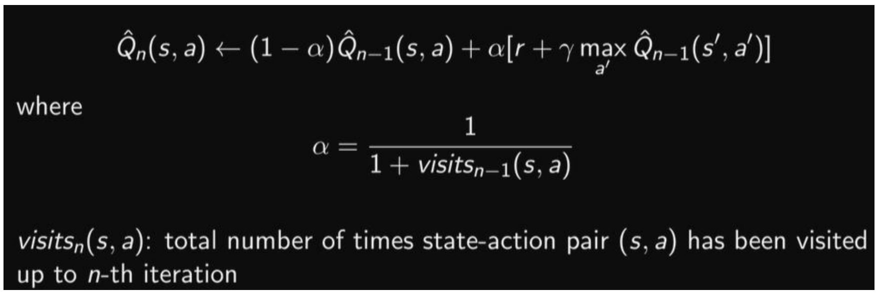
Reinforcement Learning applied to Pitfall environment

1) Introduction

This project will focus on exploring and applying two approaches in the field of reinforcement learning to obtain the best outcome possible by playing the 1982 videogame “Pitfall”, environment taken from “Gymnasium”.
The main components of the following reinforcement learning problem (which can be identified by environment, agent, states and reward) are respectively represented by the famous Atari game itself, the playable character, the different areas in which it can be (along with internal and external features that are effect of agent’s actions) and finally the score and the points the agent is able to collect or lose.
The two methods that will be implemented are:

- Tabular Q-Learning, which consists of building a matrix (or a table, or in my case a dictionary) in which the columns are associated with all the actions that can possibly be executed by the agent and the rows are related to all the different states that the agent can explore. Each entry will contain a numeric value computed through the deep q values update formula in the non-deterministic case:



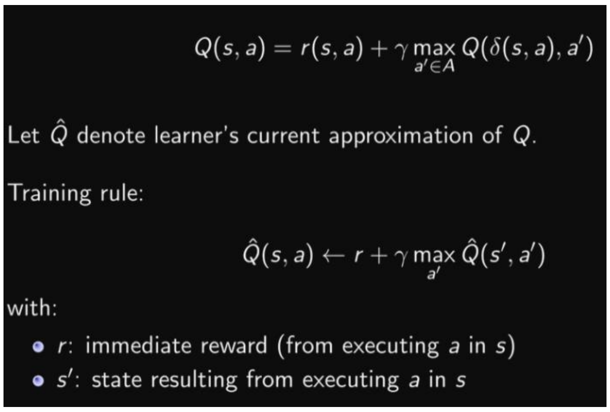
$$\hat{Q}_n(s,a) \leftarrow (1-\alpha)\hat{Q}_{n-1}(s,a) + \alpha[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')]$$

where

$$\alpha = \frac{1}{1 + visits_{n-1}(s,a)}$$

$visits_n(s,a)$: total number of times state-action pair (s,a) has been visited up to n-th iteration

- Deep Q-Learning, which uses the same formula to update the q values but does not rely on Q table anymore. Instead, a neural network is employed with a feedforward technique and a simple structure in the architecture. The update rule for this approach is the following:



$$Q(s,a) = r(s,a) + \gamma \max_{a' \in A} Q(\delta(s,a),a')$$

Let $\hat{Q}$ denote learner's current approximation of Q.

Training rule:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

with:

- r: immediate reward (from executing a in s)
- s': state resulting from executing a in s

The employed environment observation space is RAM which gives a representation of a state through an array of 128 cells, each containing an integer value belonging to the set [0, 255]. In the “gymnasium” webpage of this environment no more explanation is given about what each value of the state array means, so it is assumed to be just used as a mere input to the two approaches implemented.

Key aspects of the project will include:

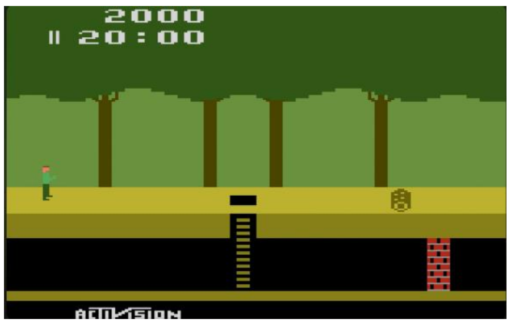
a) Analyzing how the two learning methods will perform in the environment (in terms of time and metrics), using the same hyperparameters and settings throughout the training.
b) Observing the changes and\or improvements made on the two methodologies during time.

2) Brief definition of the macro-concepts

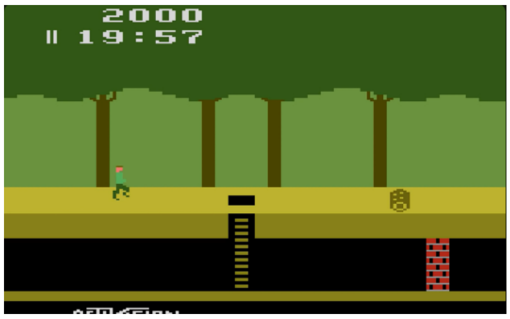
- Reinforcement Learning: branch of machine learning where an agent learns to make decisions by interacting with an environment, aiming to maximize cumulative rewards through trial and error.
  The agent takes actions, receives feedback as rewards or penalties, and adjusts its strategy (policy) to improve future performance based on its experiences.
- Deep Q-Learning: model-free reinforcement learning algorithm that helps an agent learn the optimal policy by estimating the value (Q-value) of each action in each state. It updates these Q-values based on the Bellman equation, allowing the agent to maximize cumulative rewards over time, without requiring a model of the environment.
- Neural Network: computational model inspired by the human brain, consisting of interconnected layers of nodes (neurons) that process and learn patterns in data. It transforms inputs through weighted connections and activation functions to produce intermediate results until finally producing outputs, enabling tasks like classification, prediction, and pattern recognition.

3) Task

- Regression Problem -> Q: S->R^action_number
- State: vector representing the emulator's RAM state.
- Action: an integer in the range L = [0, 17].
  Example:



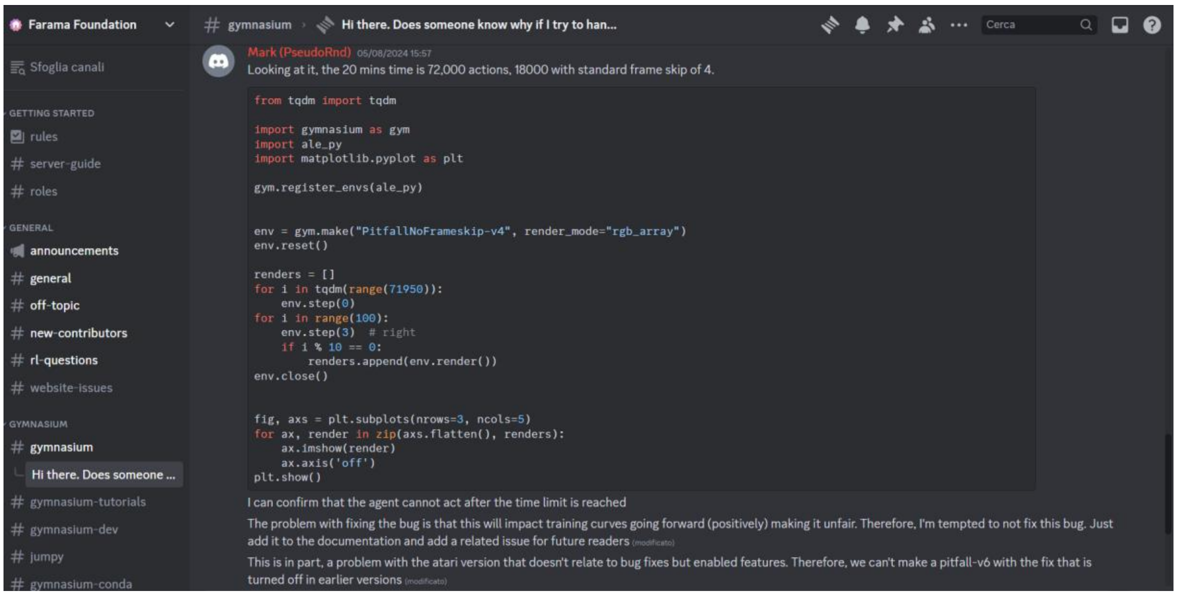
Action 0: NOOP



Action 3: RIGHT

- Reward: points returned by the environment under certain conditions.
  In the original version of the environment (which I renamed in my python files as version v1) points are lost when an obstacle is hit, or the agent falls in a hole, and they are gained when a treasure is collected.
  In the modified version of the environment (which I changed a bit and called it v2), very few points are also assigned if the agent executes an action that brings it to state where he has not lost points nor lives, but a significant malus is assigned if it loses a life.
- Goal: learn to play as best as possible (could be to avoid dying or losing points, to collect points, to survive as much as possible or, at best, to complete the game).
- Other features: game ends when the three available lives are lost, or the game time runs out or the agent wins and collects all treasures.
- NOTE: due to a bug in the environment's implementation from the "gymnasium.farama team", the game does not reset at the game time

running out, so I had to set a step limit number (so after k steps executed, the game will reset the environment).



Mark (PseudoRnd) 05/08/2024 15:57

Looking at it, the 20 mins time is 72,000 actions, 18000 with standard frame skip of 4.

```
from tqdm import tqdm

import gymnasium as gym
import ale_py
import matplotlib.pyplot as plt

gym.register_envs(ale_py)


env = gym.make("PitfallNoFrameskip-v4", render_mode="rgb_array")
env.reset()

renders = []
for i in tqdm(range(71950)):
    env.step(0)
for i in range(100):
    env.step(3)  # right
    if i % 10 == 0:
        renders.append(env.render())
env.close()


fig, axs = plt.subplots(nrows=3, ncols=5)
for ax, render in zip(axs.flatten(), renders):
    ax.imshow(render)
    ax.axis('off')
plt.show()
```

I can confirm that the agent cannot act after the time limit is reached

The problem with fixing the bug is that this will impact training curves going forward (positively) making it unfair. Therefore, I'm tempted to not fix this bug. Just add it to the documentation and add a related issue for future readers (modificato)

This is in part, a problem with the atari version that doesn't relate to bug fixes but enabled features. Therefore, we can't make a pitfall-v6 with the fix that is turned off in earlier versions (modificato)

Screenshots of the conversation on discord with the staff of gymnasium

- Environment version: “ALE/Pitfall-ram-v5”.

4) Technical Features

4.1) Overview of Environment-Agent interaction:

In the case of TQL (Tabular-Q Learning), the Agent has only one component (a Driver), which coordinates its actions and learning process towards the environment by selecting actions randomly (in case of exploration) or based on the Q-table's best values for the state where it stands (while exploiting).
Exploration-Exploitation paradigm is ruled by the epsilon-greedy policy (which will be better explained later).
Every time the agent performs an action, the q-values for that pair action-state are updated (if the state was already visited by the agent) or initialized to 0 (if the state was not discovered yet).
To sum up, the agent simply performs an action a while in state s and the environment gives back a pair (s’, r), which are respectively the new state in which the agent finds itself after executing a and reward r gained.

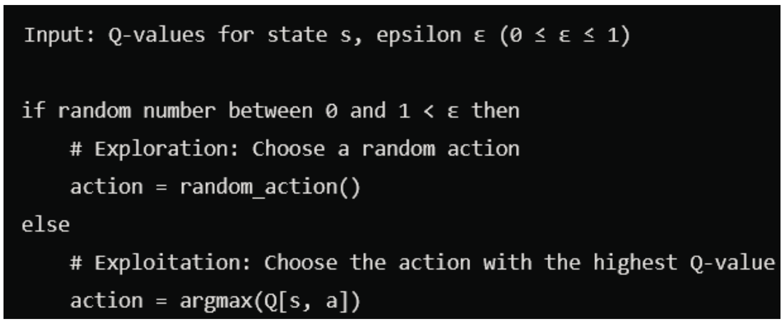
In the case of DQL (Deep-Q Learning) is conceptually the same as for the TQL, but here there is a Feedforward Neural Network instead of Q-Table and a Replay Memory, which is a technique that improves the update phase of the q-values by storing at every step the tuple (s, a, s’, r, done) in this memory and then at each step optimizes and updates values by randomly picking one of these tuples stored in the replay memory.
To sum up, the agent performs an action a while in state s and the environment gives back a pair (s’, r). Then the tuple (s, a, s’, r, done) is added to the replay memory, the q-network predicts (while updating them) the q-values, examining minibatches extracted randomly from the memory.

4.2) Epsilon-greedy policy:

Epsilon is a parameter that controls the trade-off between exploration and exploitation in reinforcement learning:

- **Exploration**: Trying new actions to discover their potential rewards.
- **Exploitation**: Choosing the best-known action to maximize the reward based on current knowledge

The epsilon greedy policy is explained here, simply and concisely:



```
Input: Q-values for state s, epsilon ε (0 ≤ ε ≤ 1)

if random number between 0 and 1 < ε then
    # Exploration: Choose a random action
    action = random_action()
else
    # Exploitation: Choose the action with the highest Q-value
    action = argmax(Q[s, a])
```

The choices I made about the epsilon throughout the entire training are to set it to 1 at the very beginning and applying it an epsilon decay at the end of each episode during training, until it reaches an epsilon minimum value and stays the same.

The epsilon-decay adopted is computed through the exponential decay formula attached inside “src” directory and the epsilon-min is in the range of [0.1,0.2].

4.3) Other hyperparameters:

The whole main training was divided into parts of K episodes in the range of [100,1000] each and it was developed in two different versions using the two mentioned approaches: 4 files in total (v1 for the original environment reward, and v2 for the modified environment reward).
Step limit to reset environment was set constantly to 17000.
Decay for epsilon is computed as follows:

- eps_decay = (eps_min / epsilon_start)^ (1/num_episodes).

As for DQL:

- learning rate = 0.01, 0.001 or 0.0001
- gamma = variable depending on how many episodes were already explored by the training.
- replay memory size = 100000
- batch size = 16 or 32

As for TQL:

- alpha = x belonging to [0, 1)
- gamma = variable depending on how many episodes were already explored by the training.

In version v2 the added rewards are:

- +0.2 if agent performs an action that does not make it lose points or lives.
- -200 if agent loses a life after action a.

4.4) FNN architecture:

It has:

- input layer, size 128.
- Hidden layer 1, size 512.
- Hidden layer 2, size 128.
- Output layer, size 18.

Activation functions used: ReLu

4.5) Note:

In the next sections the results will be examined, reporting the graphs and statistics of the main parts and their metrics.

There will also be other graphs and text files attached to the project on GITHUB describing some other aspects (including also exploration/exploitation counters per episode, time of execution, steps taken, and known/unknown states counters for q-tables).

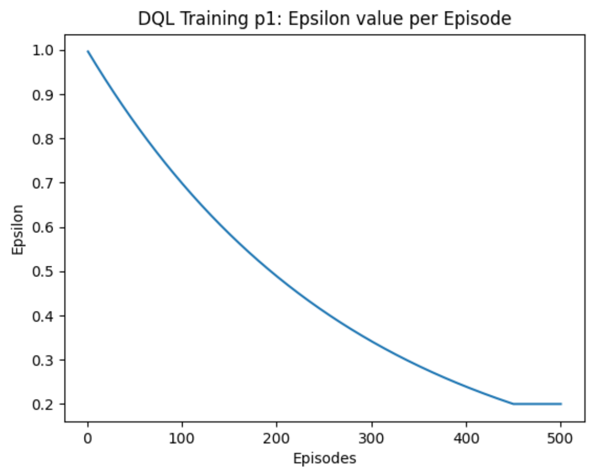
5) Training 1 performances

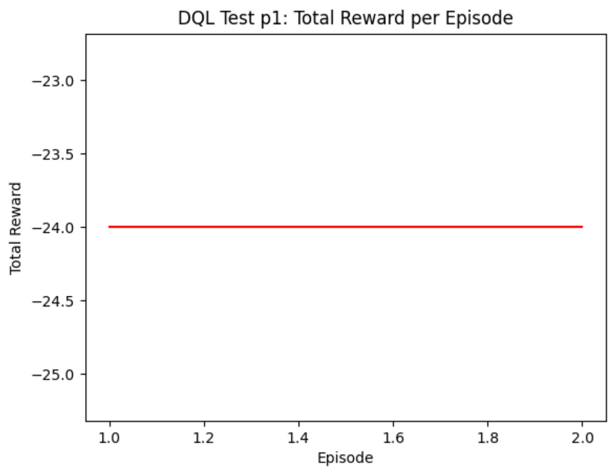
5.1) Part 1:

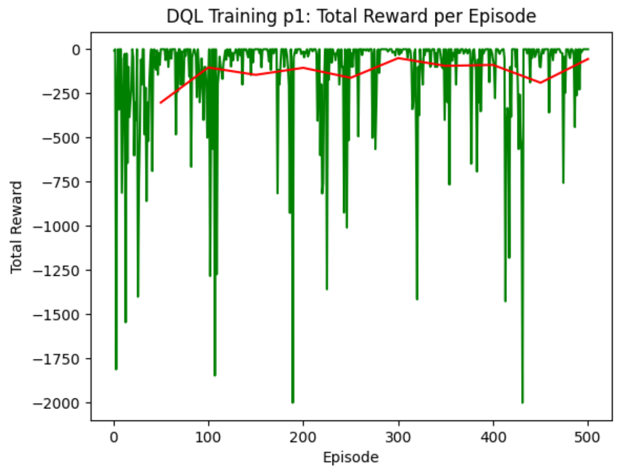
- Configuration for dql_v1:

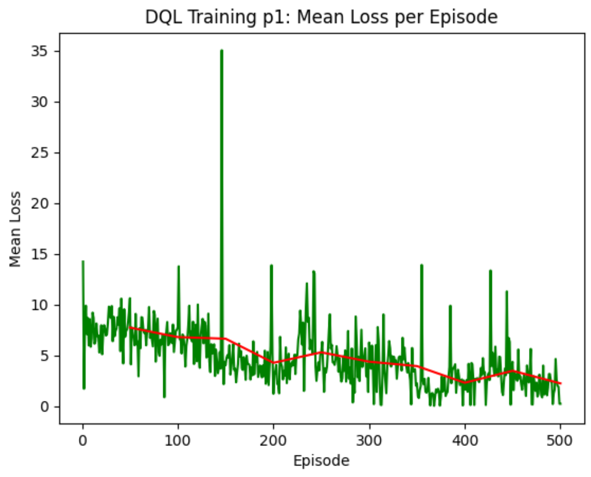
Number of training episodes=500
Number of test episodes=2
Stability=50
Epsilon=1
Epsilon Decay=0.99642985946637
Epsilon Min=0.2
Alpha=0.0001
Gamma=0.9
Replay memory size=100000
Batch size=32
Step limit for environment reset=17000

RESULTS ->





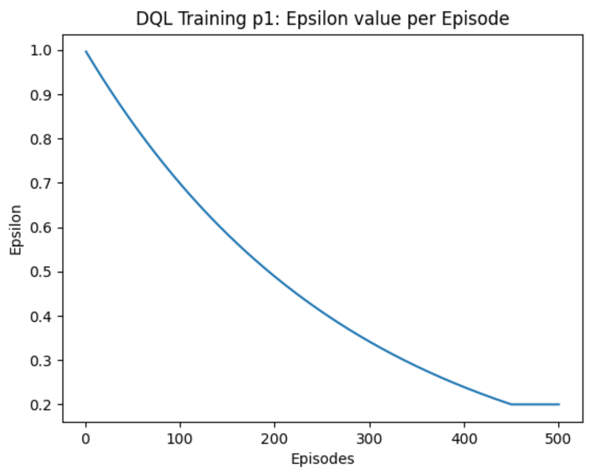


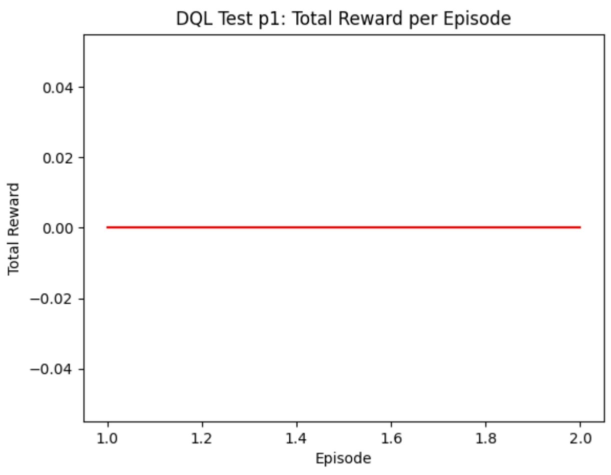


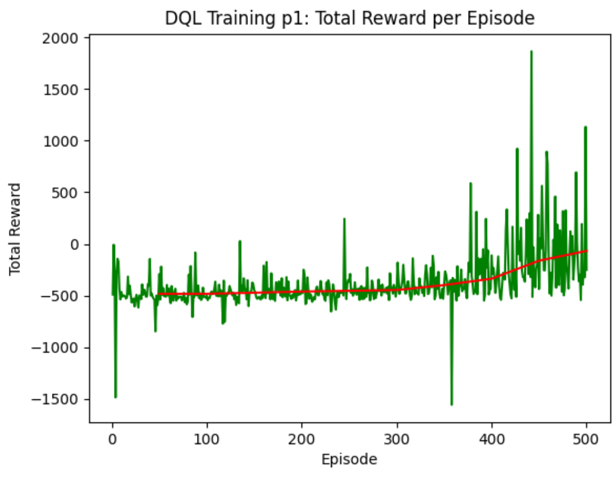
- Configuration for dql_v2:

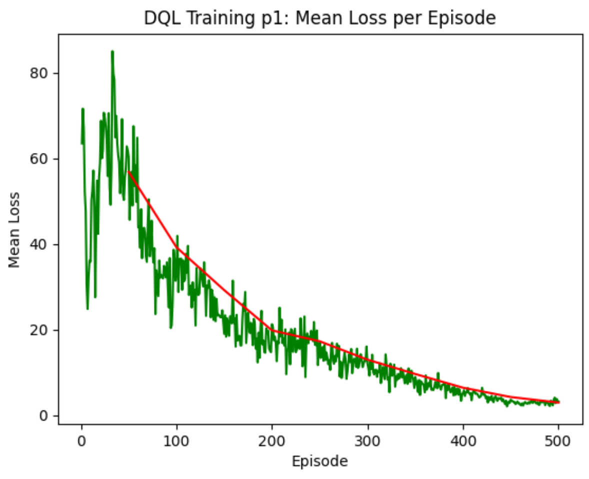
Number of training episodes=500
Number of test episodes=2
Stability=50
Epsilon=1
Epsilon Decay=0.99642985946637
Epsilon Min=0.2
Alpha=0.0001
Gamma=0.9
Replay memory size=100000
Batch size=32
Step limit for environment reset=17000

RESULTS ->





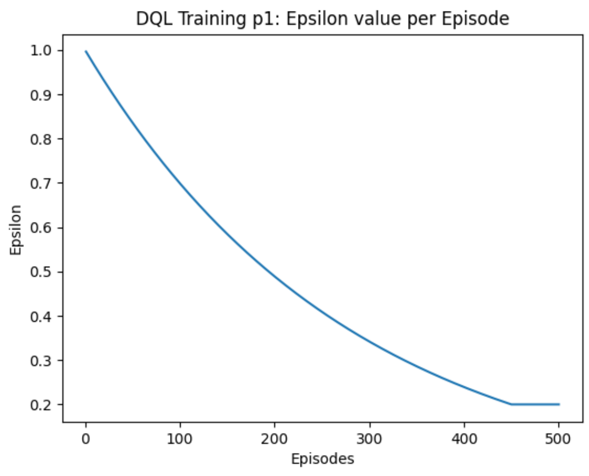


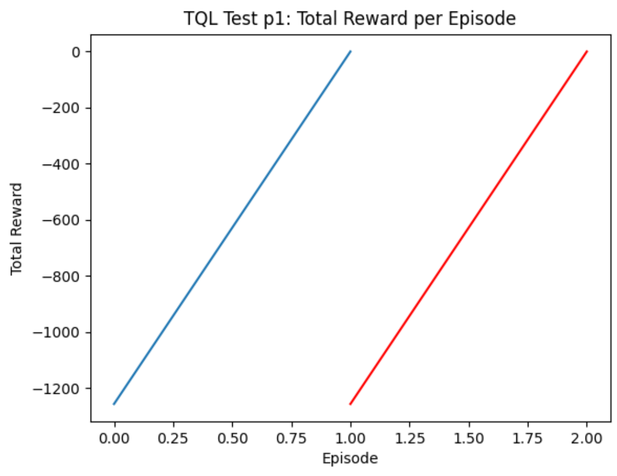


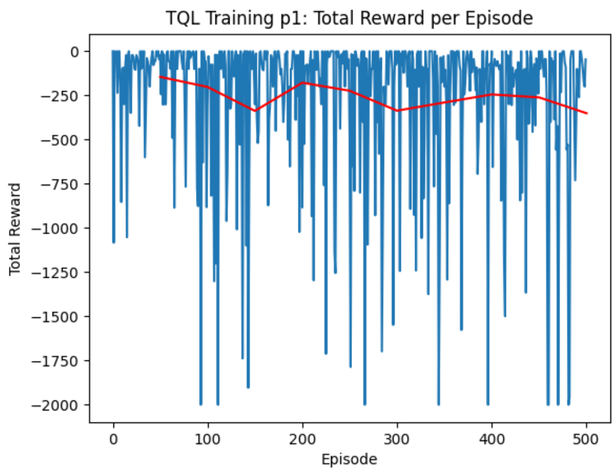
- Configuration for tql_v1:

Number of training episodes=500
Number of test episodes=2
Stability=50
Epsilon=1
Epsilon Decay=0.99642985946637
Epsilon Min=0.2
Alpha=0.3
Gamma=0.9
Step limit for environment reset=17000

RESULTS ->



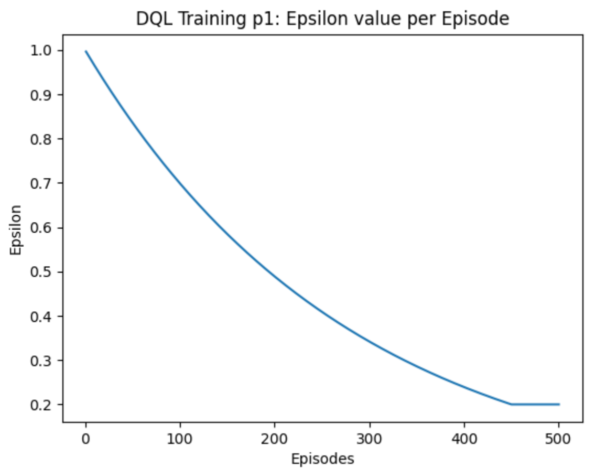


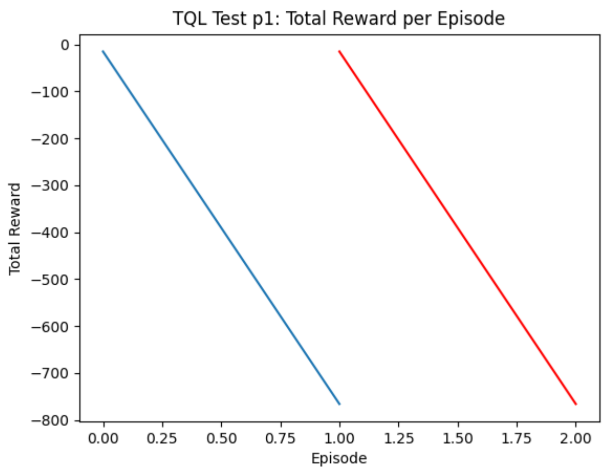


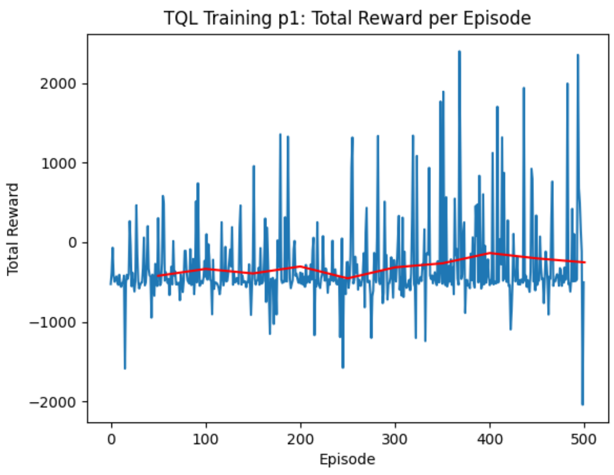
- Configuration for tql_v2:

Number of training episodes=500
Number of test episodes=2
Stability=50
Epsilon=1
Epsilon Decay=0.99642985946637
Epsilon Min=0.2
Alpha=0.3
Gamma=0.9
Step limit for environment reset=17000

RESULTS ->







5.2) Part 2:

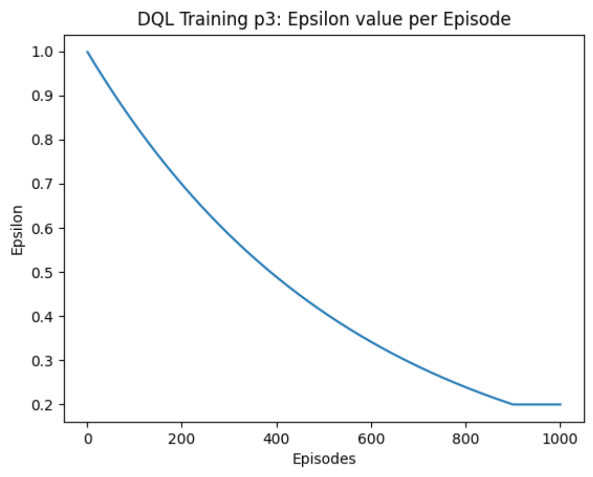
- Everything is reported and attached to “DQL_results_1” and “TQL_results_1” dedicated directory.

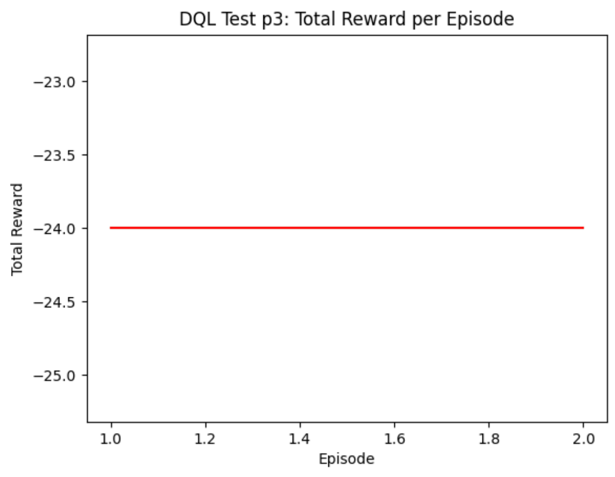
5.3) Part 3:

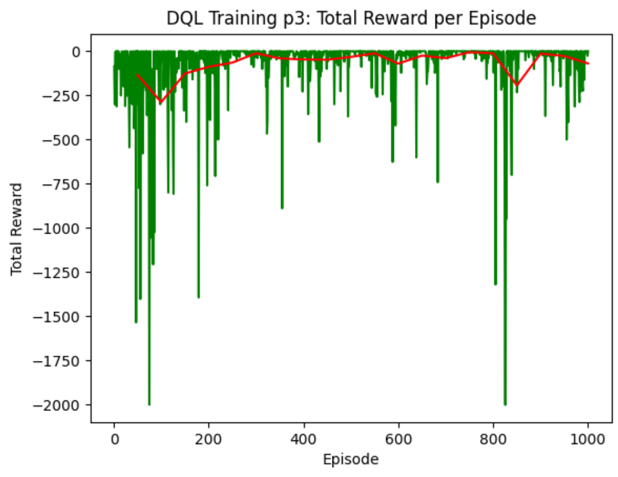
- Configuration for dql_v1:

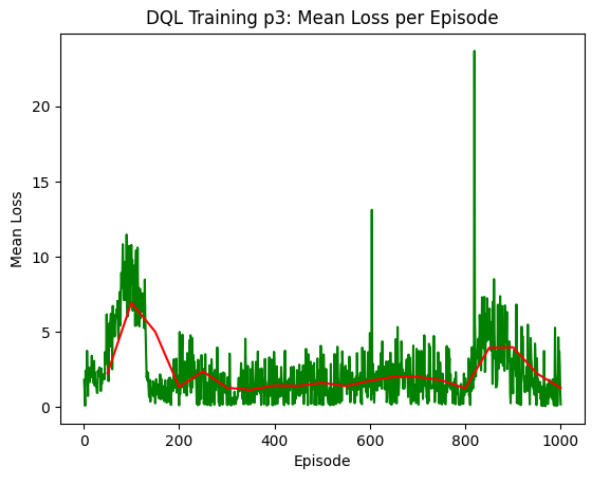
Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9982133336448527
Epsilon Min=0.2
Alpha=0.0001
Gamma=0.9
Replay memory size=100000
Batch size=32
Step limit for environment reset=17000

RESULTS ->





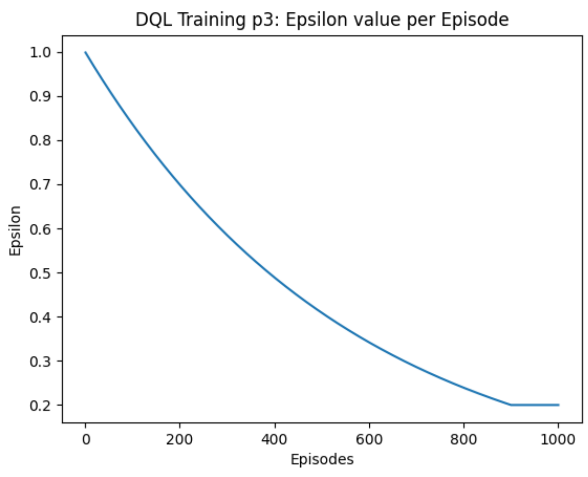


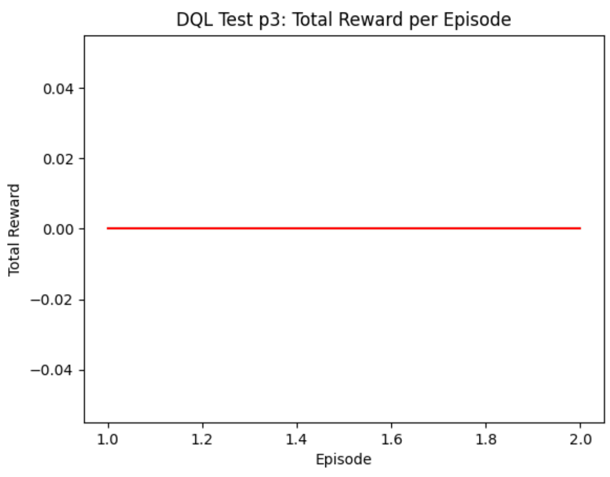


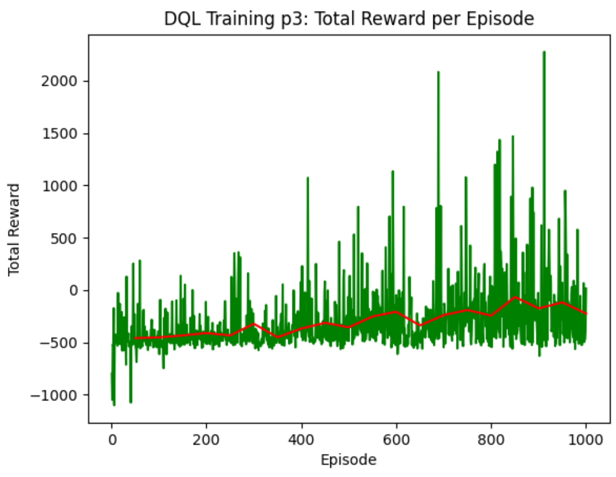
- Configuration for dql_v2:

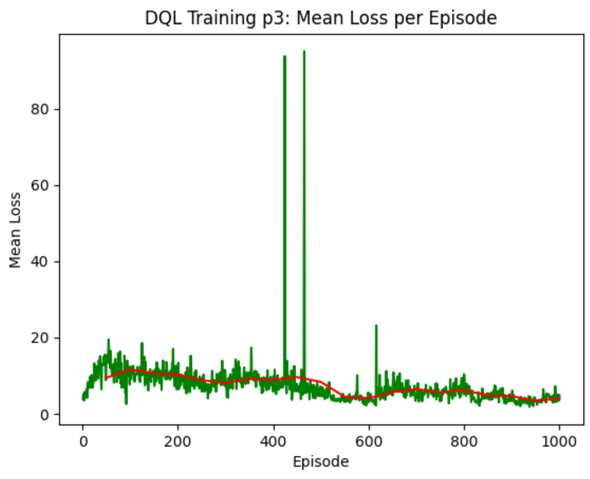
Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9982133336448527
Epsilon Min=0.2
Alpha=0.0001
Gamma=0.9
Replay memory size=100000
Batch size=32
Step limit for environment reset=17000

RESULTS ->





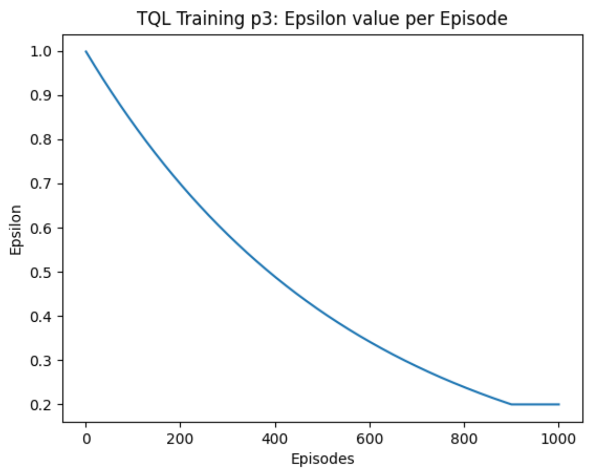


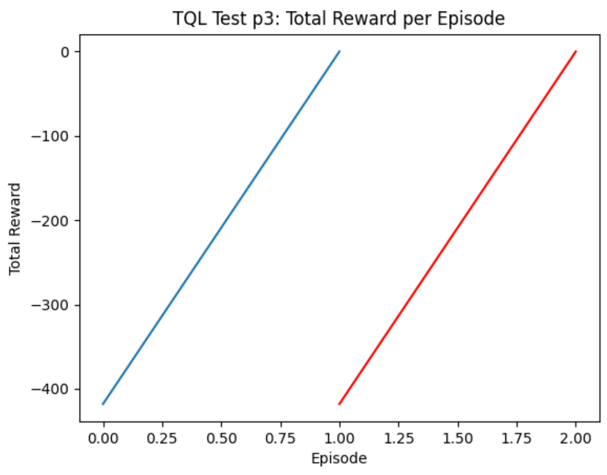


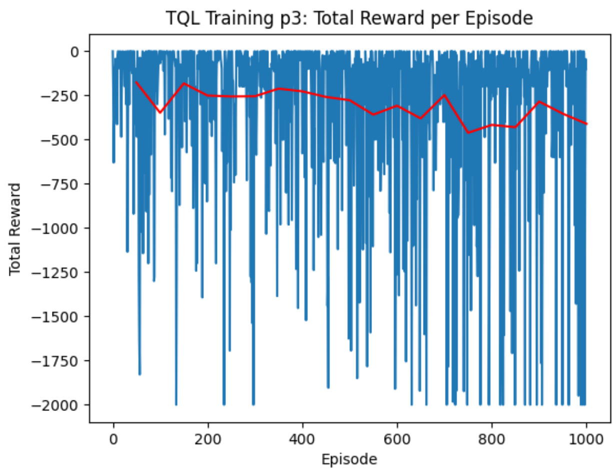
- Configuration for tql_v1:

Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9982133336448527
Epsilon Min=0.2
Alpha=0.3
Gamma=0.9
Step limit for environment reset=17000

RESULTS ->



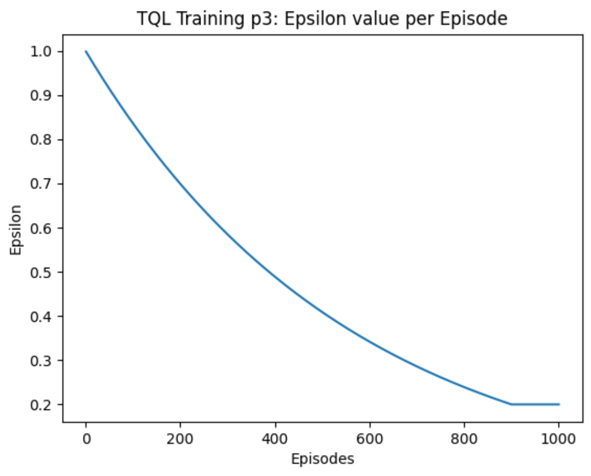


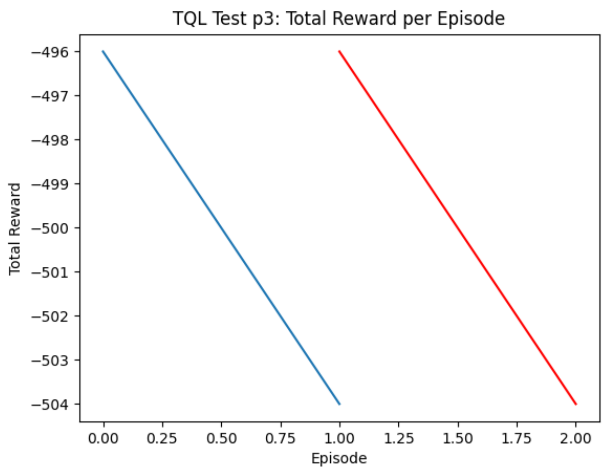


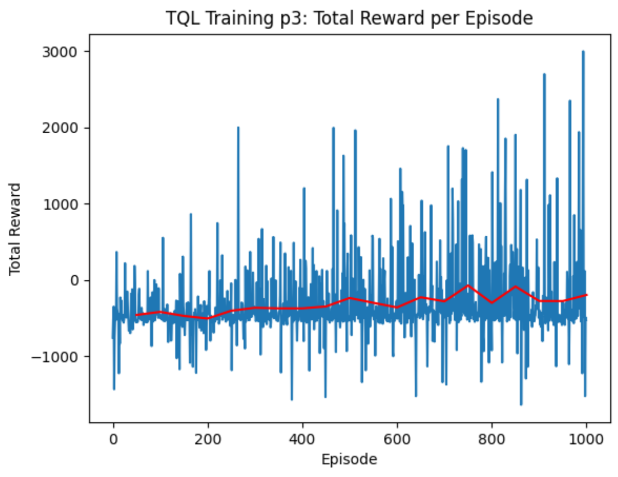
- Configuration for tql_v2:

Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9982133336448527
Epsilon Min=0.2
Alpha=0.3
Gamma=0.9
Step limit for environment reset=17000

RESULTS ->







5.4) Part 4:

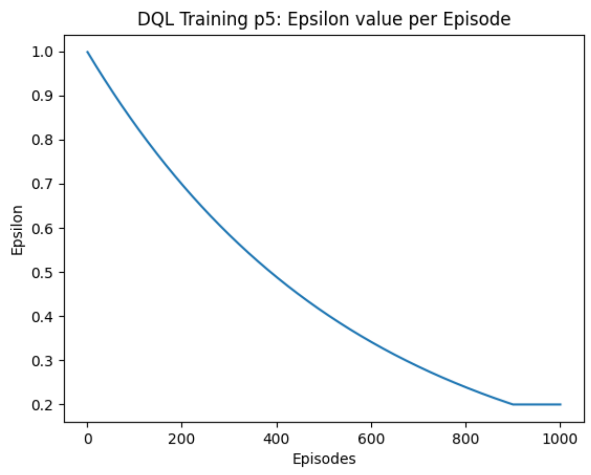
- Everything is reported and attached to the dedicated directory on GitHub.

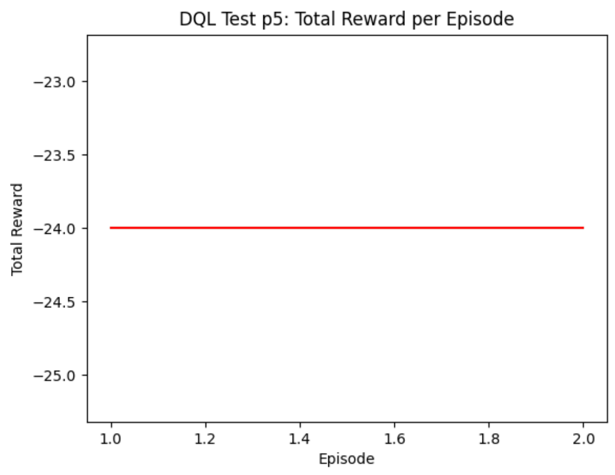
5.5) Part 5:

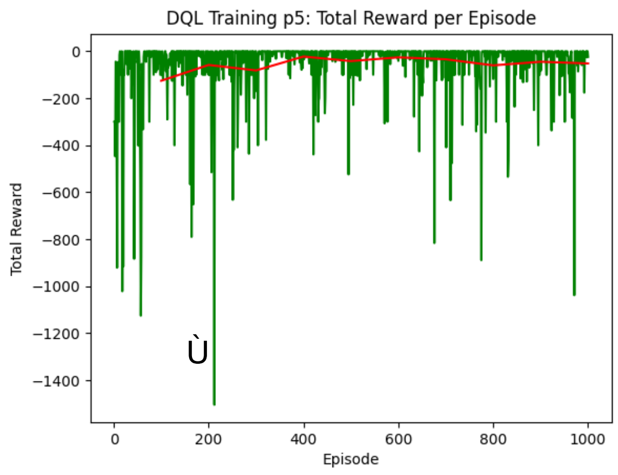
- Configuration for dql_v1:

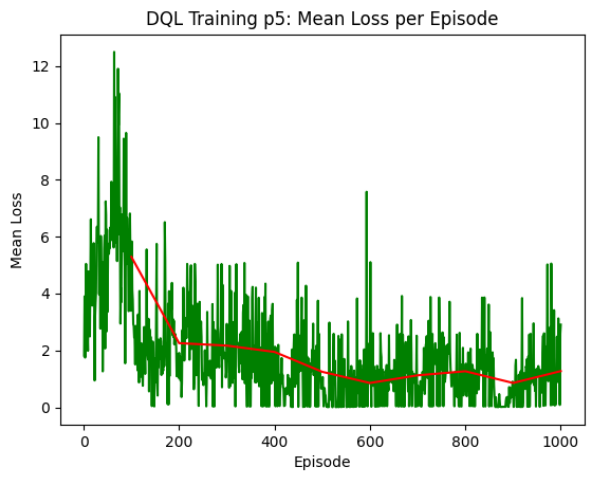
Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9982133336448527
Epsilon Min=0.2
Alpha=0.0001
Gamma=0.9
Replay memory size=100000
Batch size=32
Step limit for environment reset=17000

RESULTS ->





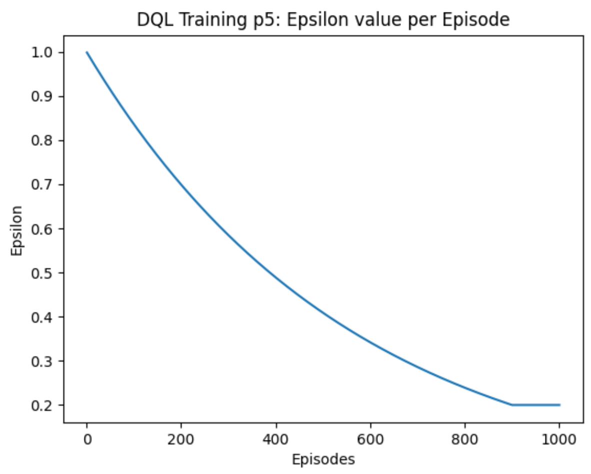


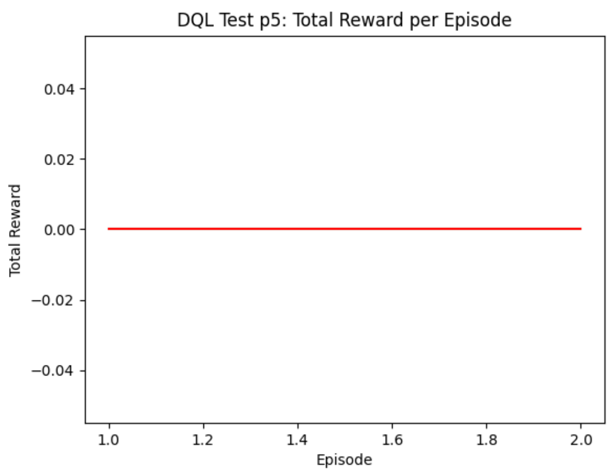


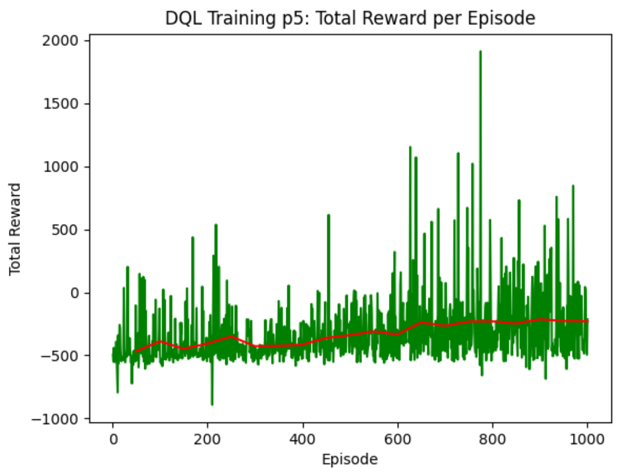
- Configuration for dql_v2:

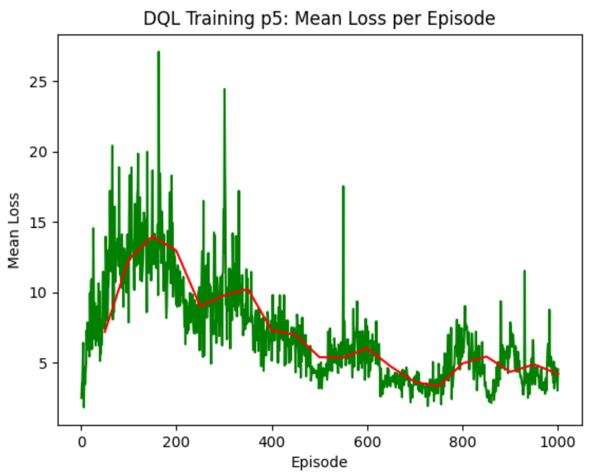
Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9982133336448527
Epsilon Min=0.2
Alpha=0.0001
Gamma=0.9
Replay memory size=100000
Batch size=32
Step limit for environment reset=17000

RESULTS ->





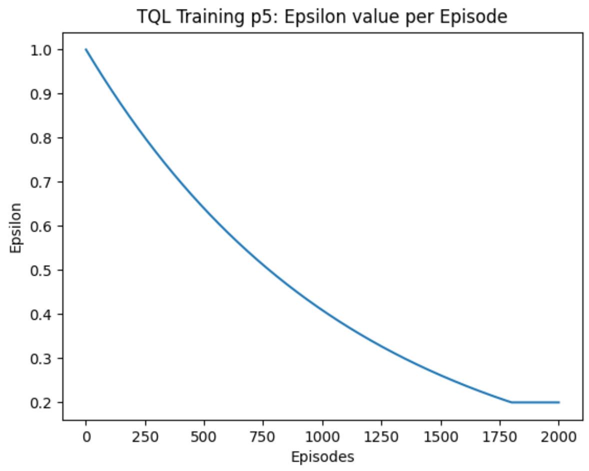


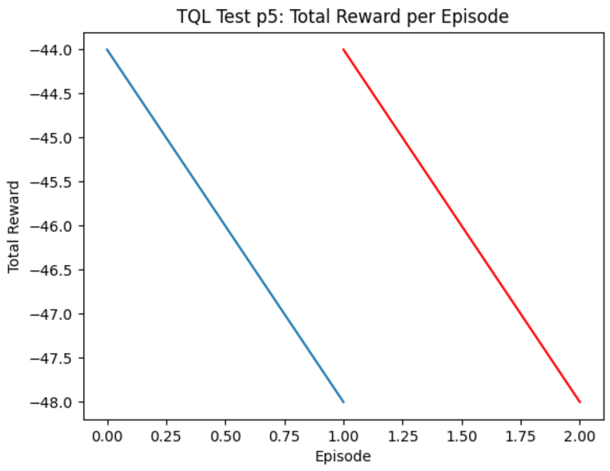


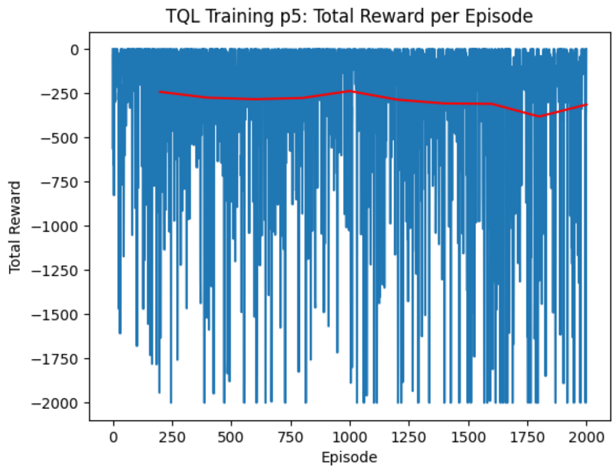
- Configuration for tql_v1:

Number of training episodes=2000
Number of test episodes=2
Stability=200
Epsilon=1
Epsilon Decay=0.9991062674434851
Epsilon Min=0.2
Alpha=0.3
Gamma=0.9
Step limit for environment reset=17000

RESULTS ->



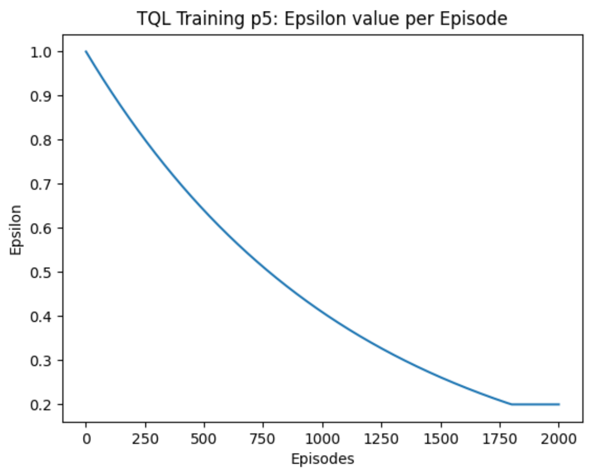


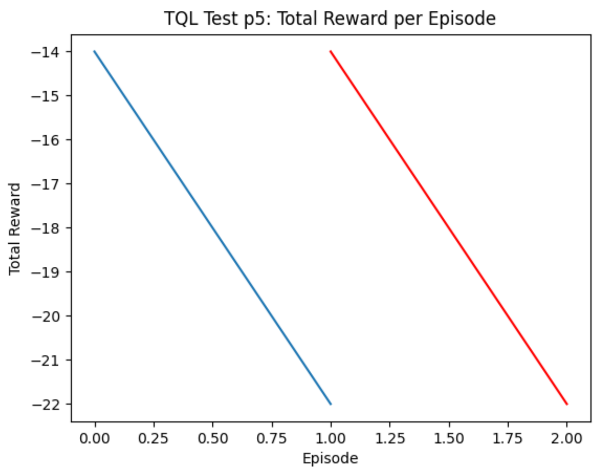


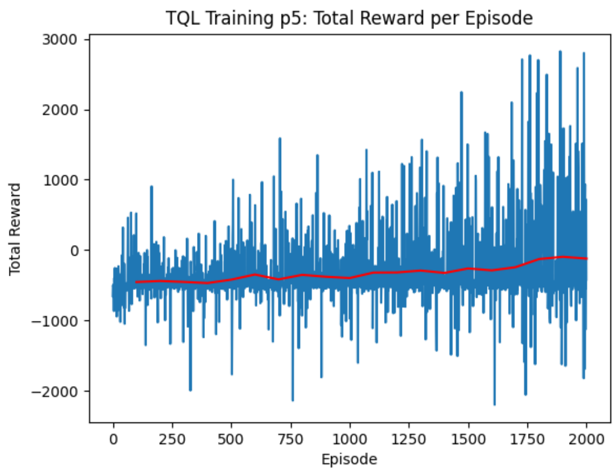
- Configuration for tql_v2:

Number of training episodes=2000
Number of test episodes=2
Stability=200
Epsilon=1
Epsilon Decay=0.9991062674434851
Epsilon Min=0.2
Alpha=0.3
Gamma=0.9
Step limit for environment reset=17000

RESULTS ->







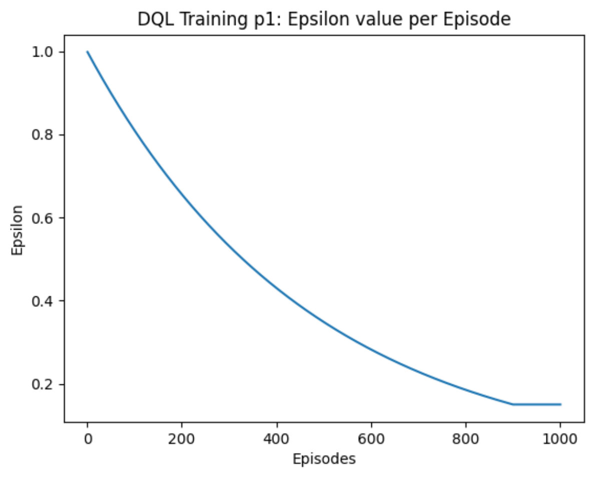
6) Training 2 performances

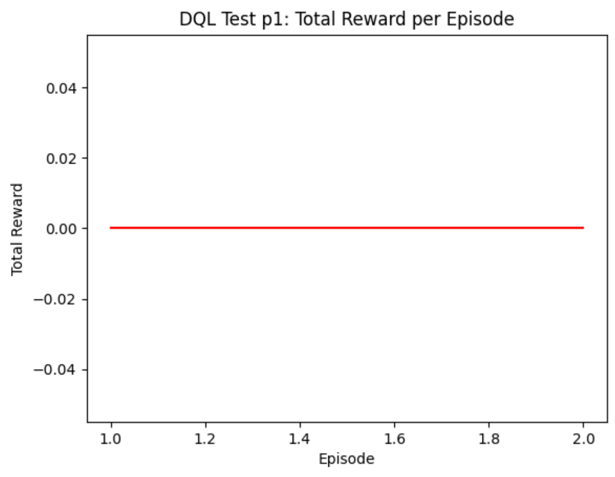
6.1) Part 1 (only dql, tql on GitHub):

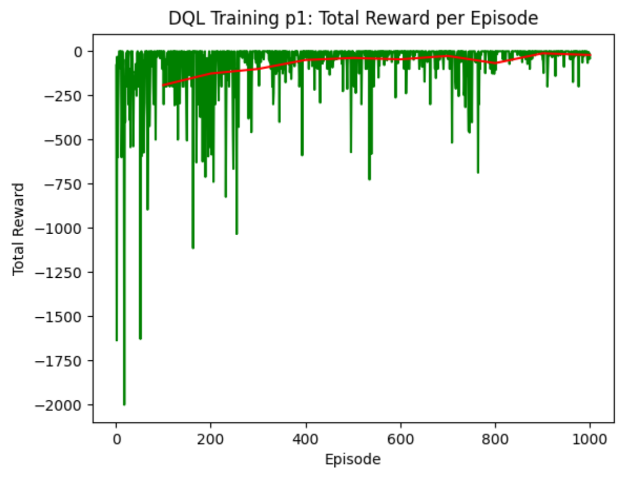
- Configuration for dql_v1:

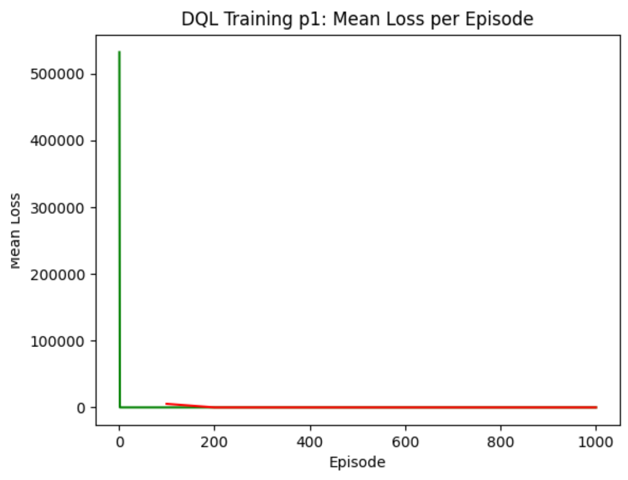
Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9978943089900857
Epsilon Min=0.15
Alpha=0.01
Gamma=0.9
Replay memory size=100000
Batch size=32
Step limit for environment reset=17000

RESULTS ->





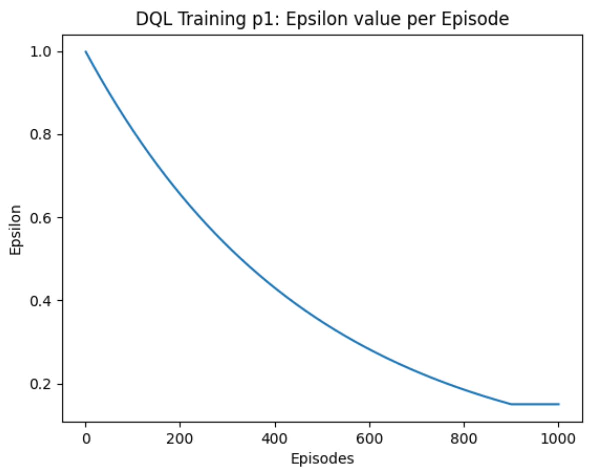


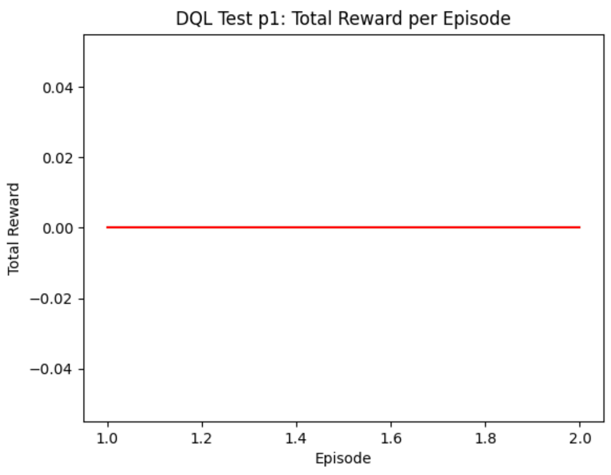


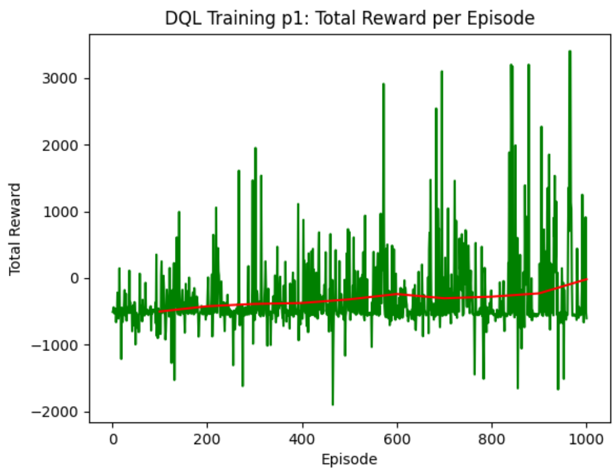
- Configuration for dql_v2:

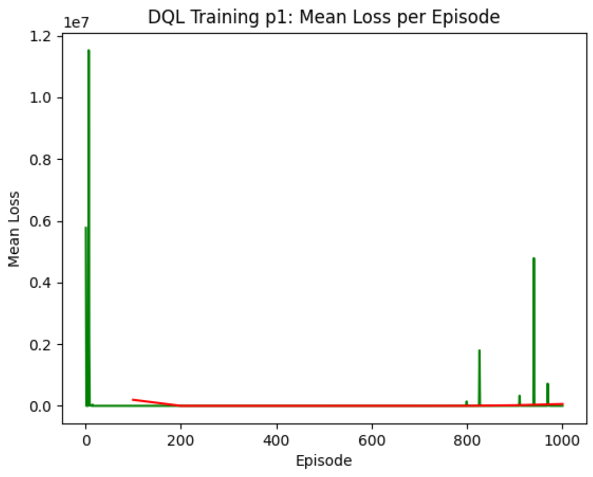
Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9978943089900857
Epsilon Min=0.15
Alpha=0.01
Gamma=0.9
Replay memory size=100000
Batch size=32
Step limit for environment reset=17000

RESULTS ->







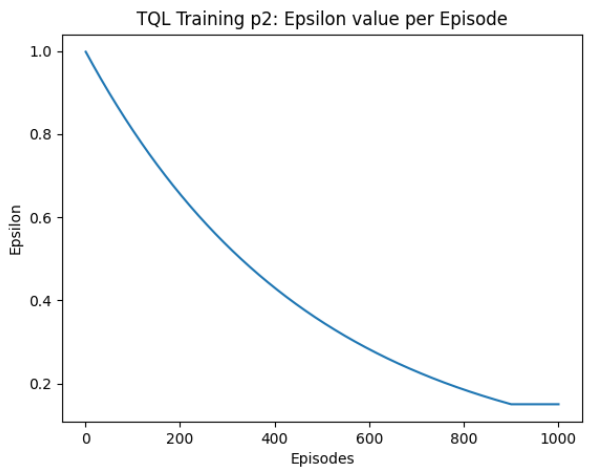


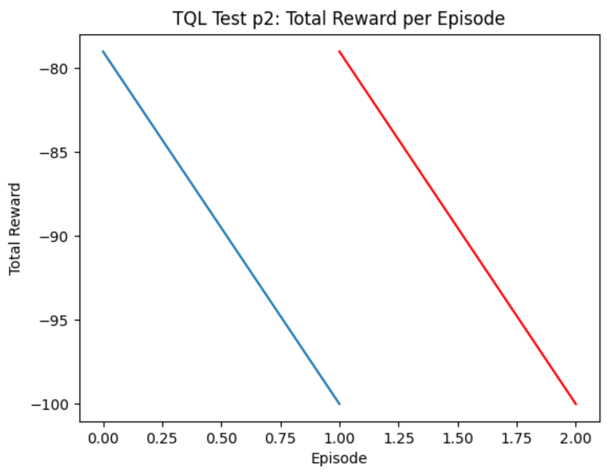
6.2) Part 2 (only tql, dql on GitHub):

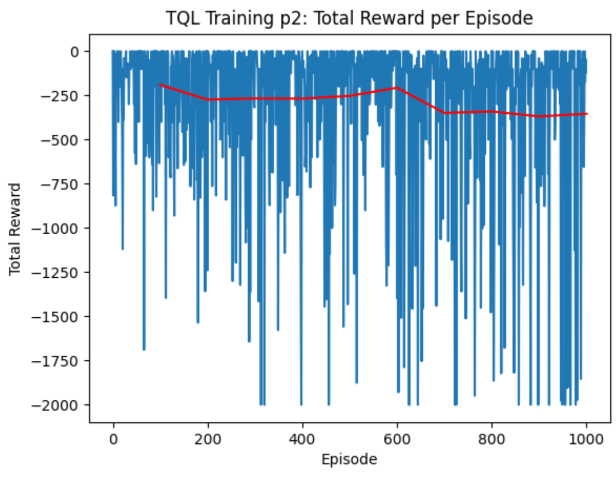
- Configuration for tql_v1:

Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9978943089900857
Epsilon Min=0.15
Alpha=0.7
Gamma=0.7
Step limit for environment reset=17000

RESULTS ->



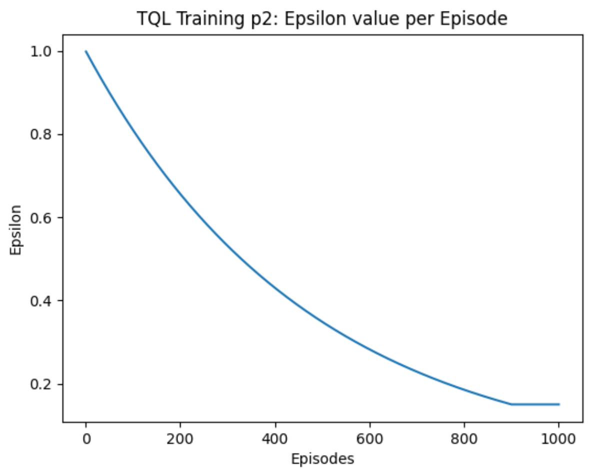


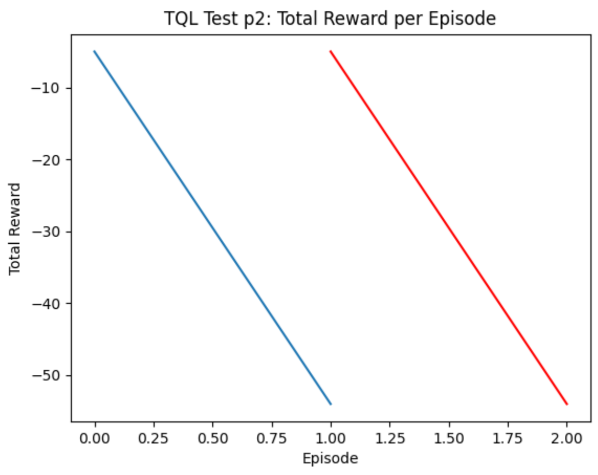


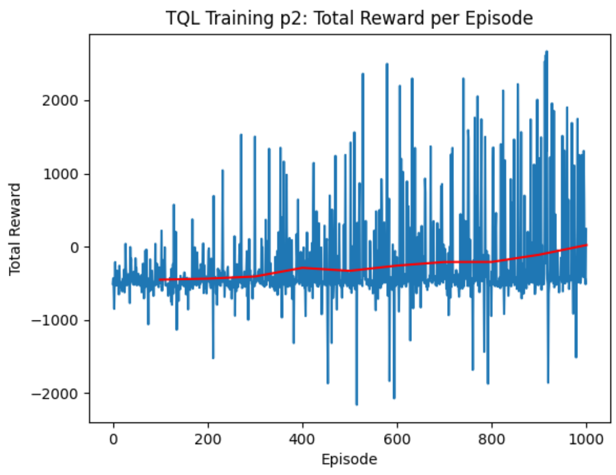
- Configuration for tql_v2:

Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9978943089900857
Epsilon Min=0.15
Alpha=0.7
Gamma=0.7
Step limit for environment reset=17000

RESULTS ->





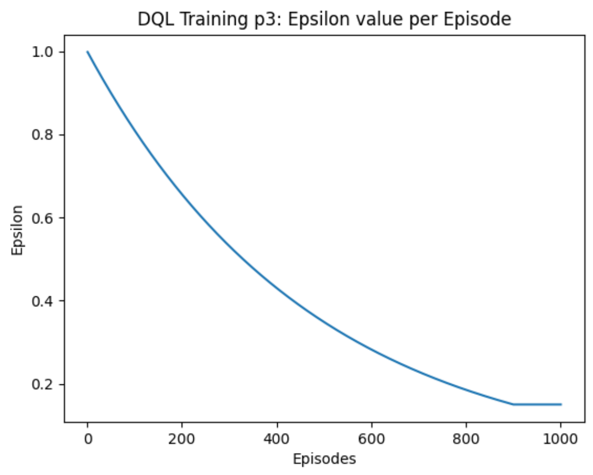


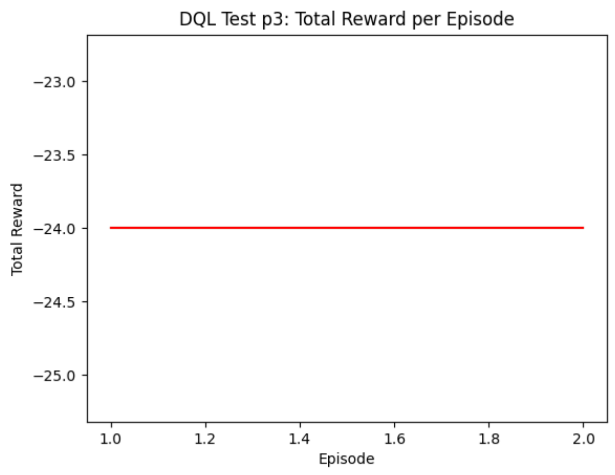
6.3) Part 3 (only dql, tql on directories):

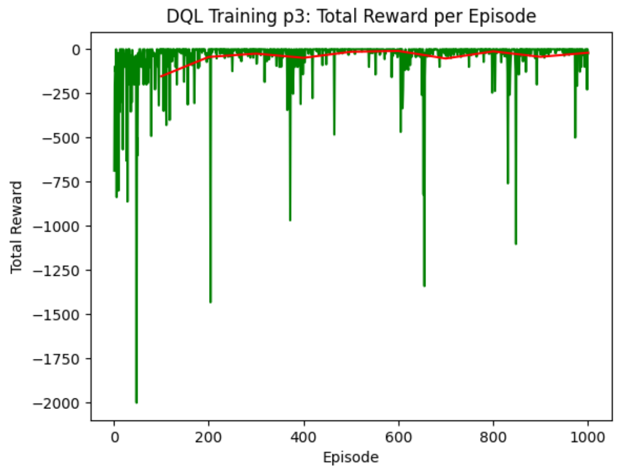
- Configuration for dql_v1:

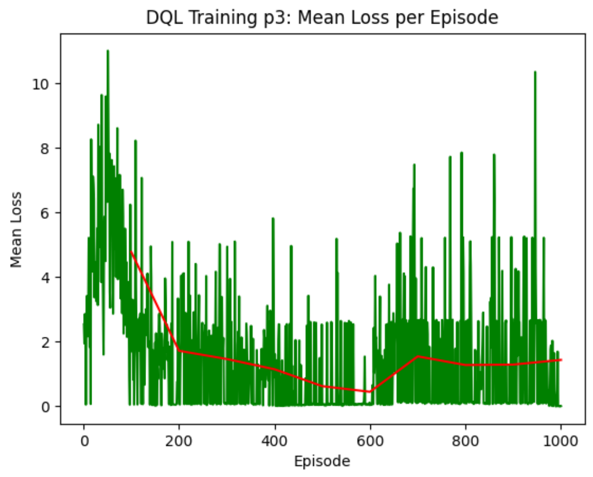
Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9978943089900857
Epsilon Min=0.15
Alpha=0.0001
Gamma=0.7
Replay memory size=100000
Batch size=16
Step limit for environment reset=17000

RESULTS ->





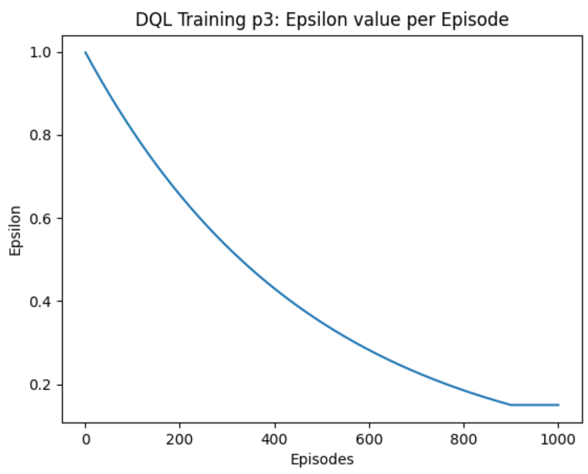


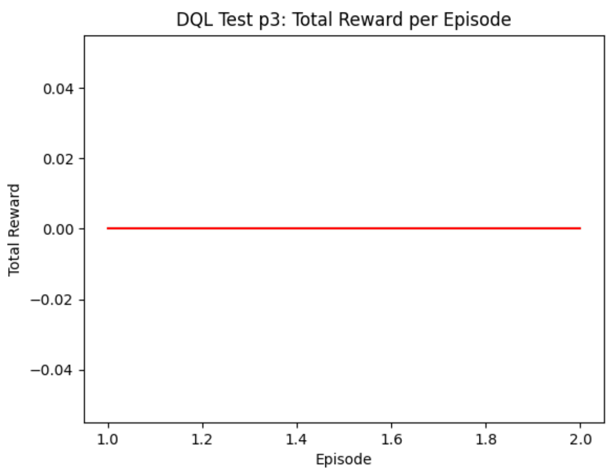


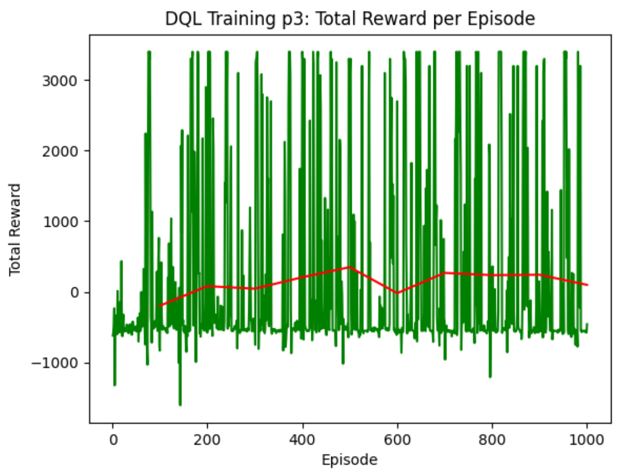
- Configuration for dql_v2:

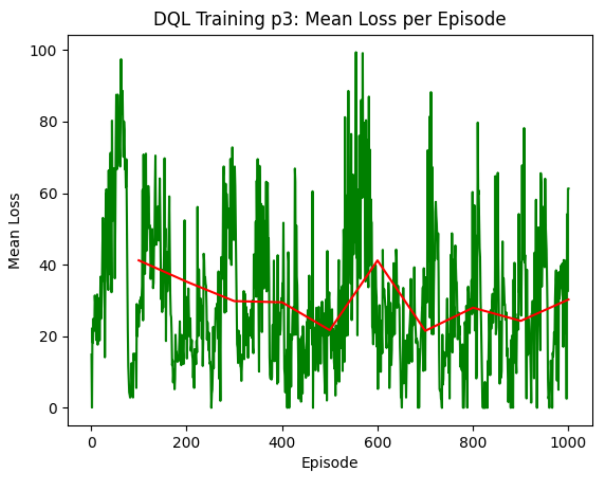
Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9978943089900857
Epsilon Min=0.15
Alpha=0.0001
Gamma=0.7
Replay memory size=100000
Batch size=16
Step limit for environment reset=17000

RESULTS ->







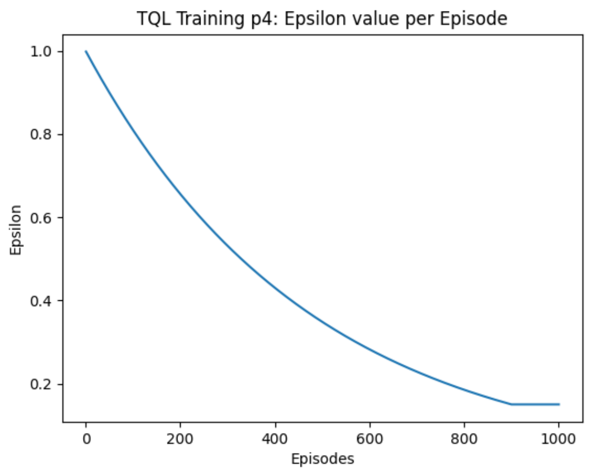


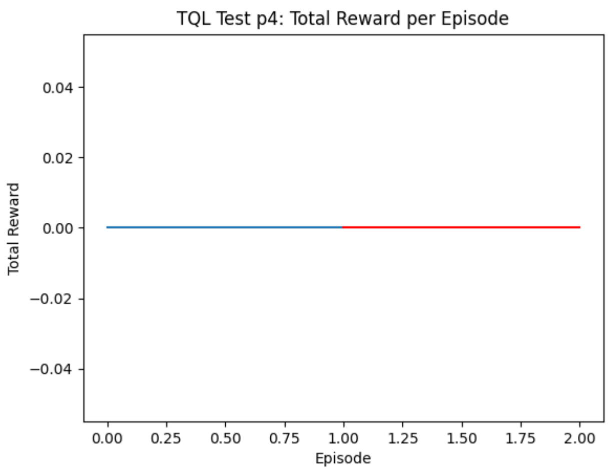
6.4) Part 4 (only tql):

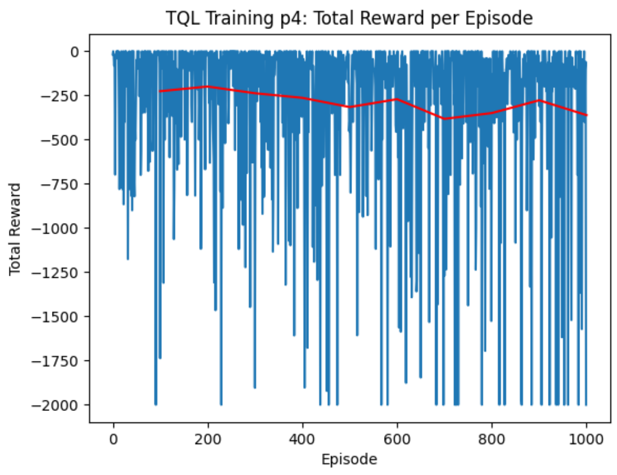
- Configuration for tql_v1:

Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9978943089900857
Epsilon Min=0.15
Alpha=0.3
Gamma=0.3
Step limit for environment reset=17000

RESULTS ->



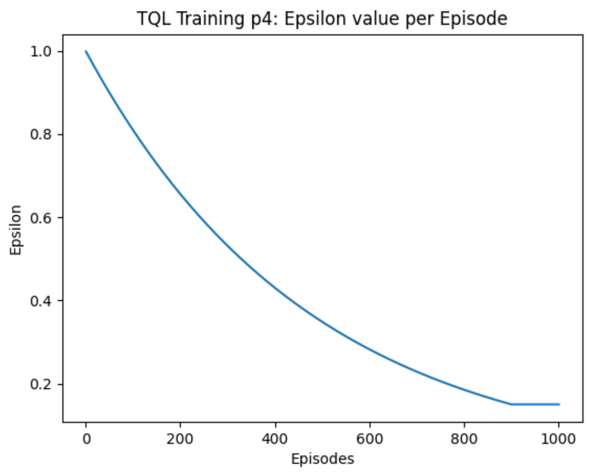


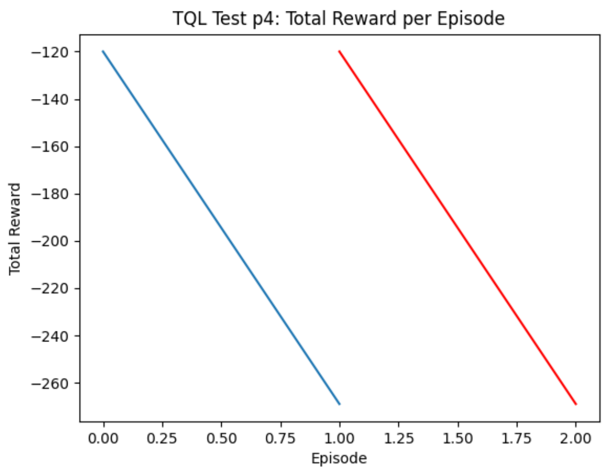


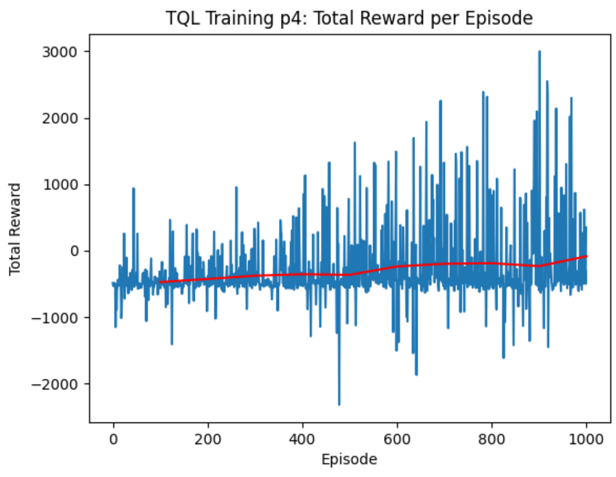
- Configuration for tql_v2:

Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9978943089900857
Epsilon Min=0.15
Alpha=0.3
Gamma=0.3
Step limit for environment reset=17000

RESULTS ->







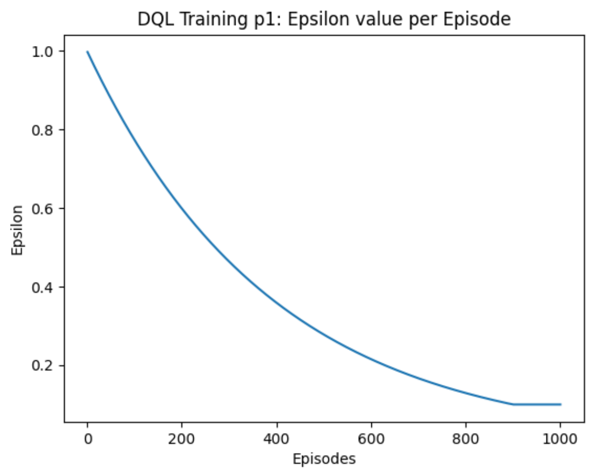
7) Training 3 performances

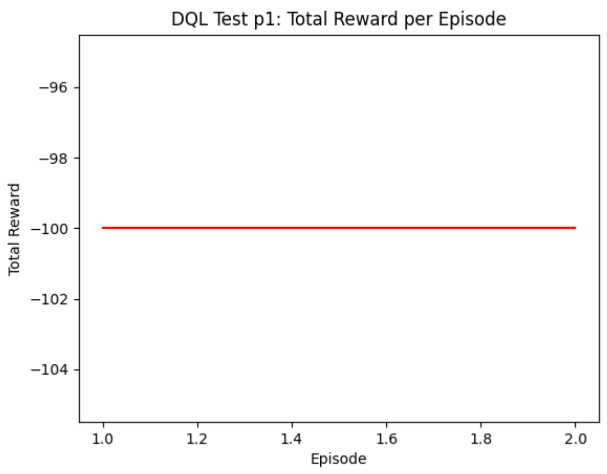
7.1) Part 1:

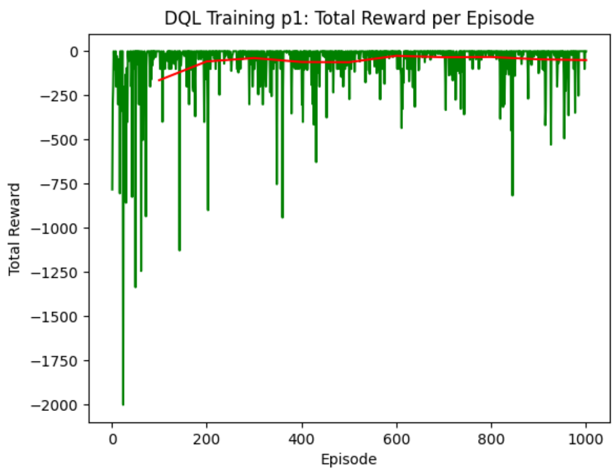
- Configuration for dql_v1:

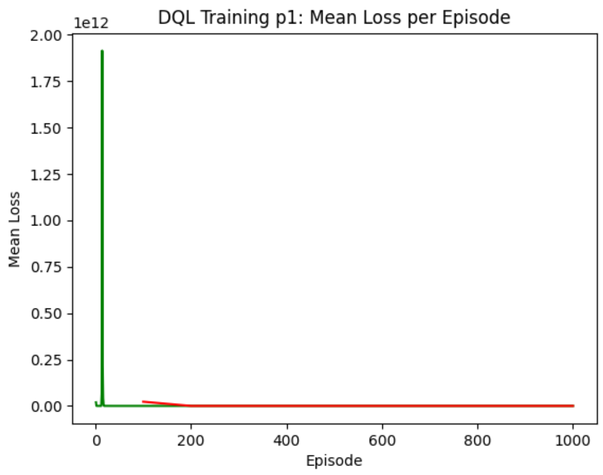
Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay = 0.9974448421062369
Epsilon Min=0.1
Alpha=0.001
Gamma=0.99
Replay memory size=100000
Batch size=16
Step limit for environment reset=17000

RESULTS ->





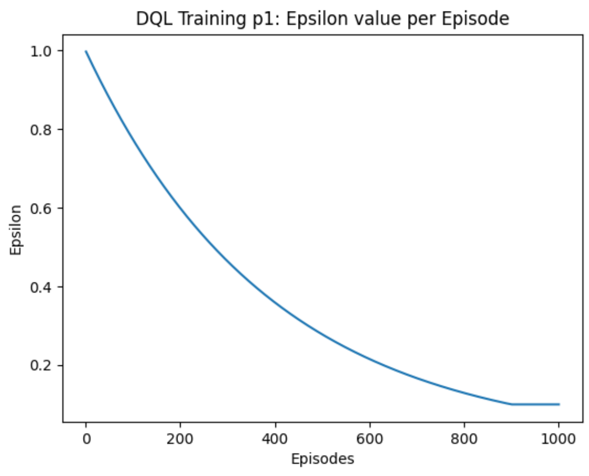


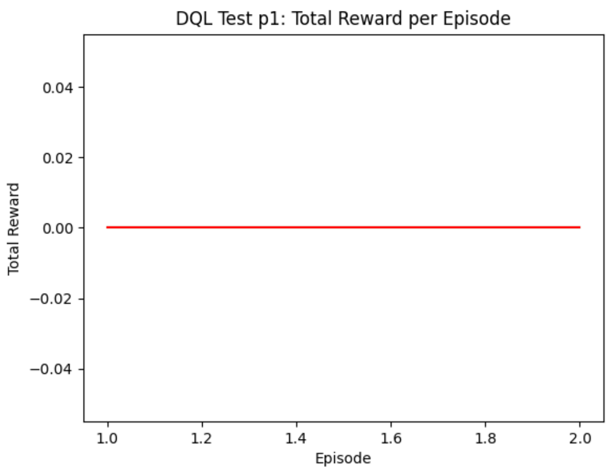


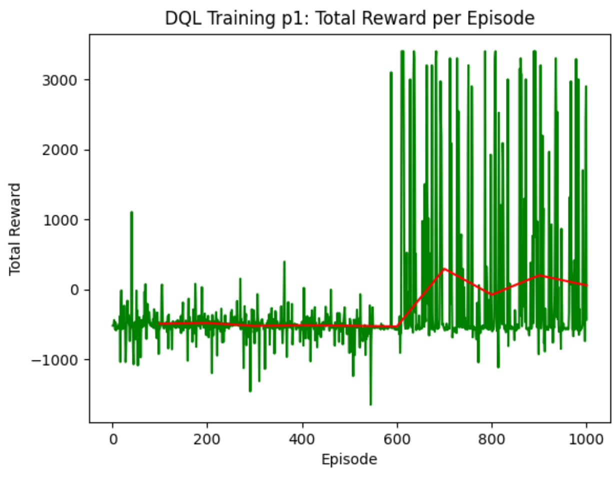
- Configuration for dql_v2:

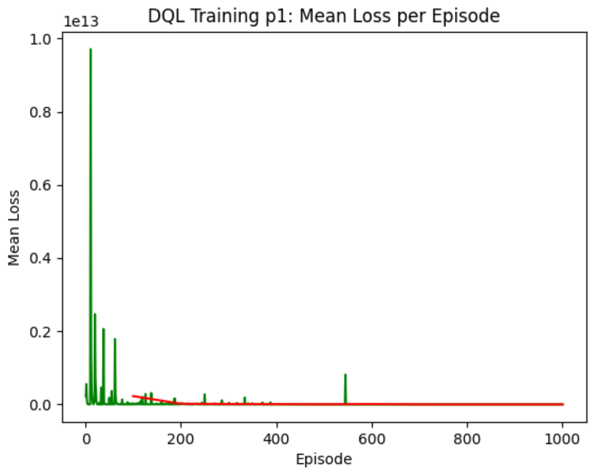
Number of training episodes=1000
Number of test episodes=2
Stability =100
Epsilon=1
Epsilon Decay = 0.9974448421062369
Epsilon Min=0.1
Alpha=0.001
Gamma=0.99
Replay memory size=100000
Batch size=16
Step limit for environment reset=17000

RESULTS ->





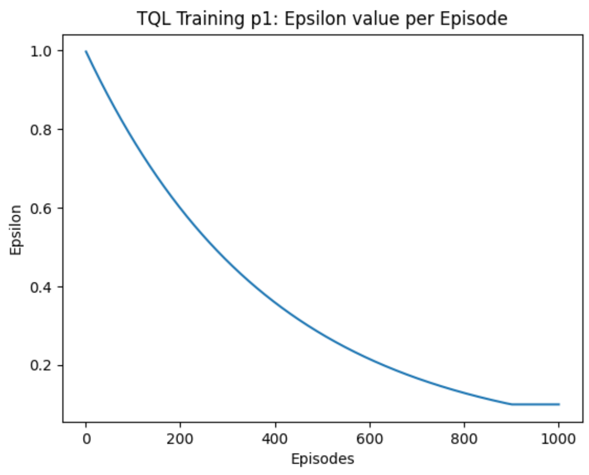


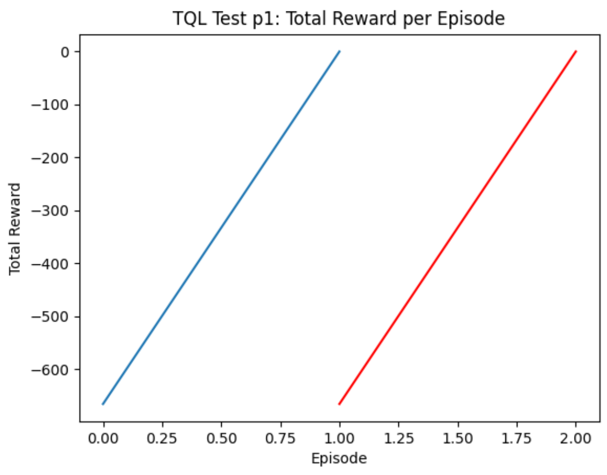


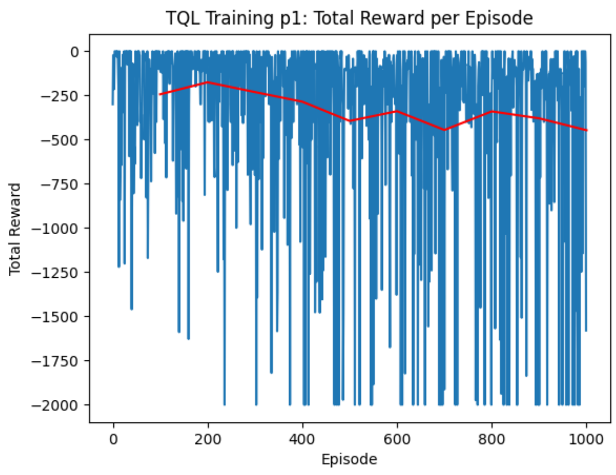
- Configuration for tql_v1:

Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay = 0.9974448421062369
Epsilon Min=0.1
Alpha=0.99
Gamma=0.99
Step limit for environment reset=17000

RESULTS ->



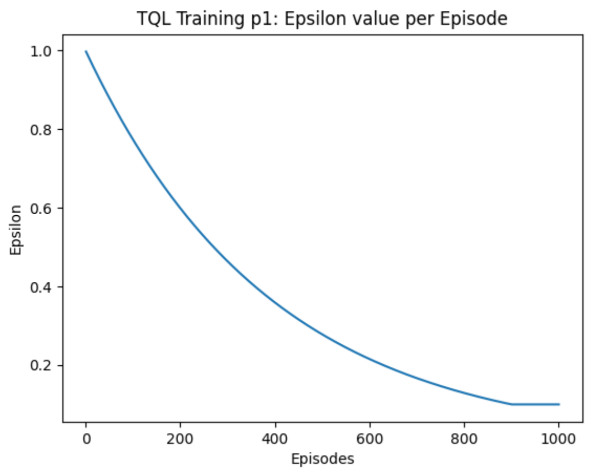


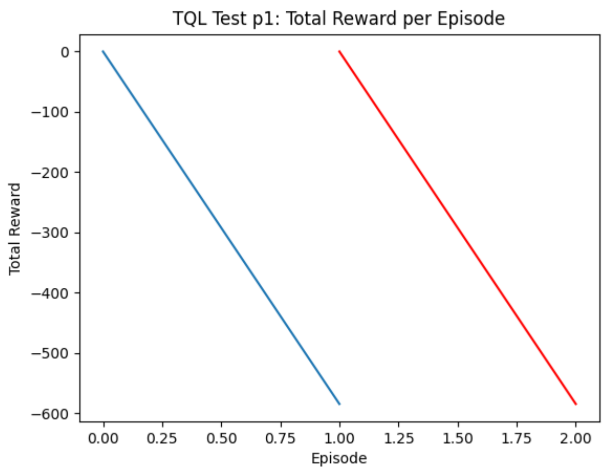


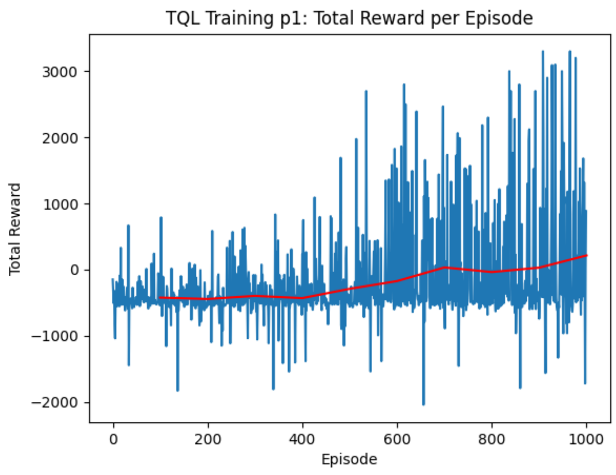
- Configuration for tql_v2:

Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9974448421062369
Epsilon Min=0.1
Alpha=0.99
Gamma=0.99
Step limit for environment reset=17000

RESULTS ->





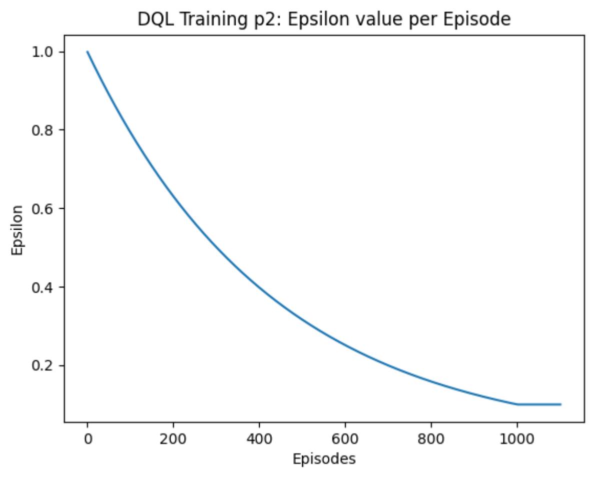


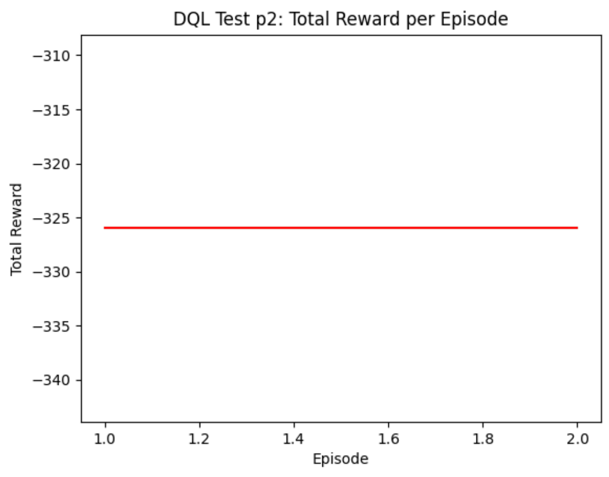
7.2) Part 2 (only dql_v1, rest on GitHub):

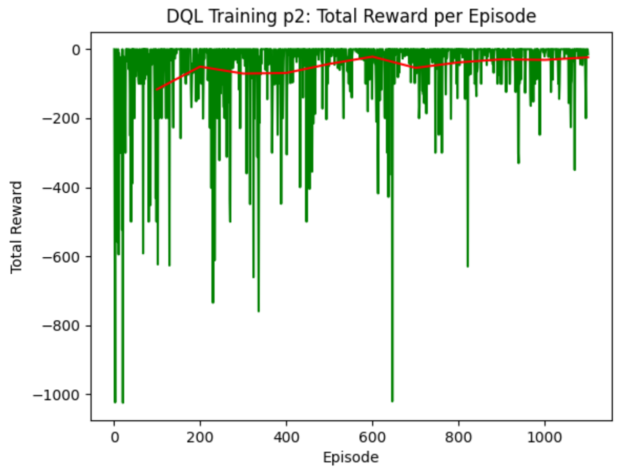
- Configuration for dql_v1:

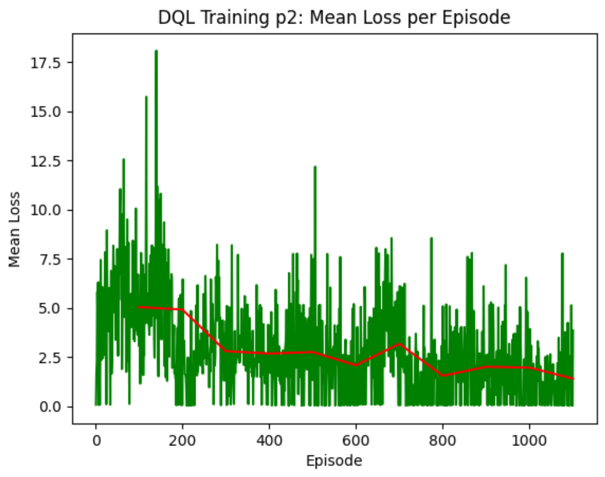
Number of training episodes=1100
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9760536373079002
Epsilon Min=0.1
Alpha=0.001
Gamma=0.99
Replay memory size=100000
Batch size=16
Step limit for environment reset=17000

RESULTS ->







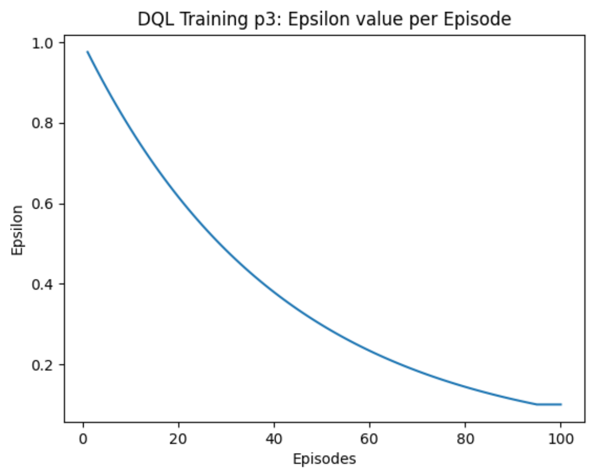


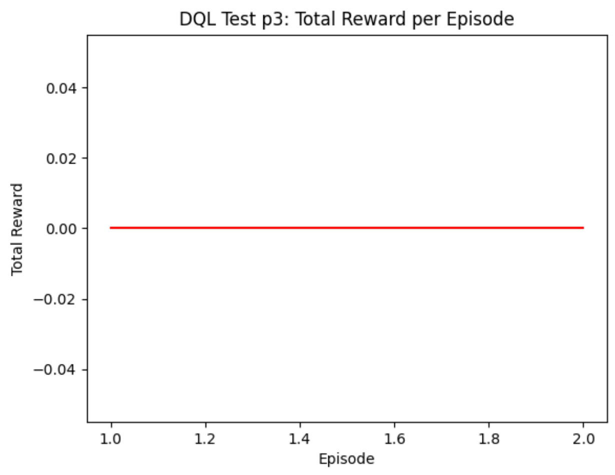
7.3) Part 3(all, except dql_v1):

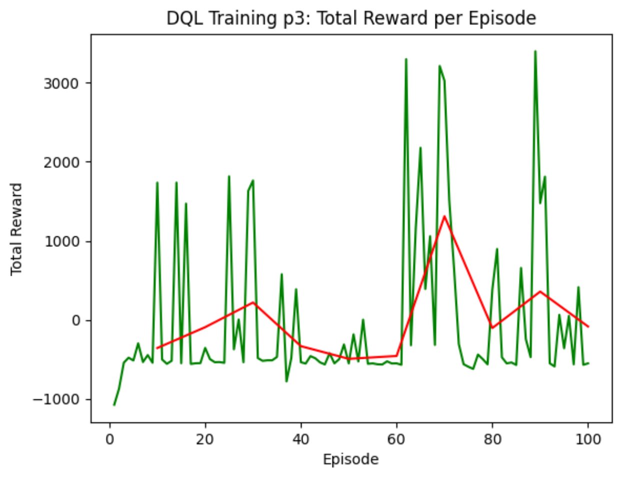
- Configuration for dql_v2:

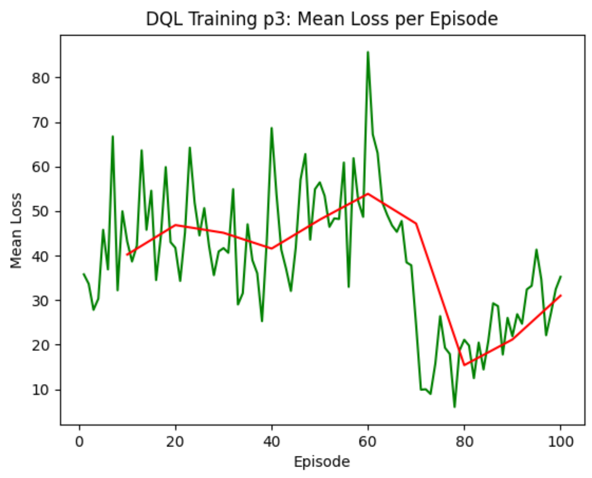
Number of training episodes=100
Number of test episodes=2
Stability=5
Epsilon=1
Epsilon Decay=0.9760536373079002
Epsilon Min=0.1
Alpha=0.001
Gamma=0.99
Replay memory size=100000
Batch size=32
Step limit for environment reset=17000

RESULTS ->





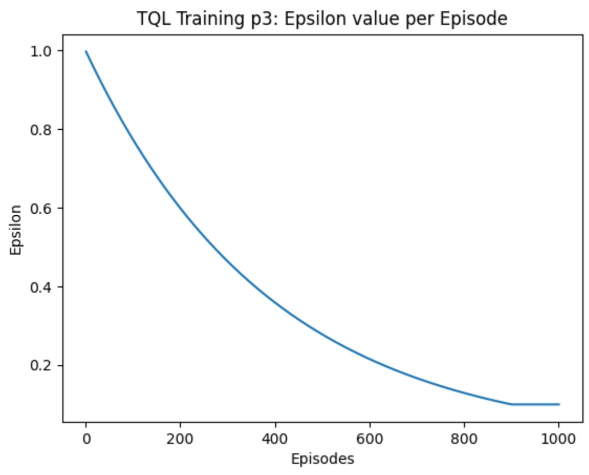


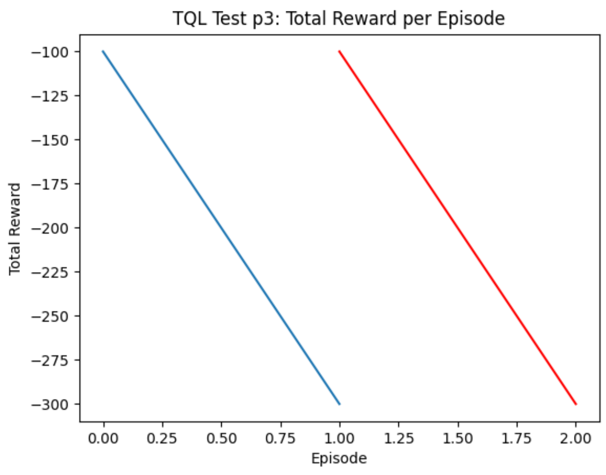


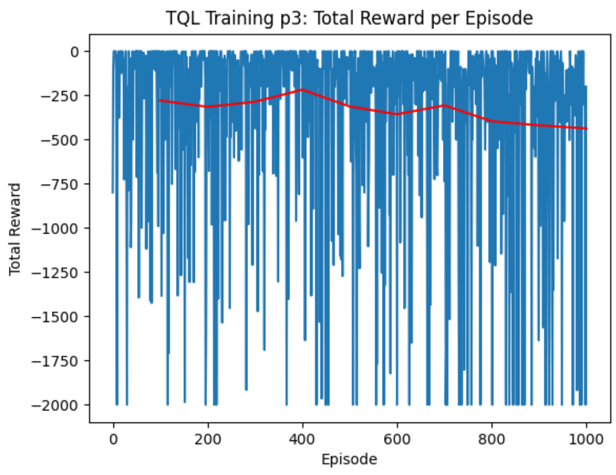
- Configuration for tql_v1:

Number of training episodes=1000
Number of test episodes=2
Stability=100
Epsilon=1
Epsilon Decay=0.9974448421062369
Epsilon Min=0.1
Alpha=0.99
Gamma=0.99
Step limit for environment reset=17000

RESULTS ->



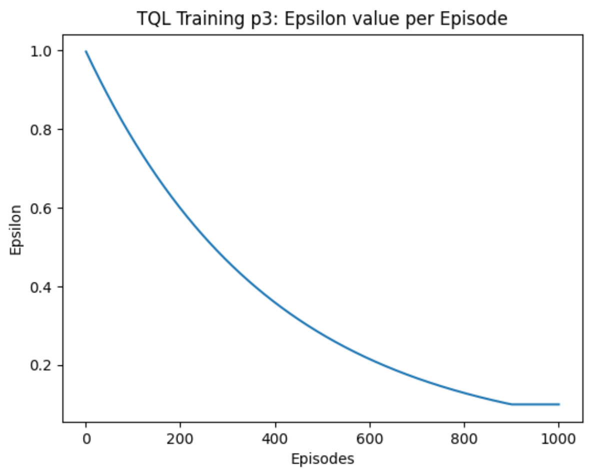


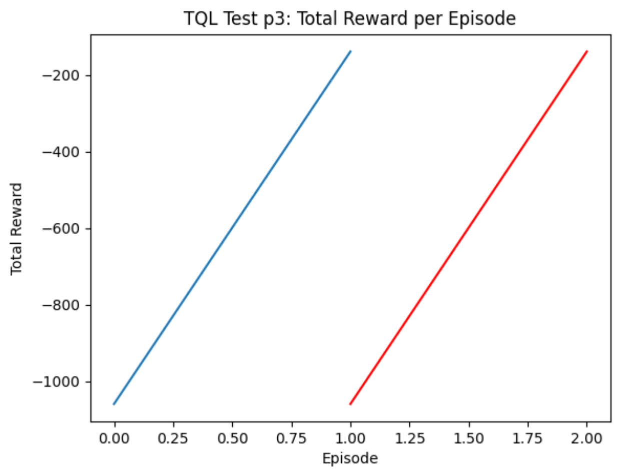


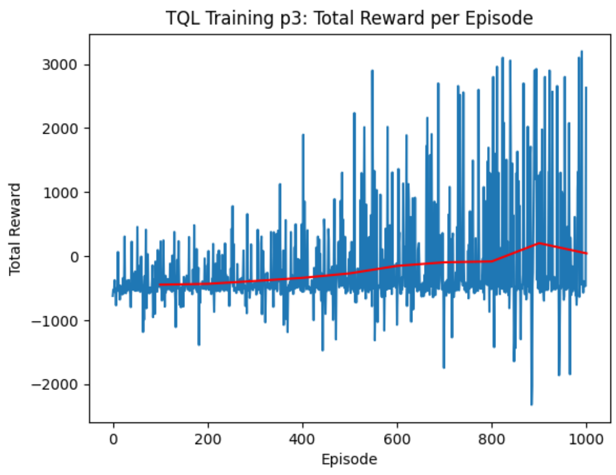
- Configuration for tql_v2:

Number of training episodes=1000
Number of test episodes=2
Stability =100
Epsilon=1
Epsilon Decay = 0.9974448421062369
Epsilon Min=0.1
Alpha=0.99
Gamma=0.99
Step limit for environment reset=17000

RESULTS ->


TQL Training p3: Epsilon value per Episode
Epsilon
Episodes


TQL Test p3: Total Reward per Episode
Total Reward
Episode


TQL Training p3: Total Reward per Episode
Total Reward
Episode