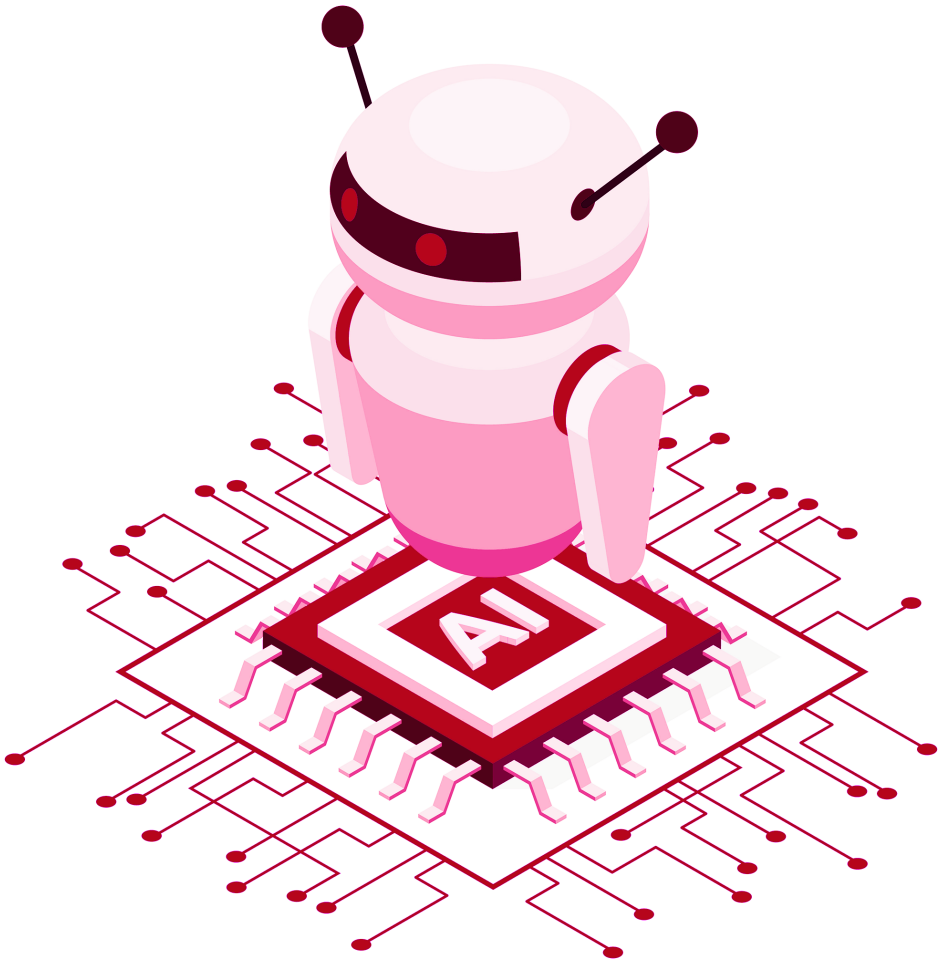


Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 1:

Introduction to RL

By:

[Ali Ghasemzadeh]

[401106339]

RIML

Deep RL Course

Spring 2025

Contents

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]

You can download the code from this part it has a big size so I upload it in google drive and give the link here : my code

I first define the FrozenLake-v1 as the first env and check with the parameters below :

- PPO_1 : default amounts
- PPO_2 : learning rate = 0.01, gamma = 0.99
- PPO_3 : learning rate = 0.01, gamma = 0.8
- PPO_4 : learning rate = 0.0001, gamma = 0.99
- PPO_5 : learning rate = 0.0001, gamma = 0.8
- PPO_6 : default amounts but using wrapper for reward
- DQN_1 : default amounts
- DQN_2 : learning rate = 0.01, gamma = 0.99
- DQN_3 : learning rate = 0.01, gamma = 0.8
- DQN_4 : learning rate = 0.0001, gamma = 0.99
- DQN_5 : learning rate = 0.0001, gamma = 0.8
- DQN_6 : default amounts but using wrapper for reward

We change the parameters learning_rate and gamma, there are some other parameters and we can change them also.

If we define a big lr convergence disturbed and overshoot so choosing learning rate is too important because it controls how parameters updates. If we choose small gamma it doesn't pay attention to the future rewards so it plays bad.
In the second part we define the wrapper for the change of the reward function and for the zero rewards we define the -0.01 then the game should win fast.
At the end we use tensorboard to see the output and convergences and other information.
From what I observed with a big gamma and a small learning rate we can get better results and convergence, we have also use the default amounts and see the results and because it has small lr and big gamma it converges, These results happen for both algorithms.

Then I define the CartPole-v1 as the second env and check with the parameters below :

- PPO_1 : default amounts
- PPO_2 : learning rate = 0.01, gamma = 0.99
- PPO_3 : learning rate = 0.01, gamma = 0.8
- PPO_4 : learning rate = 0.0001, gamma = 0.99
- PPO_5 : learning rate = 0.0001, gamma = 0.8
- PPO_6 : default amounts but using wrapper for reward

- DQN_1 : default amounts
- DQN_2 : learning rate = 0.01, gamma = 0.99
- DQN_3 : learning rate = 0.01, gamma = 0.8
- DQN_4 : learning rate = 0.0001, gamma = 0.99
- DQN_5 : learning rate = 0.0001, gamma = 0.8
- DQN_6 : default amounts but using wrapper for reward

we have the last results for this part too but this game is harder and we need more time_steps to learn it and convergence. plots and outputs :

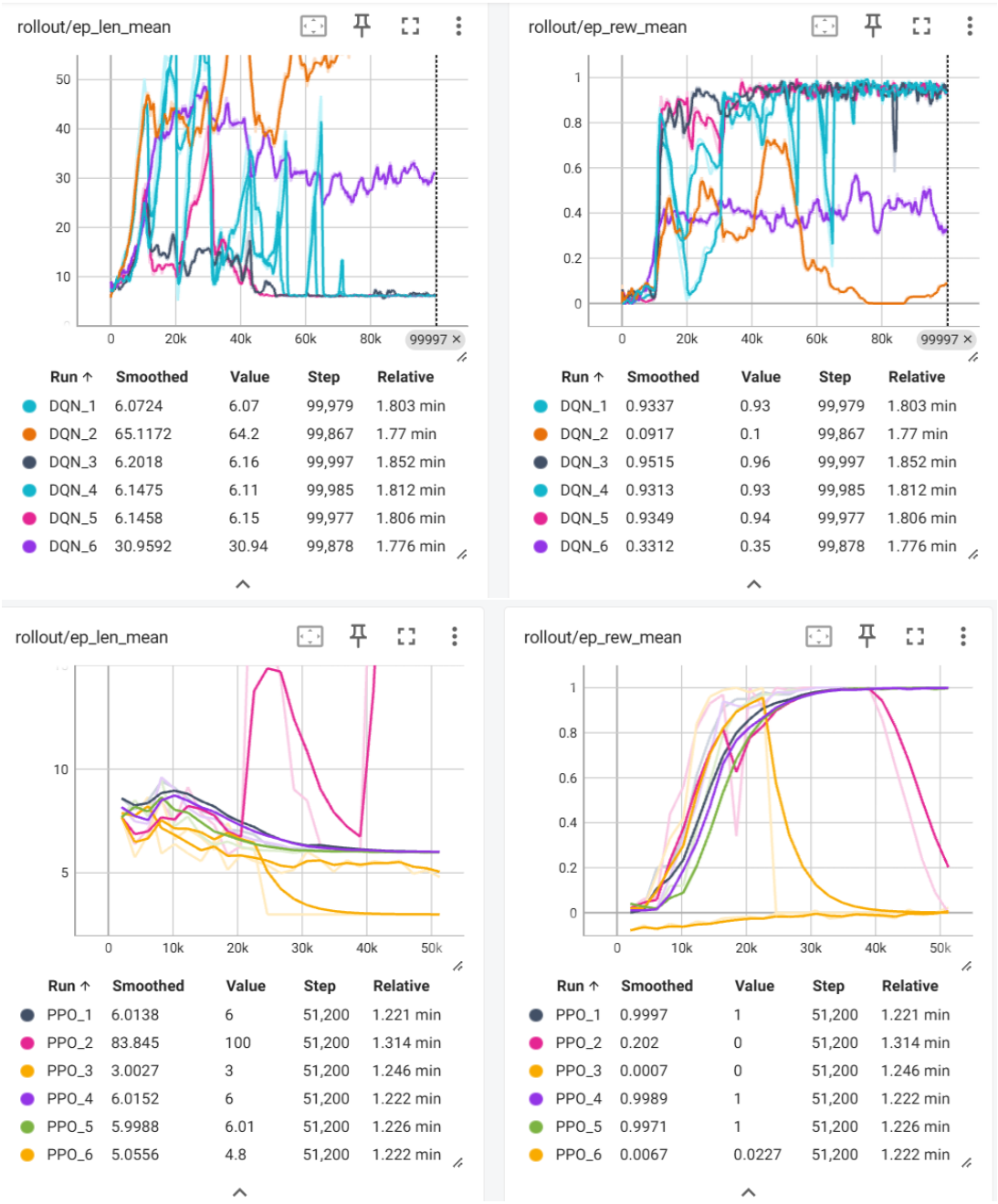


Figure 1: frozen ppo, dqn

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

We define our model and custom env if we go to the obstacles we get negative reward of -5 and if we reach the goal we get +100, for other parts we get reward 0.

- PPO_1 : default amounts
- PPO_2 : learning rate = 0.01, gamma = 0.99
- PPO_3 : learning rate = 0.01, gamma = 0.8
- PPO_4 : learning rate = 0.0001, gamma = 0.99
- PPO_5 : learning rate = 0.0001, gamma = 0.8
- DQN_1 : default amounts
- DQN_2 : learning rate = 0.01, gamma = 0.99
- DQN_3 : learning rate = 0.01, gamma = 0.8
- DQN_4 : learning rate = 0.0001, gamma = 0.99
- DQN_5 : learning rate = 0.0001, gamma = 0.8

In this part we have our custom env and as we can see almost all of the PPO's converge because the game is easy and with different parameters it can learn fast but PPO_2 converge a bit better than others. In the DQN we have some overshooting and convergence isn't smooth and DQN_3 converge better than other DQN's.(by convergence I means convergence of mean-reward)
If I want to explain more about the env and implementation : we have action space left, right, up, down and if we go up and we can't go up means if we are at the top of the grid world we return to the first pos. In the render we print the grid.

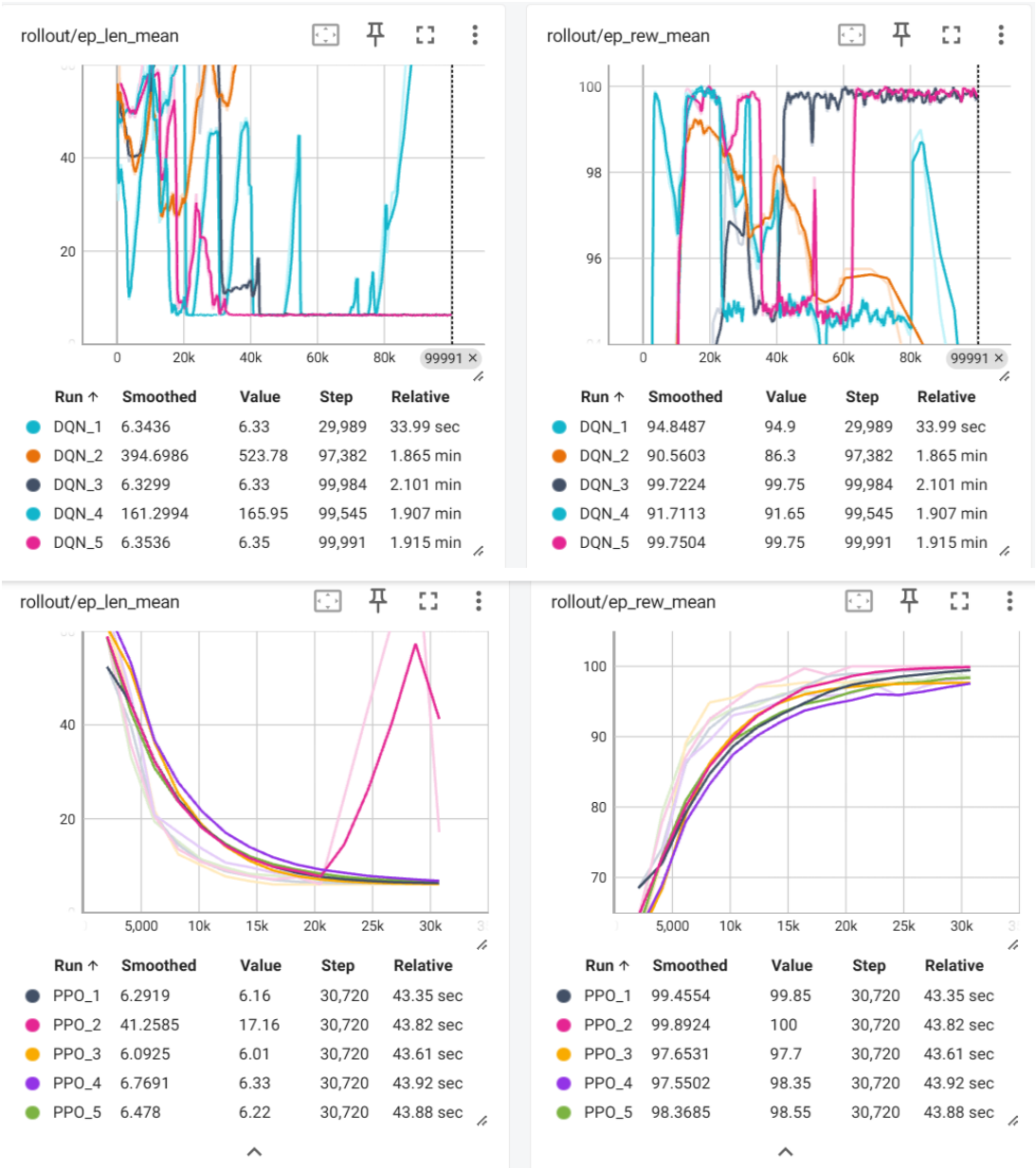


Figure 2: task2

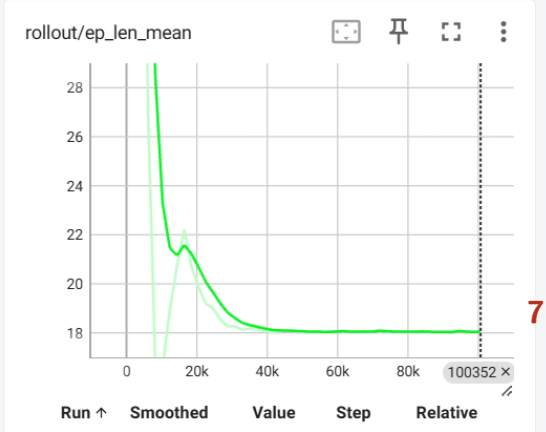
3 Task 3: Pygame for RL environment [20-points]

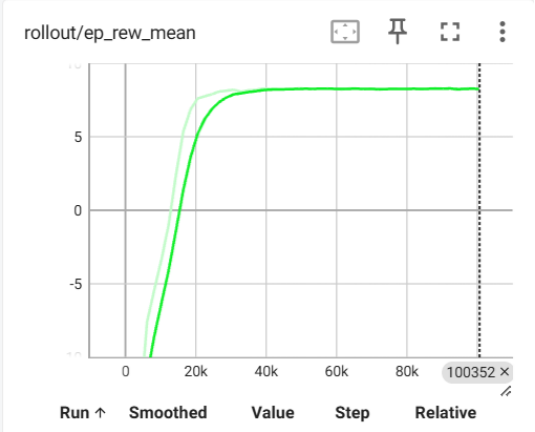
In this part we have the code like the last task but just using pygame for visualization but I change a little from the last part If we go up but we can't then we get negative reward and also make the grid world bigger and a little change of reward. (main changes are on the render function using pygame) In the picture below :

if we set the mode rgb_array the render function return the pygame surfarray it returns the graphic view of the game and we can make video from it. we have pygame.draw.line to to draw the grids and Rect for Obstacles and Goal for making them Red and Green.
Also we have pygame.draw.circle to make the circle for the player.
for quit the pygame we have pygame.quit().

```python
def render(self, mode='human'):
    # Initialize Pygame if not already done
    if self.screen is None:
        pygame.init()
        self.screen = pygame.display.set_mode((self.width, self.height))
        pygame.display.set_caption("Custom Pygame RL Environment")
        self.clock = pygame.time.Clock()

    # Fill background white
    self.screen.fill((255, 255, 255))

    # Draw grid lines for visual aid
    for x in range(0, self.width, self.cell_size):
        pygame.draw.line(self.screen, (200, 200, 200), (x, 0), (x, self.height))
    for y in range(0, self.height, self.cell_size):
        pygame.draw.line(self.screen, (200, 200, 200), (0, y), (self.width, y))

    # Draw obstacles as red squares
    for obs in self.obstacles:
        rect = pygame.Rect(obs[0] * self.cell_size, obs[1] * self.cell_size,
                           self.cell_size, self.cell_size)
        pygame.draw.rect(self.screen, (255, 0, 0), rect)

    # Draw goal as green square
    goal_rect = pygame.Rect(self.goal_pos[0] * self.cell_size, self.goal_pos[1] * self.cell_size,
                            self.cell_size, self.cell_size)
    pygame.draw.rect(self.screen, (0, 255, 0), goal_rect)

    # Draw agent as blue circle
    center = (int(self.agent_pos[0] * self.cell_size + self.cell_size / 2),
              int(self.agent_pos[1] * self.cell_size + self.cell_size / 2))
```

```python
        rect = pygame.Rect(obs[0] * self.cell_size, obs[1] * self.cell_size,
                           self.cell_size, self.cell_size)
        pygame.draw.rect(self.screen, (255, 0, 0), rect)

    # Draw goal as green square
    goal_rect = pygame.Rect(self.goal_pos[0] * self.cell_size, self.goal_pos[1] * self.cell_size,
                            self.cell_size, self.cell_size)
    pygame.draw.rect(self.screen, (0, 255, 0), goal_rect)

    # Draw agent as blue circle
    center = (int(self.agent_pos[0] * self.cell_size + self.cell_size / 2),
              int(self.agent_pos[1] * self.cell_size + self.cell_size / 2))
    pygame.draw.circle(self.screen, (0, 0, 255), center, self.cell_size // 3)

    if mode == 'human':
        pygame.display.flip()
        self.clock.tick(10)  # limit to 10 FPS
        # Process quit events to allow window closing
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.close()
    elif mode == 'rgb_array':
        return np.transpose(np.array(pygame.surfarray.array3d(self.screen)), axes=(1, 0, 2))

def close(self):
    if self.screen is not None:
        pygame.quit()
        self.isopen = False
        self.screen = None
```





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