Names: Priel Kaizman (315627448), Adir Serruya (316472455)

DRL – Assignment 1

Section 1 – Q Learning

Answers:

1. In environments where the dynamics are unknown, the Value-iteration method cannot be effectively implemented since it is model-based and requires full and accurate knowledge of the transition probabilities and rewards for each state-action pair. Without this information the agent cannot accurately compute the expected return for each state-action pair.

2. Model-free methods like SARSA and Q-learning resolve the problem of unknown or complex environment dynamics by learning directly from the interaction with the environment, rather than relying on a predefined model, therefor full knowledge of the environment dynamics is not required.

3. The main difference between SARSA and Q-learning lies in how they update their Q-values, reflecting their approach to the exploration-exploitation tradeoff:

SARSA:
An on-policy method.
In SARSA, the Q-value update is based on the action **actually taken** by the policy in the next state (S'). Therefore, the update rule uses the Q-value of the next state-action pair that the policy selects. This makes SARSA more conservative and policy-dependent, as it evaluates the policy it follows.

Q-learning:
An off-policy method.
Updates its Q-values using the maximum Q-value for the next state across all possible actions, regardless of the action the current policy would select. This approach makes Q-learning more aggressive and optimistic in seeking the optimal policy, as it evaluates the best possible action at each step, even if the policy does not currently select that action.

4. A decaying epsilon-greedy strategy is better than acting purely greedily because it ensures a balance between exploration and exploitation. Initially, a higher epsilon encourages exploration of different actions, crucial for learning about the environment because at first the agent has no information about it. As learning progresses, decreasing epsilon shifts the focus towards exploitation, using the knowledge gained to choose actions that yield higher rewards.

Report:

Hyperparameters used in the final solution:

alpha = 0.1

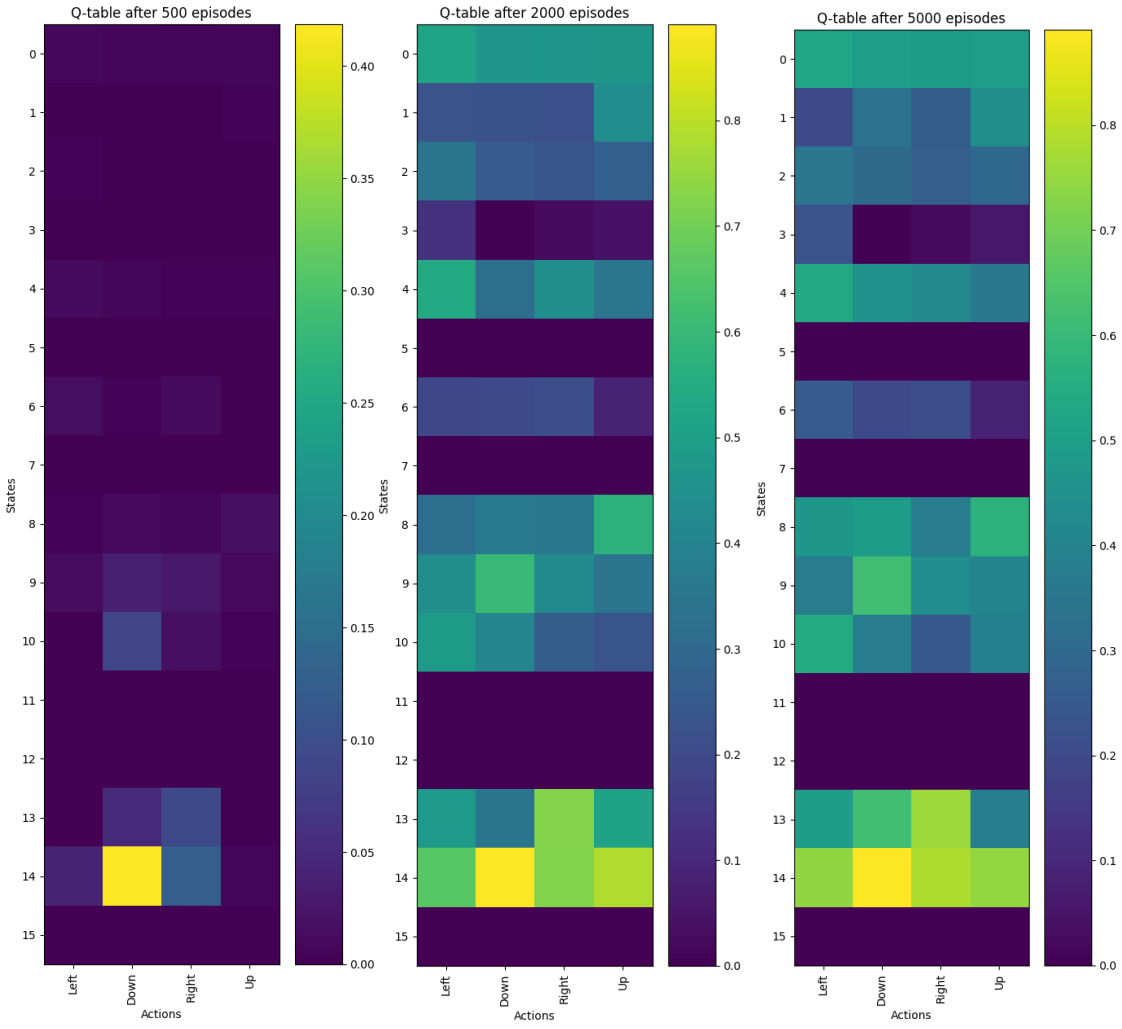
gamma = 0.99

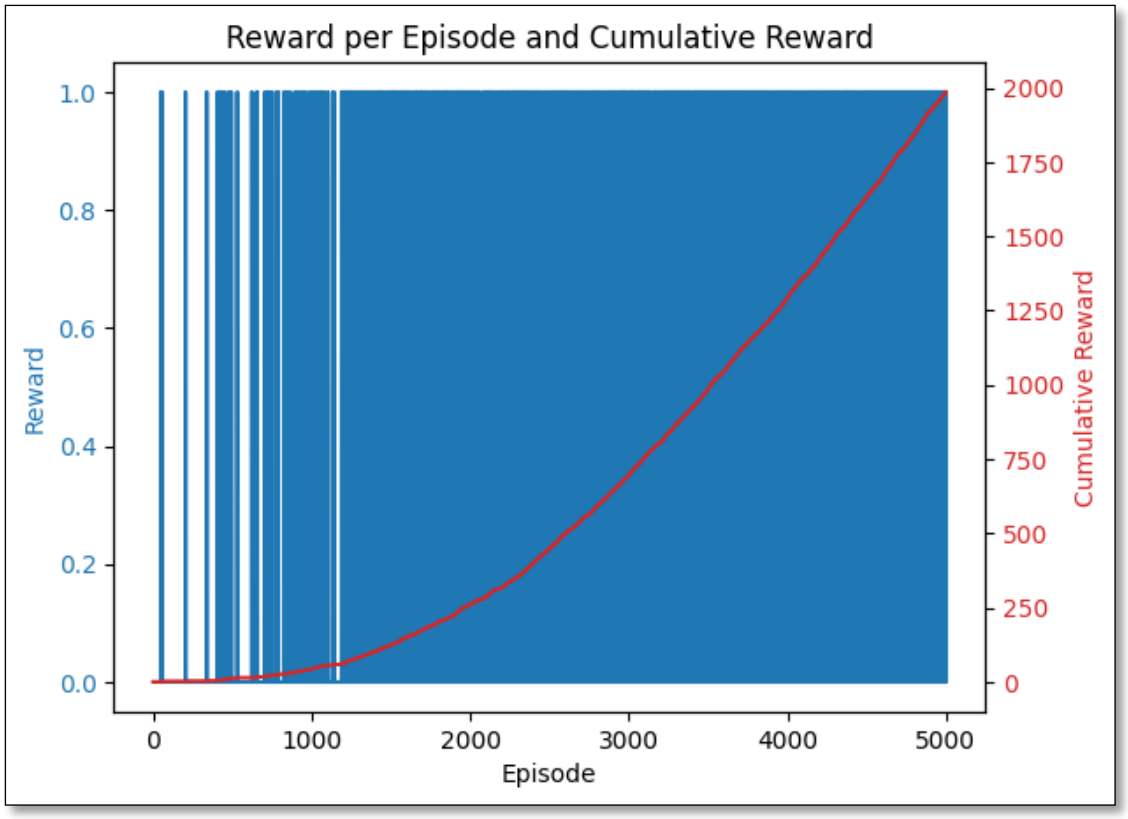
epsilon_start = 1.0

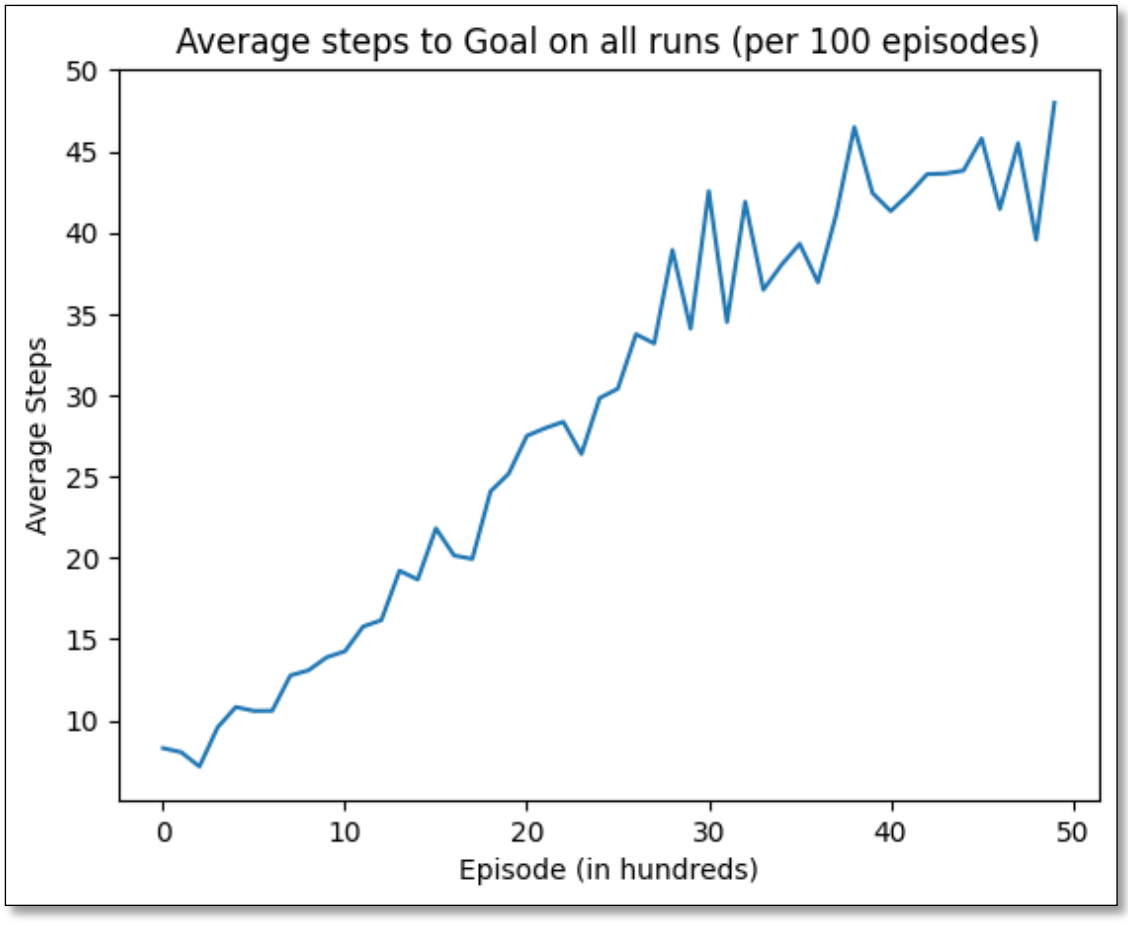
epsilon_end = 0.01

epsilon_decay (per episode) = 0.999

Q-tables:




Reward per Episode and Cumulative Reward
Reward
Cumulative Reward
Episode
0.0
0.2
0.4
0.6
0.8
1.0
0
250
500
750
1000
1250
1500
1750
2000
0
1000
2000
3000
4000
5000


Average steps to Goal on all runs (per 100 episodes)
Average Steps
Episode (in hundreds)
10
15
20
25
30
35
40
45
50
0
10
20
30
40
50

Section 2 – DQN

Answers:

1. Sampling in a random order is important since it prevents the model from learning sequential patterns which might be misleading. The sample efficiency is also enhanced when done in random since each sample has an equal chance of being included in the training. In addition, random sampling contributes to a more consistent learning trajectory by reducing the variance of learning updates. Most importantly, it helps generalize the learning using various states and actions, which makes the resulting policy more robust.

2. Using an older set of weights to compute the targets improves the model, since it stabilizes the learning process by preventing the chase of a moving target, which causes a feedback loop. This design also reduces the variance in the target Q value, which makes the training smoother and less reactive to changes. Finally, it allows the policy to consolidate before updates, which makes the learning more reliable and stable.

Report:

We tried creating a DQN with 3 layers and with 5 layers, each layer in both models had 32 units.

The 3 layer model performed better in the span of 1000 episodes, it both converged faster**, reaching the average reward of 475 over 100 episodes within 647 episodes** compared to >1000, it also reached a higher top average reward over 100 episodes (1265.51 compared to 450).

Hyperparameters used in the final solution (using a 3-layer DQN):

batch_size = 256

gamma = 0.99

epsilon_start = 1.0

epsilon_end = 0.05

epsilon_decay (per episode) = 0.99

learning_rate = 0.001

update_freq (in steps) = 1000

Replay buffer size = 100000

Max steps per episode = 10000

We found that the exploration strategy parameters (epsilon decay and gamma) are the hyperparameters that affected performance the most, we conducted an extensive comparison between the changes of both and concluded that gamma was the hyperparameter that affected results the most. Gamma determines the importance of future rewards, making the agent value future rewards more when the gamma is higher. It greatly influences the policy learned by the agent since it controls the weighting of immediate versus long term benefits.

The comparisons between the 2 hyperparameters are detailed in the following section, we found that the difference in performace when using different gamma values was greater than when using different epsilon decay values.

Comparisons:

Epsilon Decay	Max MA	Average MA	Episodes to 475
0.95	948.14	219.5	708
0.97	686.8	194.7	666
0.99	1265.51	249.47	647
0.997	466.36	93.29	>1000

Max MA distance between best run and average = 423.8
Average MA distance between best run and average = 60.23
Episodes to 475 MA distance between best run and average = 108.25

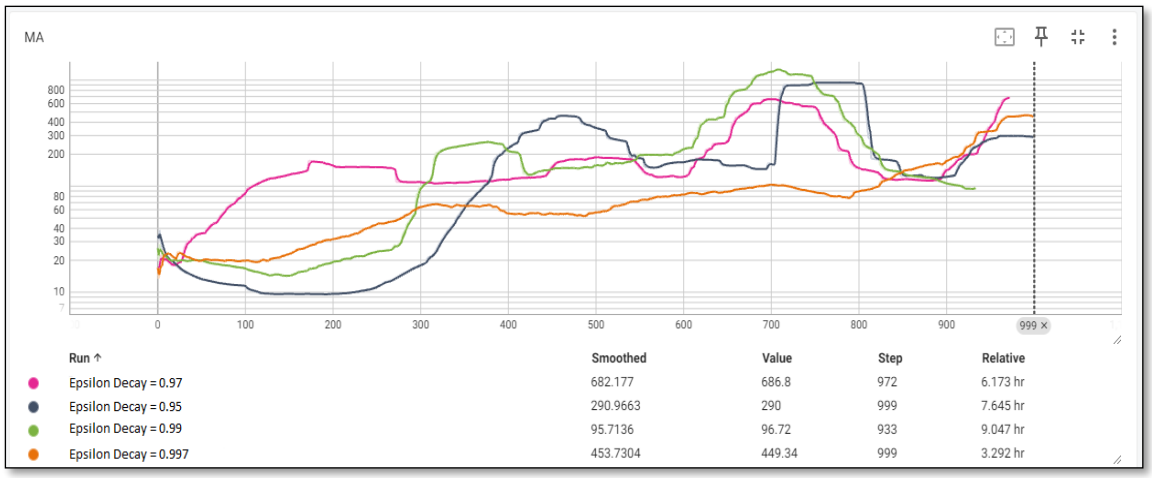
Gamma	Max MA	Average MA	Episodes to 475
0.95	268.42	117.82	>1000
0.97	557.85	145.81	946
0.99	1265.51	249.47	647
0.995	762.57	176.63	802

Max MA distance between best run and average = **551.92**
Average MA distance between best run and average = **77.03**
Episodes to 475 MA distance between best run and average = **201.75**

Having the gamma value of 0.99 caused the training to reach an average reward of at least 475 over 100 consecutive episodes sooner, it also led to the highest average MA reward across the entire training period (1000 episodes) and the highest max MA reward across the entire training period.

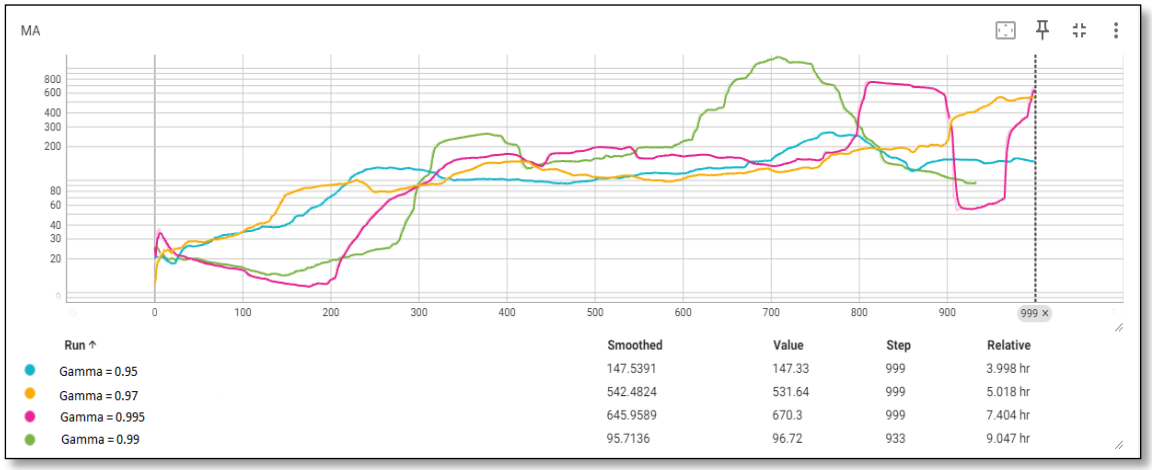
** MA = The moving average of the reward over 100 consecutive episodes*

Epsilon decay MA graph:



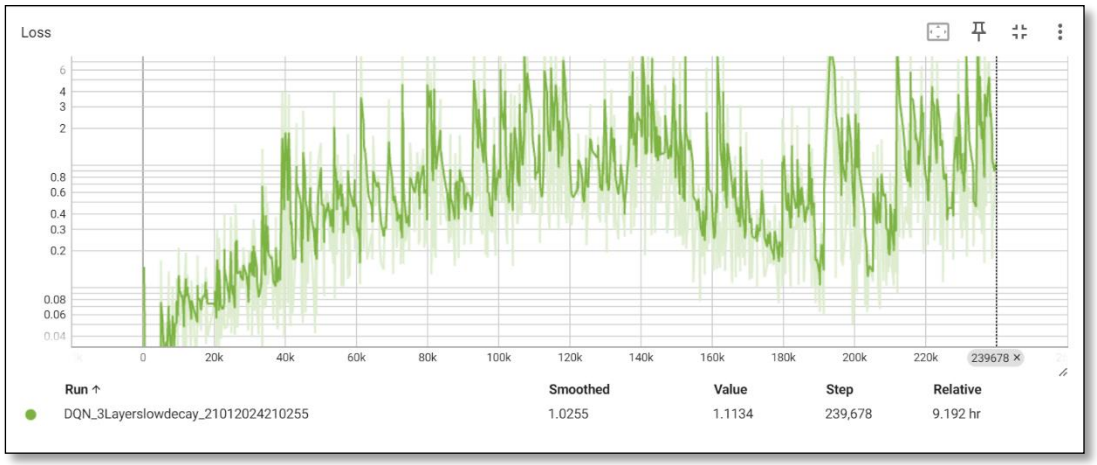
Run	Smoothed	Value	Step	Relative
Epsilon Decay = 0.97	682.177	686.8	972	6.173 hr
Epsilon Decay = 0.95	290.9663	290	999	7.645 hr
Epsilon Decay = 0.99	95.7136	96.72	933	9.047 hr
Epsilon Decay = 0.997	453.7304	449.34	999	3.292 hr

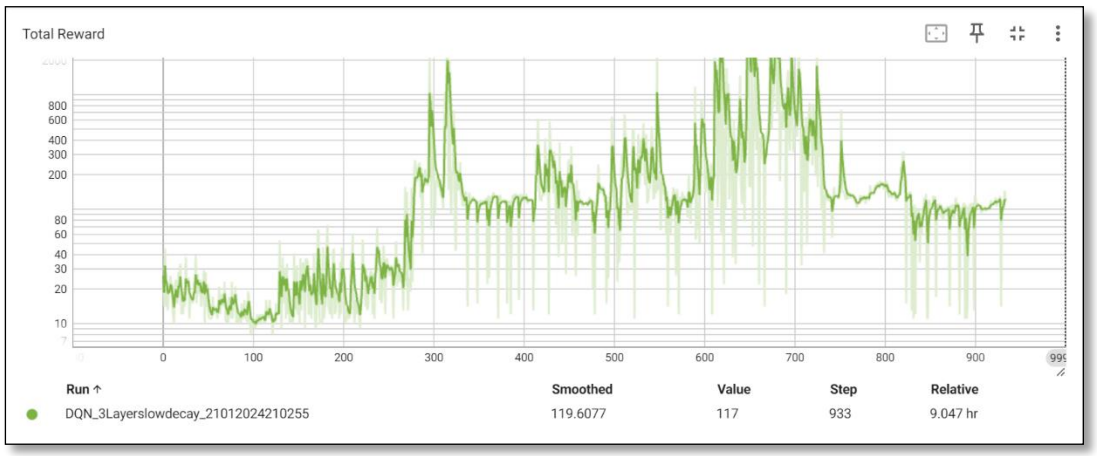
Gamma MA graph:

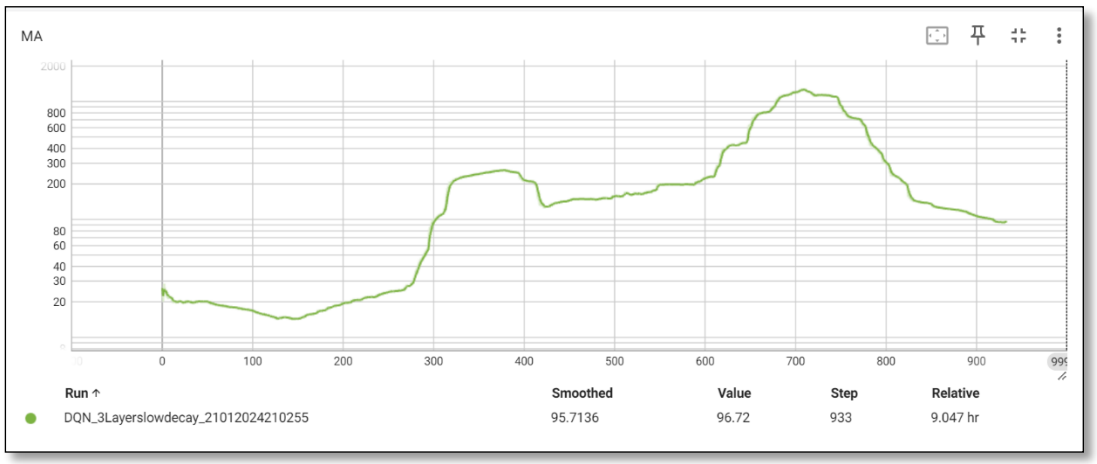


Run	Smoothed	Value	Step	Relative
Gamma = 0.95	147.5391	147.33	999	3.998 hr
Gamma = 0.97	542.4824	531.64	999	5.018 hr
Gamma = 0.995	645.9589	670.3	999	7.404 hr
Gamma = 0.99	95.7136	96.72	933	9.047 hr

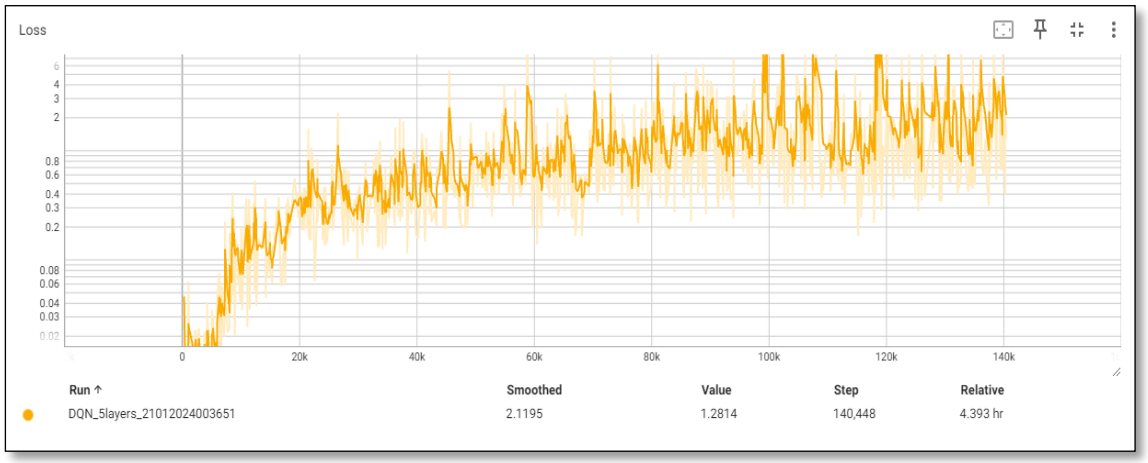
Plots of the 3-layer model with the best hyperparameters:

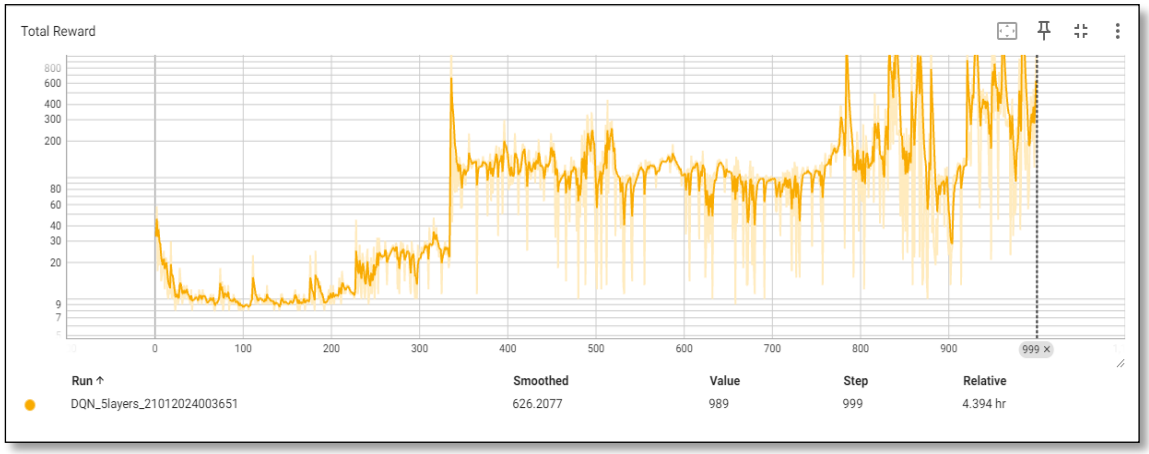


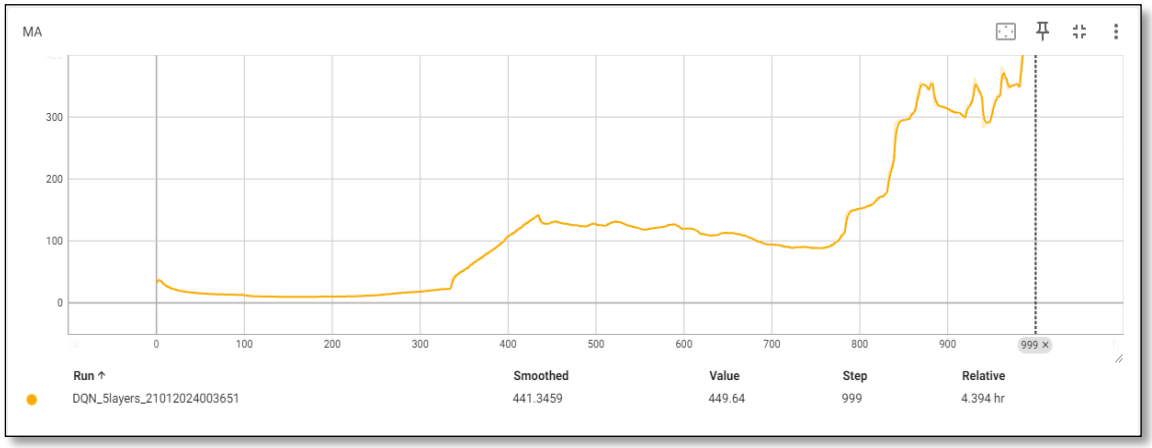




Plots of the 5-layer model:







Section 3 – Dueling DQN

Report:

We created a Dueling DQN with the same 3 layers, each with 32 units.

Hyperparameters used in the final solution (same as the DQN, for better comparison):

batch_size = 256

gamma = 0.99

epsilon_start = 1.0

epsilon_end = 0.05

epsilon_decay (per episode) = 0.99

learning_rate = 0.001

update_freq (in steps) = 1000

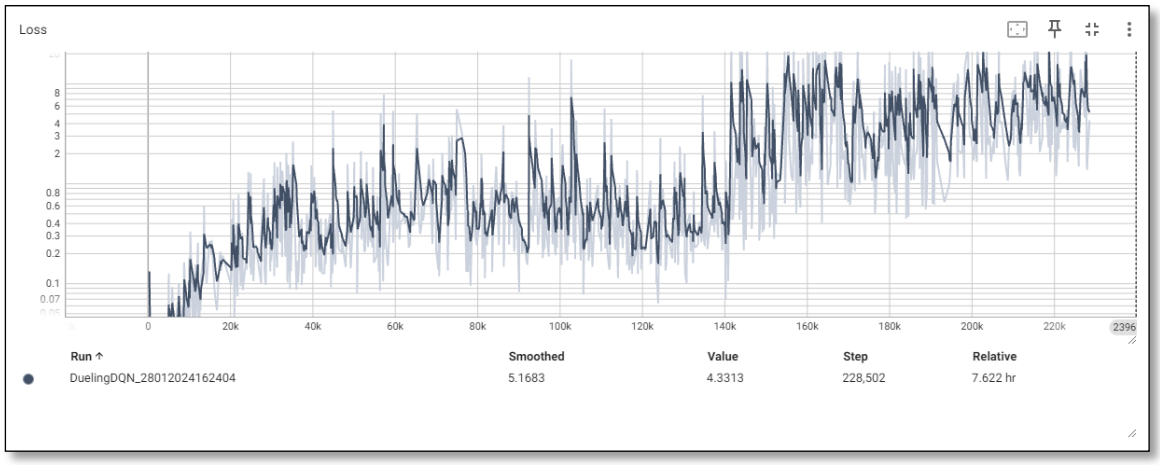
Replay buffer size = 100000

Max steps per episode = 10000

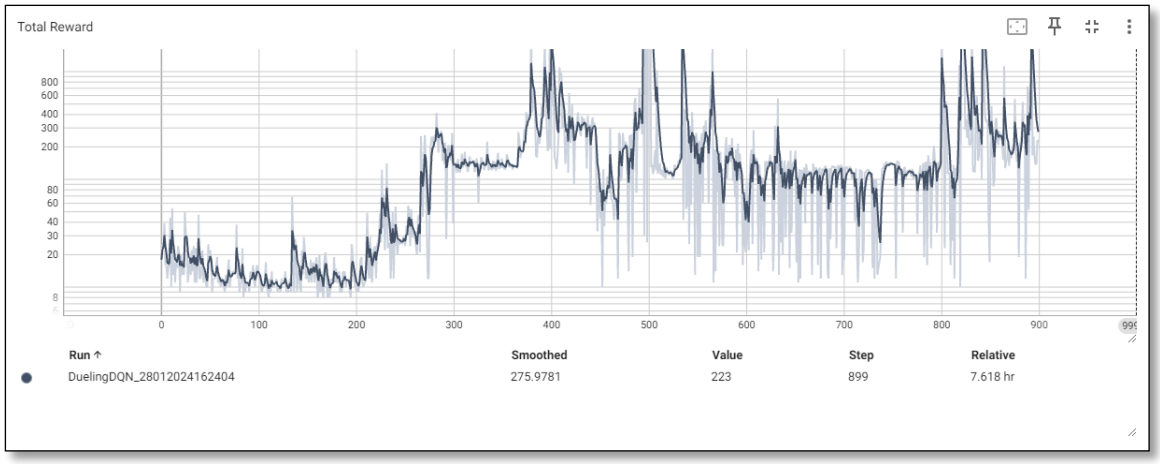
Advantages of using a Dueling DQN:

The Dueling DQN architecture differentiates between the values of states and the advantage of each action taken in a specific state, compared to the regular DQN which uses a Q-value for each state-action pair without distinction.
By splitting the state value and action advantage estimations, the agent benefits especially when the action choice doesn't significantly change the outcome of the current situation, this applies to the CartPole problem since in certain states where the pole is about to fall the next action is unlikely to remedy the situation, the agent will most likely try to avoid these dangerous states altogether.
This difference allows the Dueling DQN architecture to have more efficient learning and to potentially converge faster.

Loss:



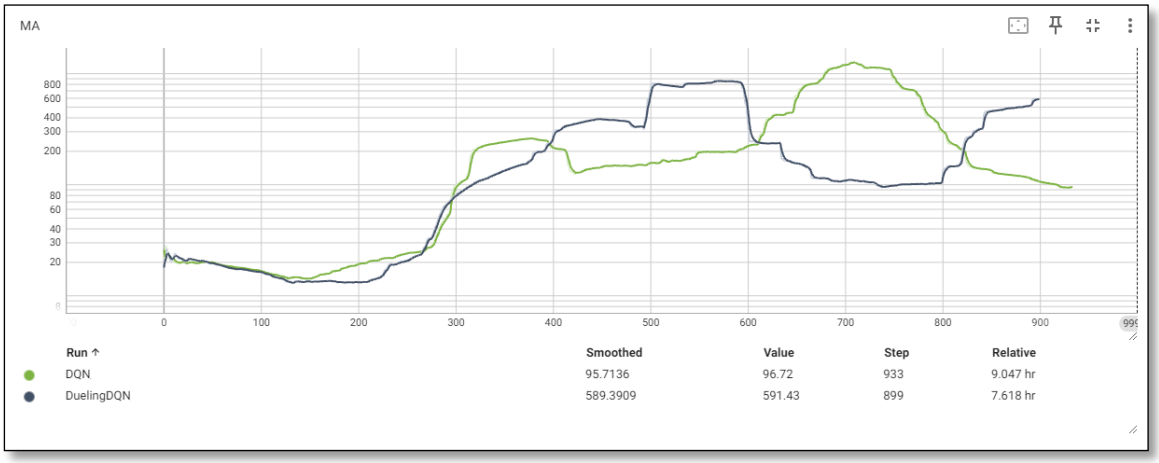
Reward per episode:



Comparison:

Architecture	Max MA	Average MA	Episodes to 475
DQN	1265.51	249.47	647
Dueling DQN	859.73	225.47	496

Moving average:



The Dueling DQN architecture had a faster convergence as expected due to its learning efficiency, however the rest of the run was worse than the basic DQN architecture, which had a higher maximum moving average over 100 consecutive episodes and a higher reward average. The Dueling DQN agent got to an average reward of 475 over 100 consecutive episodes within 496 episodes, which is significantly faster than any of our previous runs.
The shortcomings of the Dueling DQN may be due to overfitting and convergence to a sub-optimal policy (since it converged quickly).

Instructions for running the scripts

1. Preliminary Setup:

Ensure that all required packages are installed in your environment, these dependencies are listed in the requirements.txt file.

You can install them using the command: pip install -r requirements.txt

2. Script Descriptions:

- QLearning.py: This script is for Task 1, focused on the FrozenLakeV1 environment. It implements the Q-Learning algorithm.

- DQN.py: This script addresses Task 2 and is tailored for the CartPoleV1 environment, utilizing the Deep Q-Network (DQN) approach.

- DuelingDQN.py: Designed for Task 3, this script also targets the CartPoleV1 environment but employs the Dueling DQN strategy.

3. Execution and Outputs:

- Each script has been configured with optimal hyper-parameters that yield our best results.

- Upon execution, these scripts will save necessary logs and display the relevant plots as specified in the requirements.

4. Running the Scripts:

- To execute a script, simply run it in your Python environment. For example by using the command: python QLearning.py

- Repeat the process for DQN.py and DuelingDQN.py as needed.