Homework 3

Total Points: 300 (Each Assignment 100 points)

Due: Monday, Nov. 10

Overview

This homework consists of **three independent RL projects**, each worth **100 points** (total = 300 points).

Students **must implement their own RL code** (policy network, value update, replay buffer, etc.). Using prebuilt RL frameworks such as **Stable-Baselines3**, **RLlib**, or any other high-level RL library will result in a **−10 point penalty per assignment**. Each submission must include:

- Working source code
- One "best" learning curve (no averaging across runs)
- A recorded 1-minute demonstration video
- A README file explaining how to execute the trained model

HW3-1: RL on Gymnasium (LunarLander Discrete)

Goal: Train any RL algorithm (PPO, A2C, DQN, etc.) on `LunarLander-v3` (discrete).

Environment Setup: Use `gymnasium.make("LunarLander-v3")`. Implement your own training pipeline (network, update rules, etc.). Prebuilt RL libraries ⇒ −10 pts penalty.

Steps

1. Train until the agent achieves stable performance (average reward ≥ 200 recommended).
2. The **training curve must show an upward trend and convergence.**
3. Submit only **one best learning curve** (no need to average multiple runs).
4. Record a 1-minute video showing your best model landing successfully.
5. Upload the video to **YouTube (public or unlisted)** and include the link in your README.

Deliverables

```
main.py    (train + test modes)
model.pth
train_plot.png
README.md  (parameters + YouTube link)
```

Execution Requirement:

```
python main.py --test
```

The grader must be able to load the trained model and see successful execution.

HW3-2: DQN on an Atari Environment

Goal: Implement and train your own Deep Q-Network (DQN) on any Atari game, e.g., `BreakoutNoFrameskip-v4`, `PongNoFrameskip-v4`, or `SpaceInvaders-v4`.

Steps:

1. Implement your own DQN components (Q-network, target network, replay buffer, ϵ-greedy policy, etc.).
2. Train for at least 2 million steps or until convergence.
3. Submit one best training curve showing clear upward trend and convergence.
4. Prebuilt RL libraries $\Rightarrow$ –10 pts penalty.
5. Record and upload a 1-minute gameplay video to YouTube (public/unlisted).

Deliverables

```
main.py
model.pth
train_plot.png
README.md  (algorithm details + video link)
```

Execution Requirement:

```
python main.py --test
```

HW3-3: Custom AI Game + RL Agent

Goal: Design a simple custom environment (Unity ML-Agents, Pygame, Pyxel, PettingZoo, etc.) and implement an RL agent that learns to play it.

Steps

1. Define observation space, action space, and reward function.
2. Implement your own RL algorithm (Q-learning, PPO, Actor-Critic, etc.).
3. Train until the learning curve is monotonic upward and convergent.
4. Submit one best training curve.
5. Record and upload a 1-minute demonstration video to YouTube (public/unlisted).

Deliverables

```
main.py
model.pth
train_plot.png
README.md  (environment description + YouTube link)
```

Submission Format

Submit three folders:

```
HW3_1/
HW3_2/
HW3_3/
```

Each folder must contain:

- main.py
- model.pth
- train_plot.png
- README.md (environment, algorithm, hyperparameters, training summary, video link)

Grading Criteria (Per Assignment = 100 Points)

Criterion	Description	Points
Algorithm Implementation	RL algorithm implemented correctly by student	25
Learning Curve	Upward trend and convergence (best run only)	25
Performance	Agent performs the task successfully	25
Documentation	Clear code, parameters, and README	15
Reproducibility & Video	Model loads correctly and video link works	10
Penalty	Use of Stable-Baselines3 or similar library	–10
Total		**100 pts**

Total Homework 3 Points: 300

Tips

- You may use **PyTorch** or **TensorFlow** for neural network implementation, but the RL algorithm logic must be your own.
- Test model loading with:

```
model = torch.load("model.pth")
```

- Keep YouTube videos public or unlisted (private videos will not be graded).
- Submit only your best convergent learning curve.