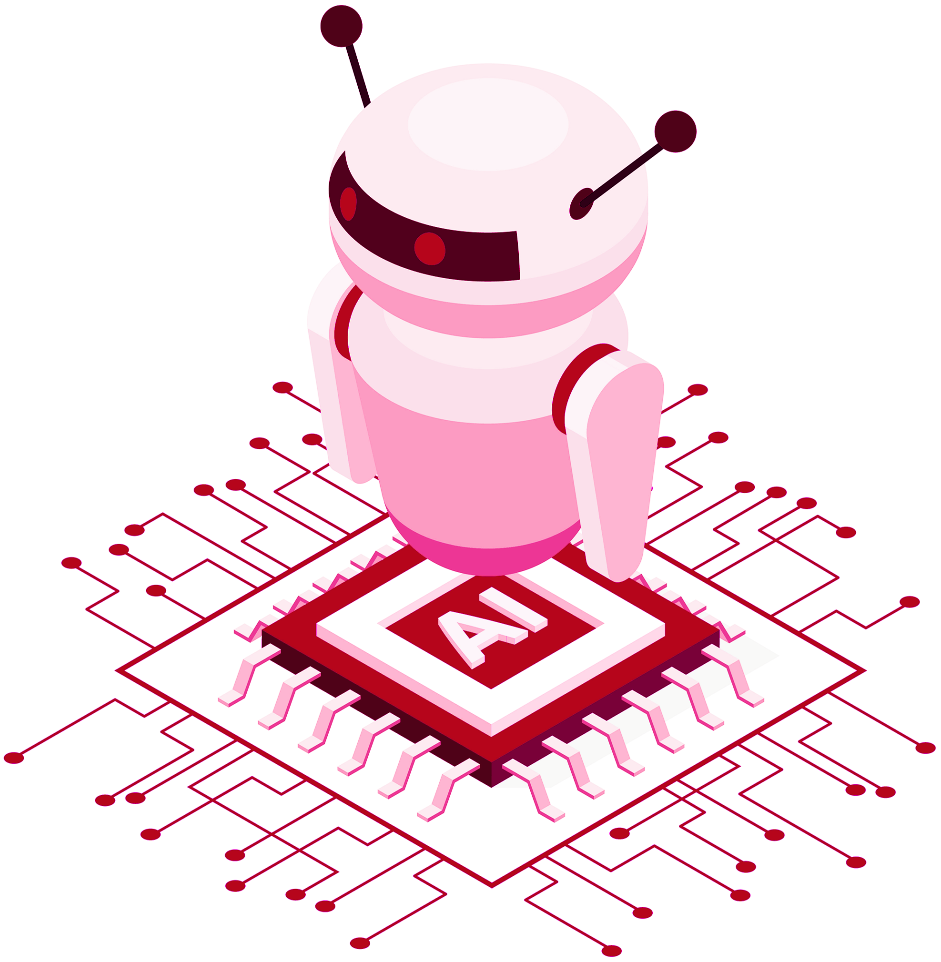


Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 1:

Introduction to RL

By:

[Ashkan Majidi]

[400109984]

Spring 2025

Contents

1 Task 1: Solving Predefined Environments [45-points] 1

2 Task 2: Creating Custom Environments [45-points] 3

3 Task 3: Pygame for RL environment [20-points] 4

Grading

The grading will be based on the following criteria, with a total of 100 points:

Task	**Points**
Task 1: Solving Predefined Environments	45
Task 2: Creating Custom Environments	45
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1: Writing a wrapper for a known env	10
Bonus 2: Implementing pygame env	20
Bonus 3: Writing your report in Latex	10

Notes:

- Include well-commented code and relevant plots in your notebook.
- Clearly present all comparisons and analyses in your report.
- Ensure reproducibility by specifying all dependencies and configurations.

1 Task 1: Solving Predefined Environments [45-points]

In this problem we have to solve two game environments in the gymnasium library. I choose *FrozenLake* and *Taxi*. FrozenLake is an environment consisting a grid world, a human, some possible obstacles and a desired cell and goal is to reach that desired cell. Taxi is an environment with four designated pick-up and drop-off locations (Red, Green, Yellow and Blue) in the 5x5 grid world. The goal is move the taxi to the passenger's location, pick up the passenger, move to the passenger's desired destination, and drop off the passenger.

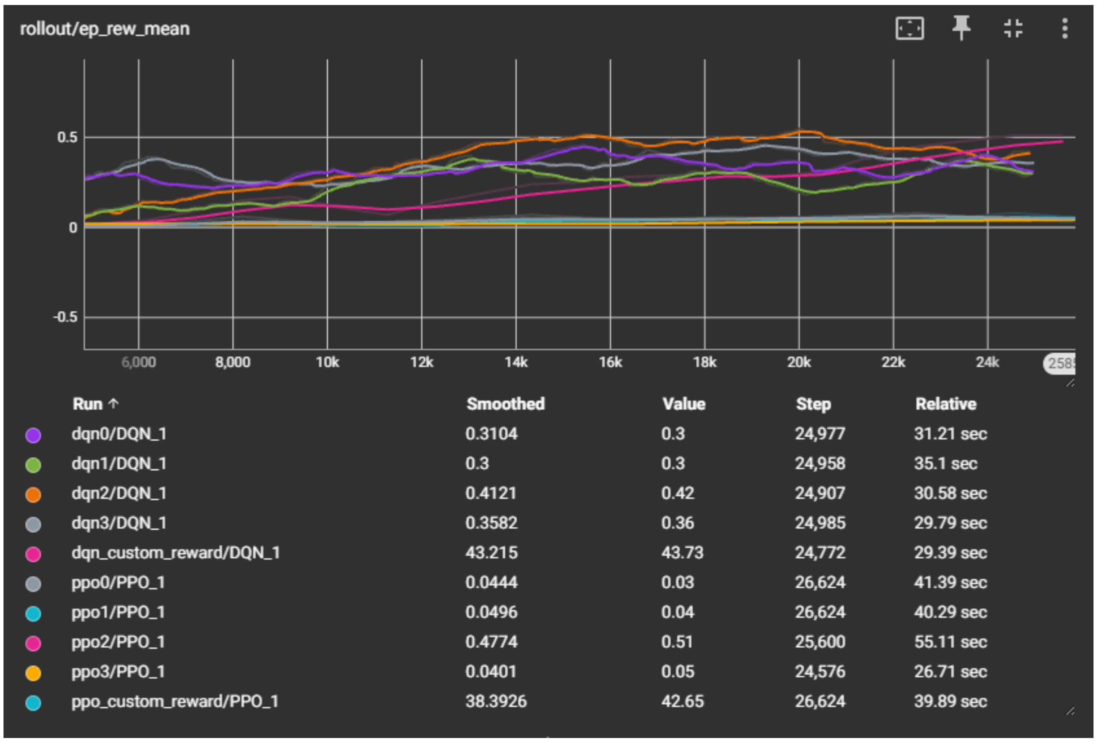
To implement this we use the pre-defined environments in gymnasium library as you see in below:

```
env1 = gym.make("FrozenLake-v1", is_slippery=True)
env2 = make_vec_env("Taxi-v3", n_envs=1)
```

Now we should optimize the policy using different methods and we use **PPO** and **DQN**. This methods are implemented in stable_baselines3.

```
from stable_baselines3 import PPO, DQN
```

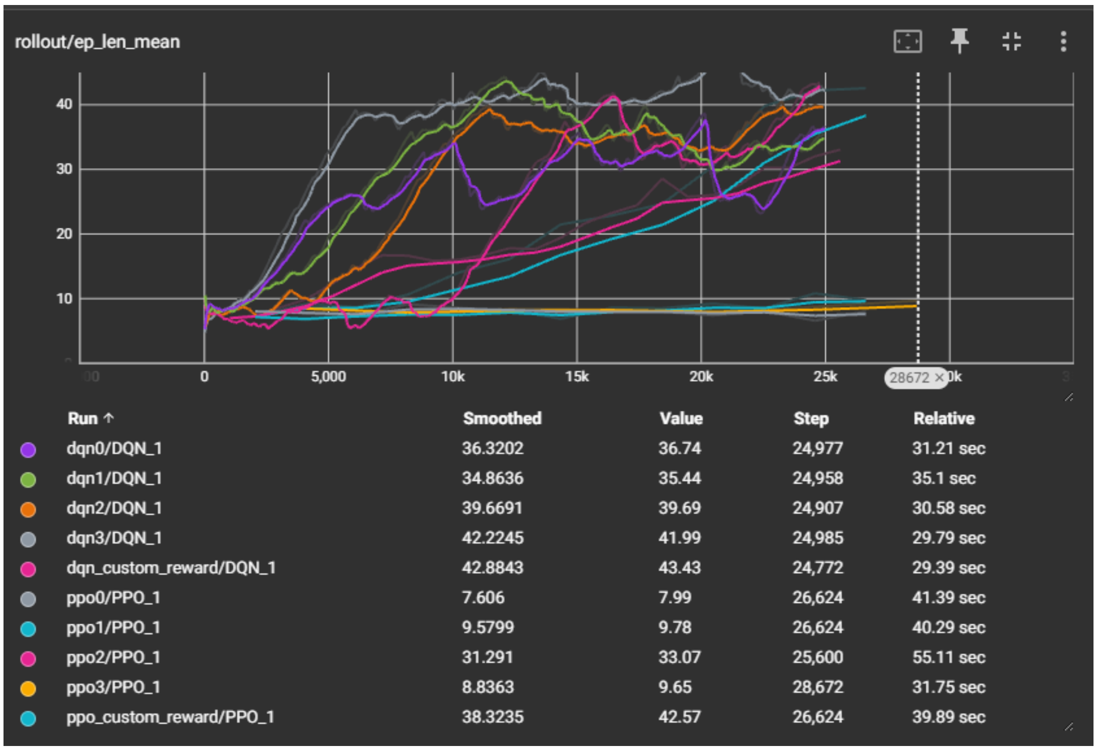
Also we run this training process using different hyperparameters and plot their return values in tensorboard. our first hyper parameters are default ones and others vary in batch size, learning rete, etc. Additionally we have a wrapper that modifies the rewards to see changes in them. Following is plot of mean returns and mean length of mentioned algorithms.



	Run ↑	Smoothed	Value	Step	Relative
●	dqn0/DQN_1	0.3104	0.3	24,977	31.21 sec
●	dqn1/DQN_1	0.3	0.3	24,958	35.1 sec
●	dqn2/DQN_1	0.4121	0.42	24,907	30.58 sec
●	dqn3/DQN_1	0.3582	0.36	24,985	29.79 sec
●	dqn_custom_reward/DQN_1	43.215	43.73	24,772	29.39 sec
●	ppo0/PPO_1	0.0444	0.03	26,624	41.39 sec
●	ppo1/PPO_1	0.0496	0.04	26,624	40.29 sec
●	ppo2/PPO_1	0.4774	0.51	25,600	55.11 sec
●	ppo3/PPO_1	0.0401	0.05	24,576	26.71 sec
●	ppo_custom_reward/PPO_1	38.3926	42.65	26,624	39.89 sec

Figure 1: mean reward of episodes of FrozenLake environment

Note that two custom rewards are not in image because they reward is much higher than others and their graph are upper than them. also you can see DQN algorithms are outperforming PPOs. Additionally, note that ppo2 at the end is best policy and it shows importance of hyperparameter tunning.



	Run ↑	Smoothed	Value	Step	Relative
●	dqn0/DQN_1	36.3202	36.74	24,977	31.21 sec
●	dqn1/DQN_1	34.8636	35.44	24,958	35.1 sec
●	dqn2/DQN_1	39.6691	39.69	24,907	30.58 sec
●	dqn3/DQN_1	42.2245	41.99	24,985	29.79 sec
●	dqn_custom_reward/DQN_1	42.8843	43.43	24,772	29.39 sec
●	ppo0/PPO_1	7.606	7.99	26,624	41.39 sec
●	ppo1/PPO_1	9.5799	9.78	26,624	40.29 sec
●	ppo2/PPO_1	31.291	33.07	25,600	55.11 sec
●	ppo3/PPO_1	8.8363	9.65	28,672	31.75 sec
●	ppo_custom_reward/PPO_1	38.3235	42.57	26,624	39.89 sec

Figure 2: mean lenght of episodes of FrozenLake environment

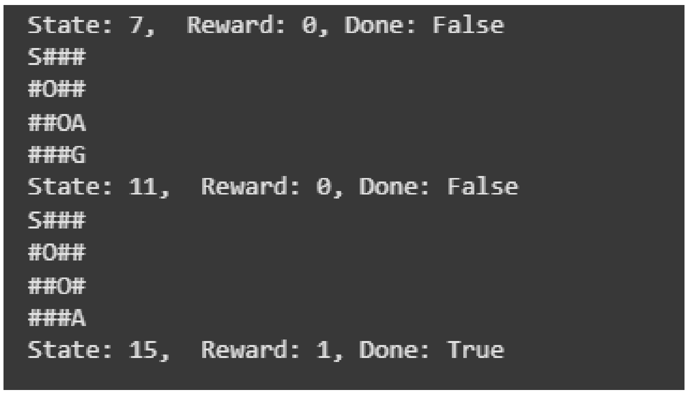
In contrast you see the episode leghts of DQNs are much higher than PPOs. the trend for Taxi environment is almost same with this diffrence that the episode lenghts are not much dirfference.

2 Task 2: Creating Custom Environments [45-points]

In this part we Implement a custom environment. As you can see in this custom environment our start pos is $(0, 0)$ and our goal is $(3, 3)$ and we have two obstacles between them so we define these in our environment.

As you know we have four direction to move. In step function of our class we should take care to not go out of the grid also reward modeling is simple and we just give reward when we reach the goal.

We train our agent with different policy optimization methods and following picture is outcome policy of PPO and as you can see it's propperly reaching the goal.



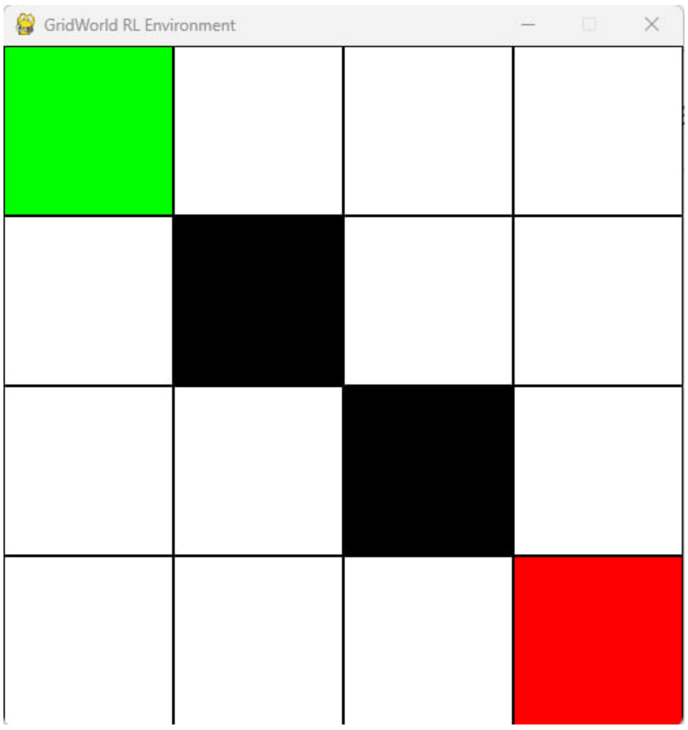
```
State: 7,  Reward: 0, Done: False
S###
#O##
##OA
###G
State: 11,  Reward: 0, Done: False
S###
#O##
##O#
###A
State: 15,  Reward: 1, Done: True
```

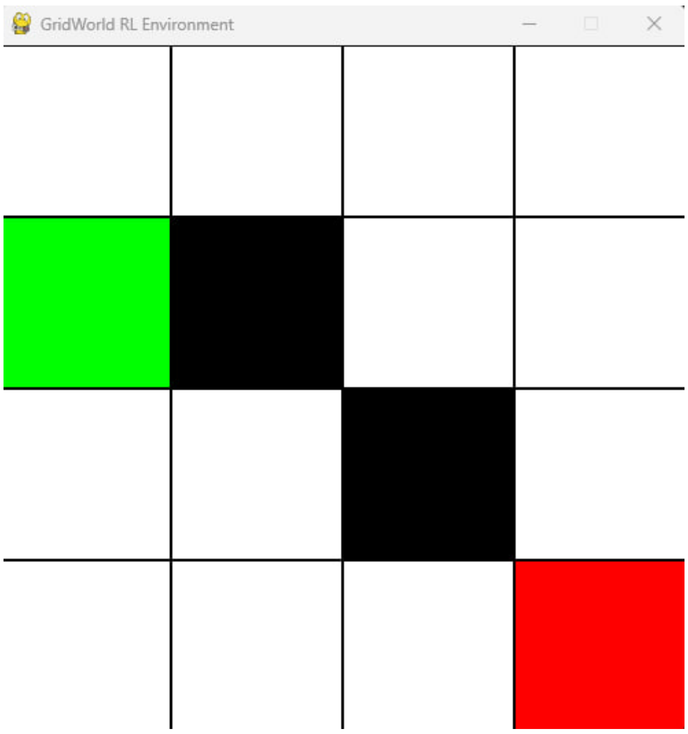
Figure 3: GridWorld agent reaching the goal

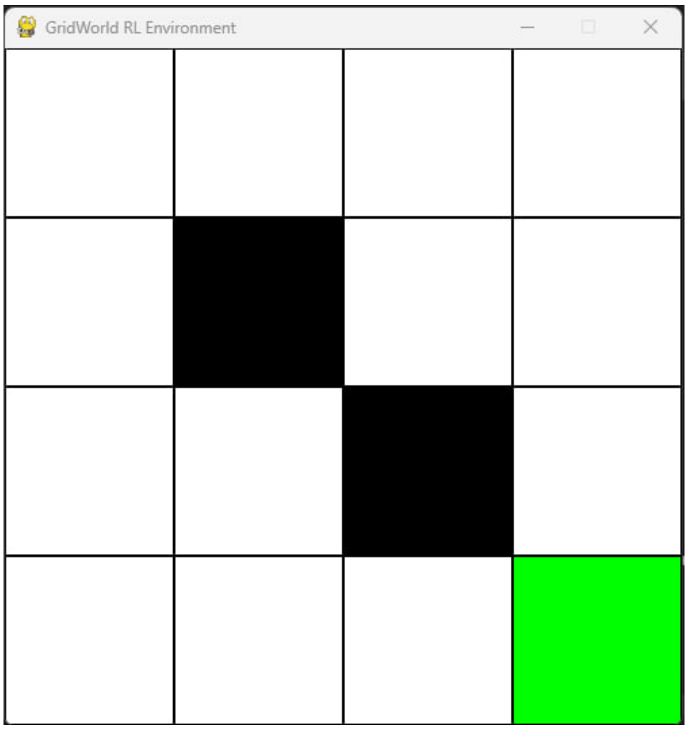
3 Task 3: Pygame for RL environment [20-points]

In this section we implement the previous custom environment in Task 2 with Pygame to have it visualized. Our parameters and train procedure is exactly same as before but we render every step we take after training and visualize it.

Here are some steps until our agent reaches the goal.







References

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd Edition, 2020. Available online: http://incompleteideas.net/book/the-book-2nd.html

[2] A. Raffin et al., "Stable Baselines3: Reliable Reinforcement Learning Implementations," GitHub Repository, 2020. Available: https://github.com/DLR-RM/stable-baselines3.

[3] Gymnasium Documentation. Available: https://gymnasium.farama.org/.

[4] Pygame Documentation. Available: https://www.pygame.org/docs/.

[5] CS 285: Deep Reinforcement Learning, UC Berkeley, Pieter Abbeel. Course material available: http://rail.eecs.berkeley.edu/deeprlcourse/.

[6] Cover image designed by freepik