Cairo University
Faculty of
Engineering

Computer Engineering Department

CMPS458 Reinforcement Learning Lab-2 Report

Team Number: 4
Mostafa Osama Nassar - 1210377
Maged Amgad Rasmt - 1210375

Supervisor: Ayman AboElhassan

November 12, 2025

Deliverables

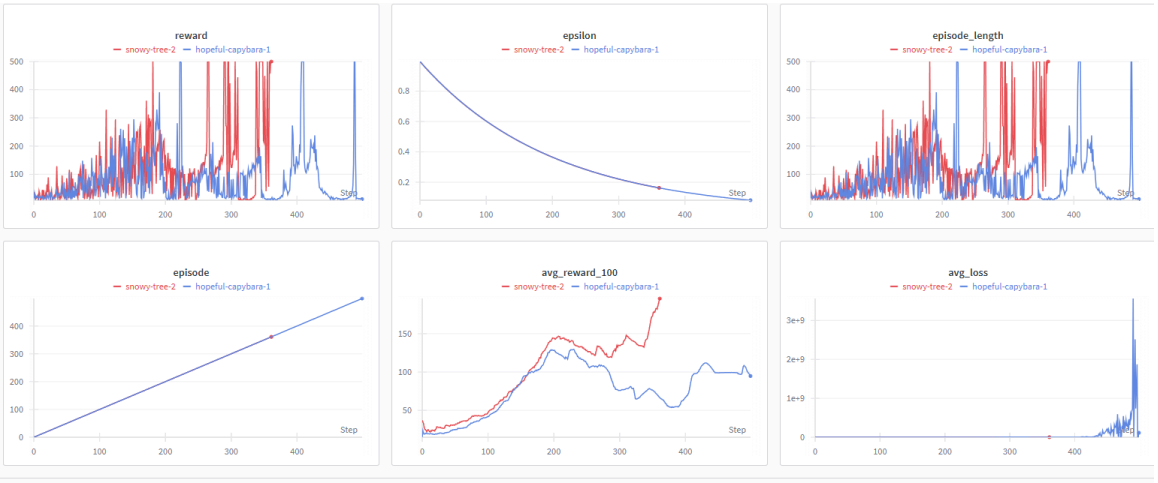
Repo Link:
https://github.com/zcrack4i0/RL-Assignments/tree/main/ASS2
Video Record Link:
https://github.com/zcrack4i0/RL-Assignments/tree/main/ASS2/videos

Discussion

0.1 Experiments

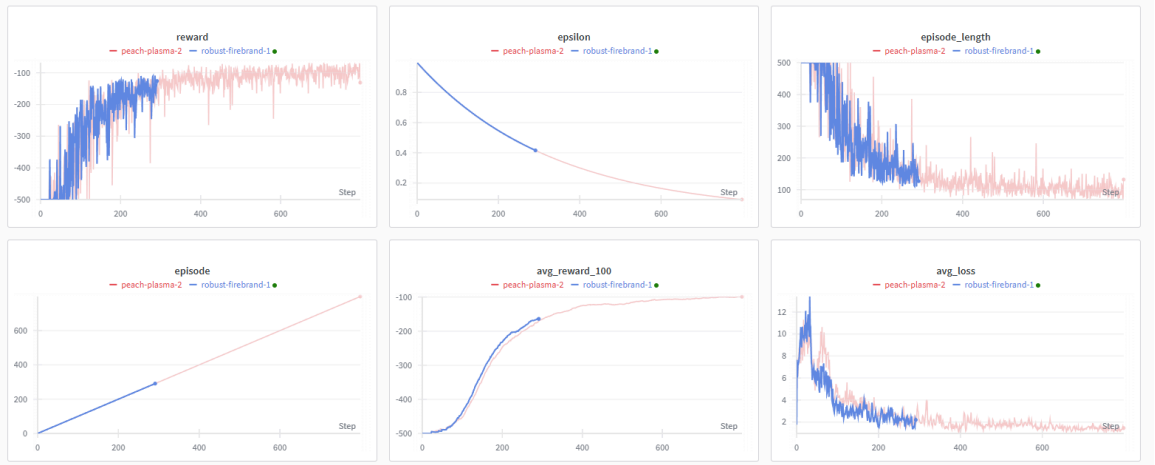
This section discusses the experiments performed and the observed outcomes. Each case studies how different parameters influence each classical environment:

Case 1 — Cartpole-v1



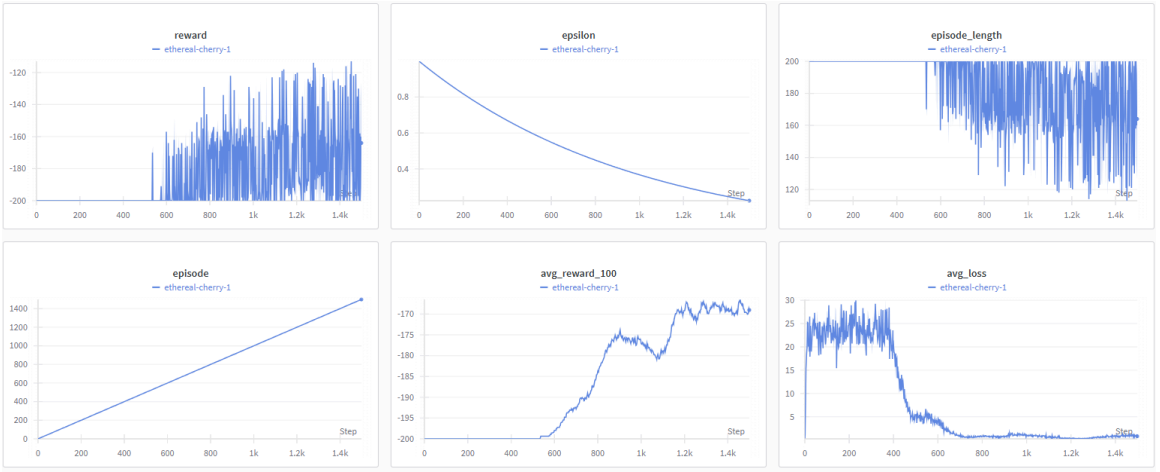
Parameter	**Effect of Increasing Value**	**Effect of Decreasing Value**
Discount Factor (γ)	Encourages infinite balancing (values long-term survival).	Agent becomes "short-sighted" and may drop the pole if immediate reward is fine.
Epsilon Decay	Exploits learned knowledge sooner (good for simple tasks).	Wastes time exploring random actions when the task is simple.
Learning Rate	Causes "overshooting" (weights change too much), leading to catastrophic forgetting.	Ensures stable, smooth convergence (though training takes longer).
Replay Memory	Improves stability by reducing data correlation.	Agent forgets how to recover from extreme angles it hasn't seen recently.
Batch size	Increasing the batch size stabilizes the gradient for smoother balancing	Decreasing it makes updates noisy and can cause the agent to "forget" how to recover from errors.

Case 2 — Acrobat-v1



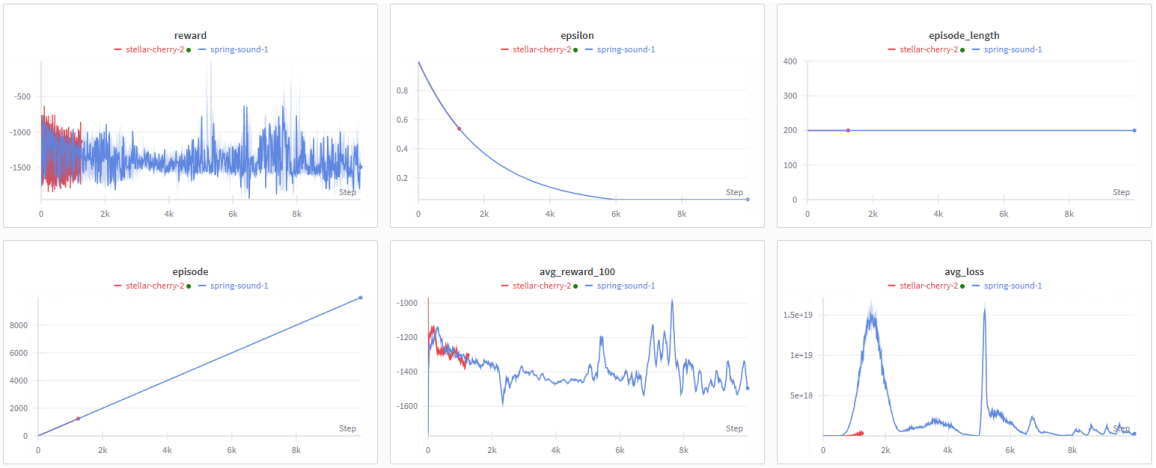
Parameter	**Effect of Increasing Value**	**Effect of Decreasing Value**
Discount Factor (γ)	Encourages the long sequence of swings required to reach the height.	Agent only learns to swing locally without aiming for the top.
Epsilon Decay	Can converge to a suboptimal "swinging" loop without reaching the target.	Allows enough exploration to find the exact swing sequence needed.
Learning Rate	Divergence is common; the physics are sensitive to bad weight updates.	Necessary for fine-tuning the swing-up mechanics.
Replay Memory	Decorrelates the highly sequential data of swinging back and forth.	High correlation in small memory makes training unstable.
Batch size	Increasing batch size helps capture the long-term momentum dependencies needed to reach the top	Decreasing it can lead to "chattering" (stuck swinging) due to high variance.

Case 3 — MountainCar-v0



Parameter	**Effect of Increasing Value**	**Effect of Decreasing Value**
Discount Factor (γ)	Necessary to propagate the value from the distant flag back to the start.	Agent sees the goal as "too far away" to care about and does nothing.
Epsilon Decay	Agent stops exploring before accidentally finding the top of the hill.	Ensures the agent explores long enough to find the flag at least once.
Learning Rate	Can destroy the delicate policy needed to build momentum.	Preferred to slowly solidify the "momentum" strategy once found.
Replay Memory	Must be large enough to store the rare successful episodes for re-training.	Agent overwrites the rare "winning" memories with "failing" ones.
Batch size	Increasing batch size is critical to ensure rare "winning" samples are included in updates	Decreasing it typically causes failure as the agent only sees negative rewards.

Case 4 — Pendulum-v1



Parameter	**Effect of Increasing Value**	**Effect of Decreasing Value**
Discount Factor (γ)	Essential for maintaining the upright position indefinitely.	Agent might spin the pendulum just to get short-term movement rewards.
Epsilon Decay	Once upright, exploration is dangerous (causes falling), so stop exploring fast.	Too much random twitching will knock the pendulum over constantly.
Learning Rate	Continuous/Discretized control is very sensitive; high LR causes wild spinning.	Low LR is needed to approximate the smooth torque curve accurately.
Replay Memory	Large memory provides a diverse dataset of positions and velocities.	Small memory leads to overfitting on a specific part of the swing.
Batch size	Increasing batch size smooths the approximation of the continuous physics curve	Decreasing it causes jerky movements and unstable torque predictions

0.2 Question Answers

Question 1:What is the difference between DQN and DDQN in terms of training time and performance?

Performance: DDQN (Double DQN) is generally expected to outperform DQN. Standard DQN suffers from **maximization bias** (it tends to overestimate the Q-values of actions because it uses the same network to select and evaluate actions). DDQN solves this by decoupling the selection of the action (using the main network) and the evaluation of that action (using the target network).

Training Time: In terms of "wall-clock" time per epoch, they are roughly similar because the architecture size is usually the same. However, DDQN often converges to a stable policy in **fewer episodes** than DQN because the learning is more stable. DQN might oscillate more before finding the optimal path.

Question 2: How stable are the trained agents?

DQN: Likely to show higher variance (instability) during training. You might see the reward go up, then suddenly crash ("catastrophic forgetting"), and then go back up.

DDQN: Should show a smoother learning curve and more consistent test durations.

Test Duration Metric:

- For **CartPole**, stability means consistently hitting the max duration (e.g., 500 steps).
- For **MountainCar**, stability means consistently solving it in fewer steps (e.g., -100 to -150 reward). If the duration varies wildly (sometimes 200 steps, sometimes 1000+), the agent is unstable.

Question 3: What is the effect of each hyperparameter value on the RL training and performance?

General Rule:

- **High Learning Rate:** Fast learning but high instability (divergence).
- **Low Epsilon Decay:** More exploration (good for sparse rewards like MountainCar) but slower convergence.
- **High Discount Factor:** Better for long-term goals (MountainCar).

Question 4: Explain from your point of view how well-suited DQN/DDQN is to solve this problem.

CartPole, Acrobot, MountainCar: DQN/DDQN are **highly suited** for these because they have **Discrete Action Spaces** (move left, move right, do nothing). DQN is designed specifically for discrete actions.

Pendulum-v1: DQN is theoretically **ill-suited** for this environment in its native form. Pendulum-v1 has a **Continuous Action Space** (applying a precise amount of torque). To solve the Pendulum with DQN, you must **discretize** the action space (e.g., force the agent to choose between Torque -2.0, 0, and +2.0). Algorithms like DDPG or SAC are usually better for Pendulum, but DQN can work if you modify the environment wrapper to accept discrete inputs.