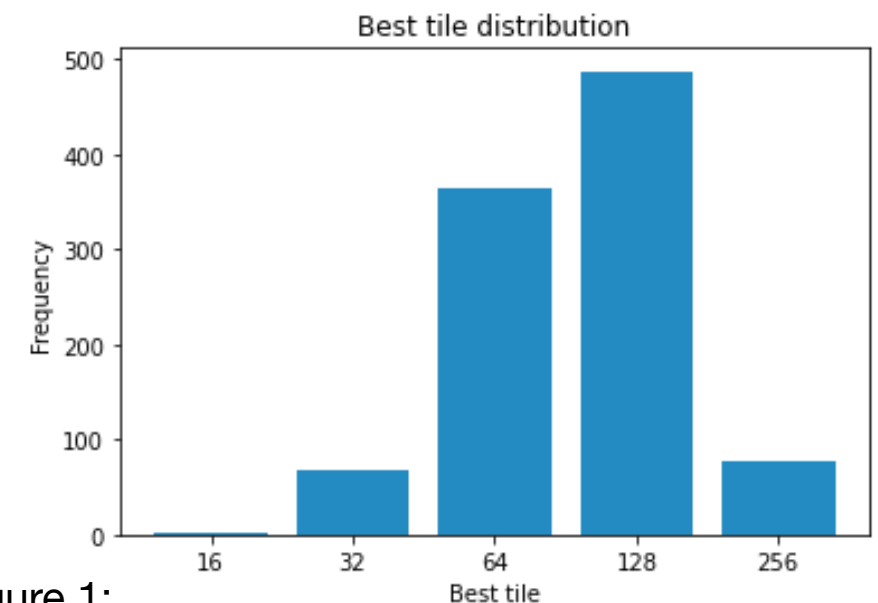- Use our Overleaf template and save in PDF format.

  - Overleaf template

Figure 1:

Policy

Policy

Package

- You can use any Python standard library. (e.g., heap, queue…)

- Don't print anything and don't use ipdb or pdb to set trace. There will be a 10-point deduction for doing it

- System level packages are prohibited (e.g., sys, os, multiprocess, subprocess, shutil…). Importing any one of them will get you a score of 0 for this assignment.

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.

Submission

Submission

- Submit on NTU COOL with following zip file structure:

  - gridworld.py: Containing your implementation for env1-GridWorld.

  - train.py, my2048_env.py: Containing your implementation for env2-2048game.

  - model.zip: Saved 2048 model that can be loaded with eval.py.

  - Get rid of pycache, DS_Store, etc. (Use terminal to zip the file)

  - Student ID with lower case

  - 10-point deduction for wrong format & printing

  - 0 score for importing system level packages

📄 b09901171.zip ➡️ 📂 **b09901171**
                              ├── gridworld.py
                              ├── train.py
                              ├── my2048_env.py
                              ├── model.zip
                              └── report.pdf

- Deadline 2024/11/07 Thu 09:30am

- 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 Huai-Chih Wang**