COL778
Assignment 3
Q-Learning

Satyam Kumar Modi 2019CS50448
Vaibhav Seth 2021MT10236

7th April 2024

Contents

1 Introduction

In this problem, we are operating in a grid world environment where we have a car and there are holes on the grid. the goal is to reach the goal state from the start state. All the experiments presented here use Q-learning as a backbone.

2 Tabular Q-Learning

We implemented the standard tabular Q-Learning algorithm:

Algorithm:

Start with $Q_0(s, a)$ for all s, a.
Get initial state s
For k = 1, 2, ... till convergence
  Sample action a, get next state s'
  If s' is terminal:
    $\text{target} = R(s, a, s')$
    Sample new initial state s'
  else:
    $\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
  $Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha\,[\text{target}]$
  $s \leftarrow s'$

Figure 1: Algorithm: Tabular Q Learning

We explore multiple exploration-exploitation strategies based on the epsilon-greedy approach. We varied the epsilon using a scheduler to maintain high levels of exploration in the beginning and gradually reducing it over time to learn a robust policy.

The final scheduler function that we used for Tabular Q-Learning is:

$$r = max(\frac{N - episode}{N}, 0)$$

$$\epsilon = (\epsilon_i - \epsilon_f) * r + \epsilon_f$$

Where N decided how fast we reduce the learning rate. A larger N means slower rate of change of learning rate and vice-versa.
ϵ_i is the initial learning rate and ϵ_f is the final learning rate.

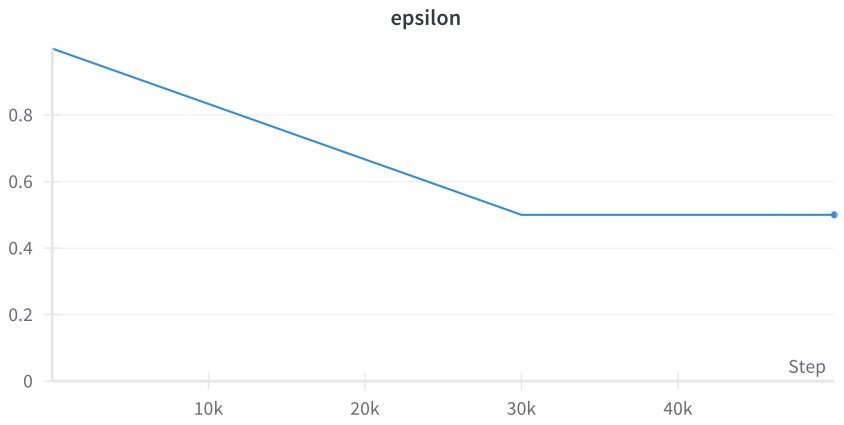

Figure 2: Example epsilon schedule with $\epsilon_i = 1$, $\epsilon_f = 0.5$ and $N = 30k$

2.1 Small Environment

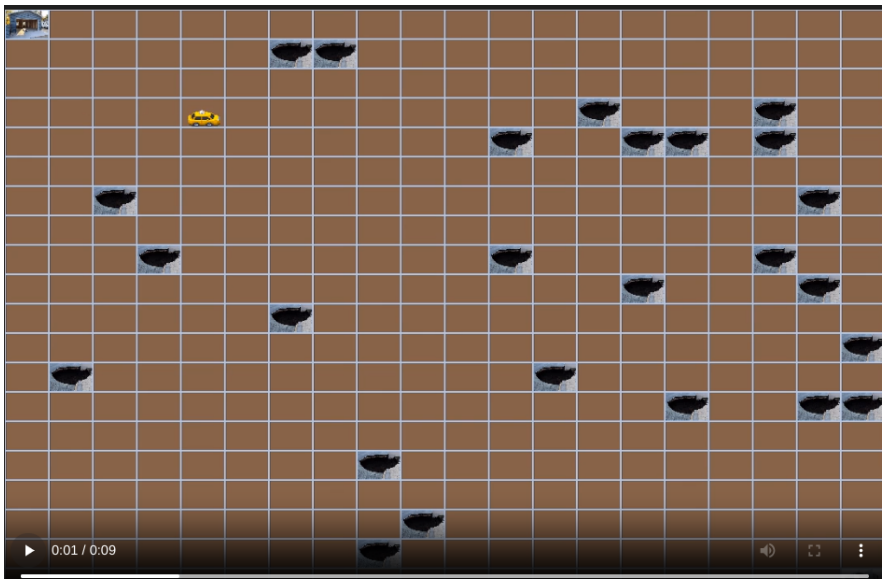

Figure 3: 20 x 20 environment

We conducted multiple experiments with different learning rate schedules and we present the Q-value and Action plot here. The best learning rate we found was 0.4

2.1.1 Case 1

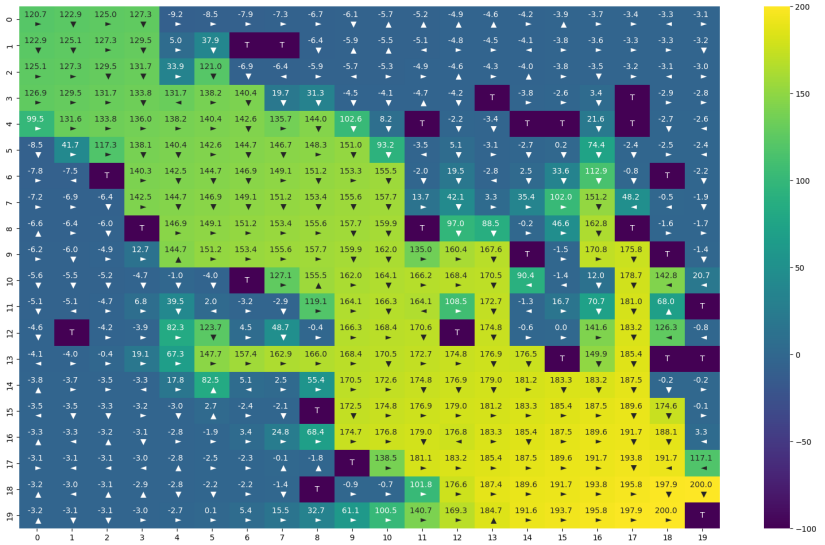

Figure 4: Policy Value Map $\epsilon_i = 1$, $\epsilon_f = 0.1$, $N = 0.4k$, Total $= 2k$

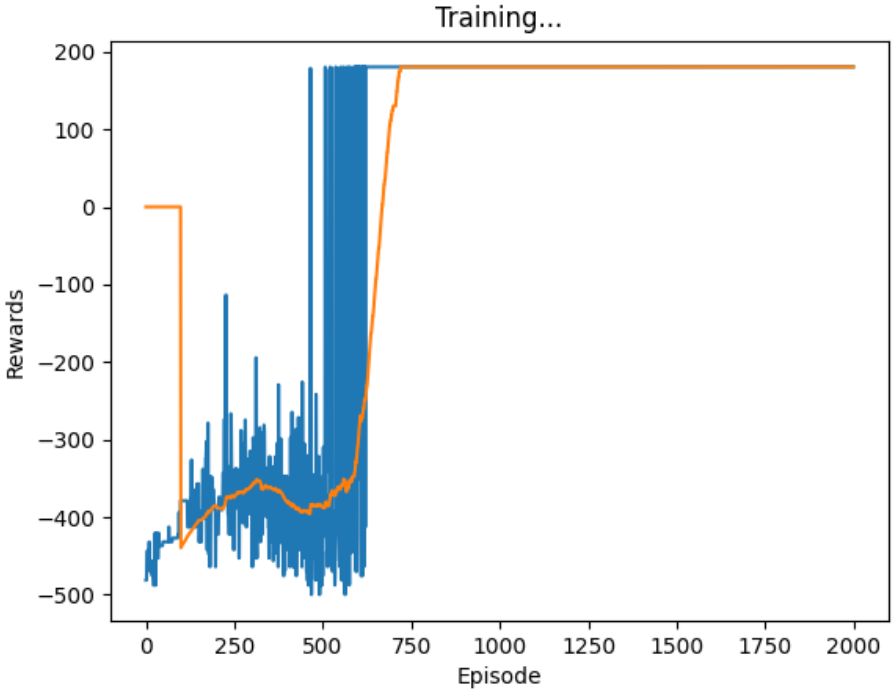

Figure 5: Rewards $\epsilon_i = 1$, $\epsilon_f = 0.1$, $N = 0.4k$, Total $= 2k$

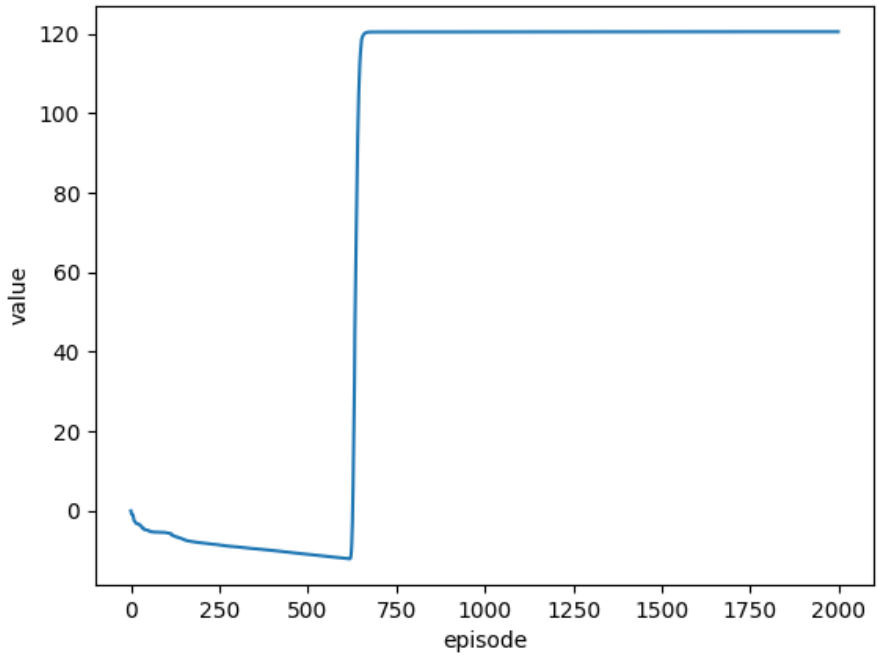

Figure 6: First State Value $\epsilon_i = 1$, $\epsilon_f = 0.1$, $N = 0.4k$, Total $= 2k$

Time taken to train (including logging) = 259.715s

We see that the rewards reach the optimum value quickly but a lot of the states are unexplored. This is because we decrease the epsilon to 0.1 which results in a low exploration for a large part of the training. Also, the optimum value for first state is also learn quickly as the optimal start to end path is explore many times as epsilon goes down in the beginning itself.

In subsequent experiments, we decrease the epsilon to a larger value and decrease it much slowly to increase exploration and develop a more robust policy, but at the cost of more time.

2.1.2 Case 2

$\epsilon_i = 1$, $\epsilon_f = 0.5$, $N = 30k$, Total $= 50k$

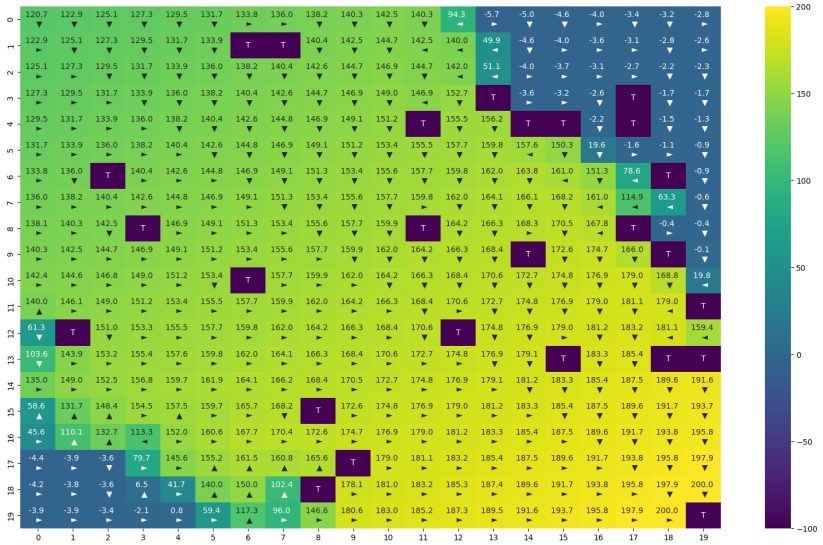

Figure 7: Policy Value Map $\epsilon_i = 1$, $\epsilon_f = 0.5$, $N = 30k$, Total $= 50k$

For this case we see that we learn a much more robust policy. There are still states that haven't been explore sufficiently. This could be because most of the optimum policy at the top left points to the right and then down. And after lots of epochs most of the states learn to point to the goal, so even if we randomly select actions, there's a high chance that the starting states will make the robot go towards the optimum rather than towards unexplored state. Hence, the probability of visiting state that are from optimum (top right annd bottom left) reduces substantially. Same happens in DQNs as well, and as a result in a finit time horsizon, perfect exploration is almost impossible. But the policy should converge to optimum given more time as we can see from the two plots.

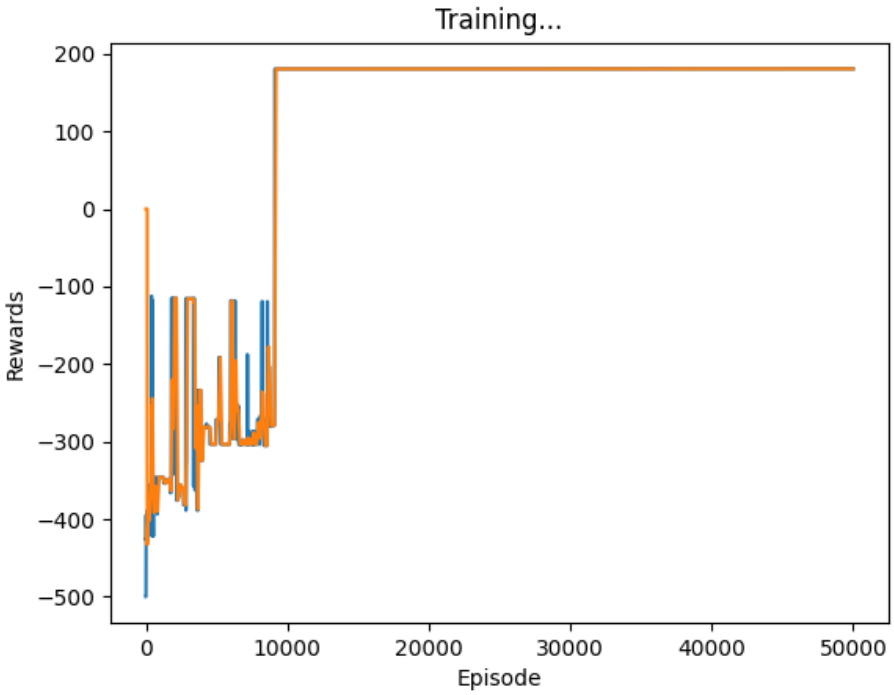

Figure 8: Rewards$\epsilon_i = 1$, $\epsilon_f = 0.5$, $N = 30k$, Total $= 50k$

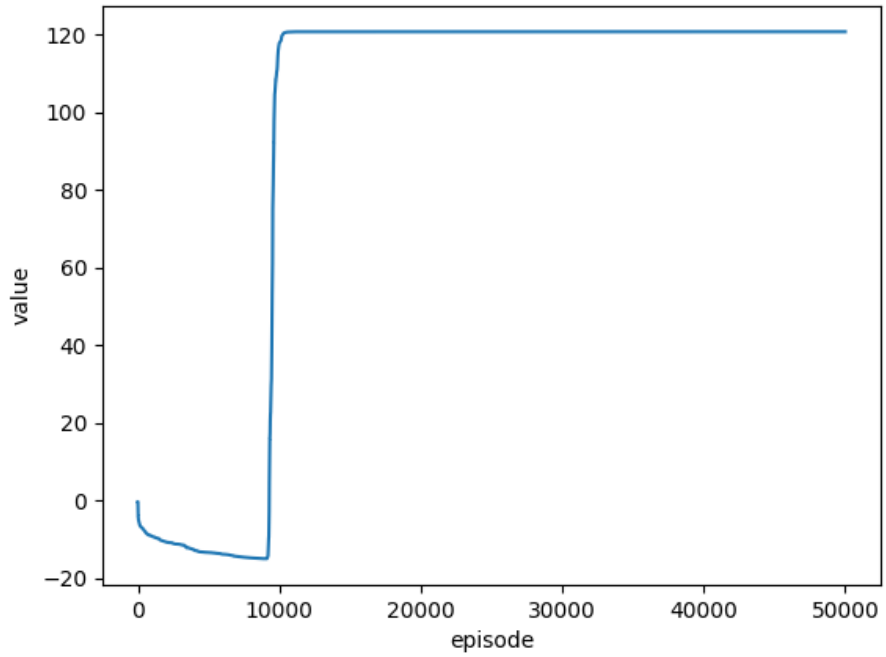

Figure 9: First State Value $\epsilon_i = 1$, $\epsilon_f = 0.5$, $N = 30k$, Total $= 50k$

Over here we see that after around 10k episodes the rewards are constantly very high, even though epsilon is 0.5 . This shows the robustness of the policy learnt. We also see that first state value takes more time to learn here because of slower decay in exploration.

2.2 Large Environment

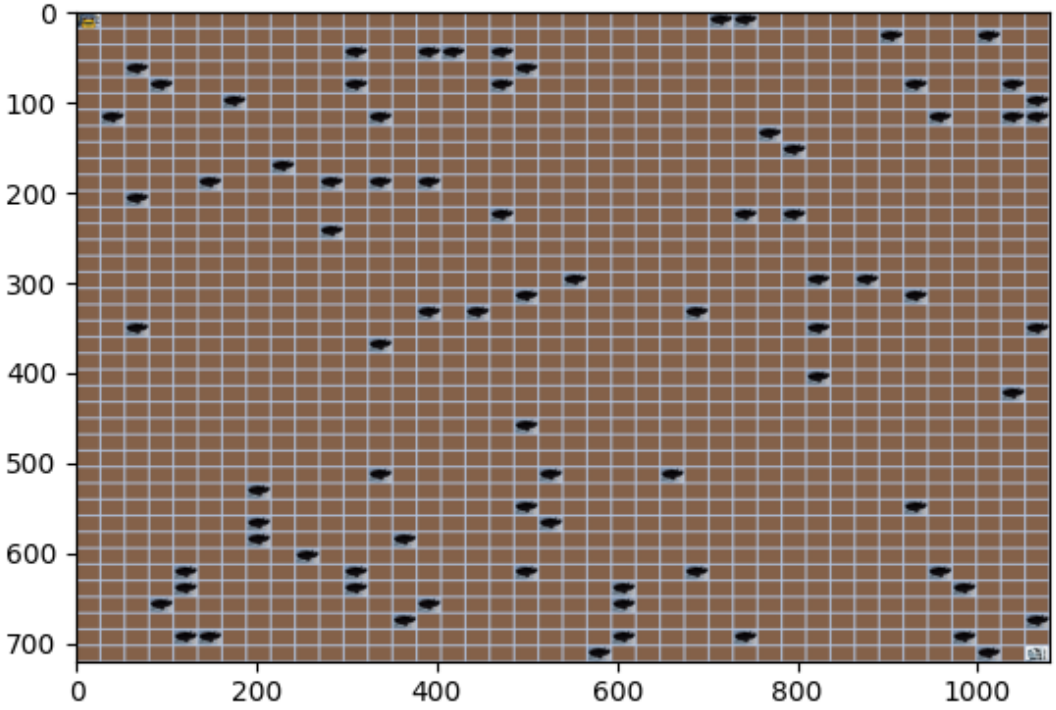

Figure 10: 40 x 40 environment

Because of the increase in environment size, we increase N and total episodes

2.2.1 Case 1

$\epsilon_i = 1$, $\epsilon_f = 0.1$, $N = 1k$, Total $= 4k$

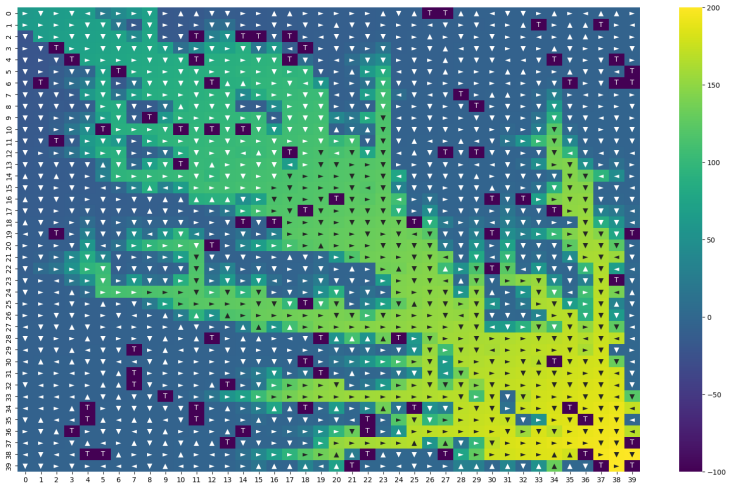

Figure 11: Policy Value Map $\epsilon_i = 1$, $\epsilon_f = 0.1$, $N = 1k$, Total $= 4k$

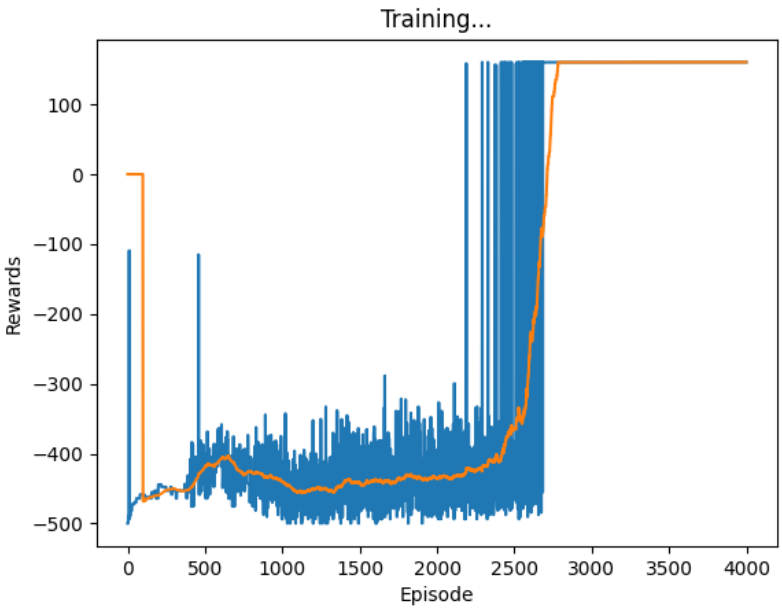

Figure 12: Rewards $\epsilon_i = 1$, $\epsilon_f = 0.1$, $N = 1k$, Total $= 4k$

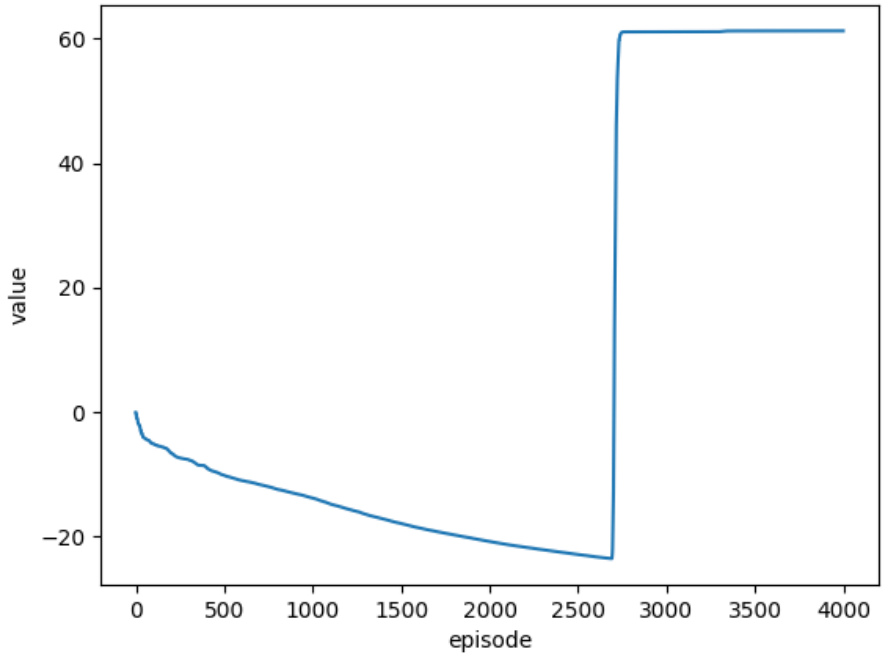

Figure 13: First State Value $\epsilon_i = 1$, $\epsilon_f = 0.1$, $N = 1k$, Total $= 4k$

Time taken to train (including logging) $= 49.611s$

Similar to small environment we are able to learn the start to goal path which but it's not as robust as a lot of states are still not visited.

2.2.2 Case 2

$\epsilon_i = 1$, $\epsilon_f = 0.6$, $N = 30k$, Total $= 50k$

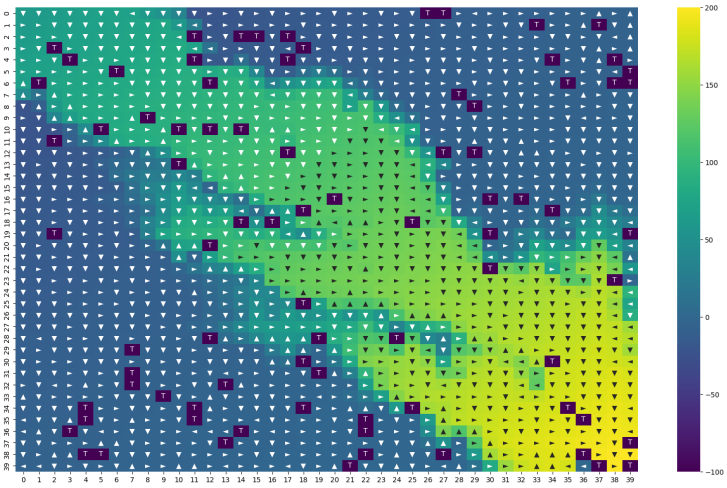

Figure 14: Policy Value Map $\epsilon_i = 1$, $\epsilon_f = 0.6$, $N = 30k$, Total $= 50k$

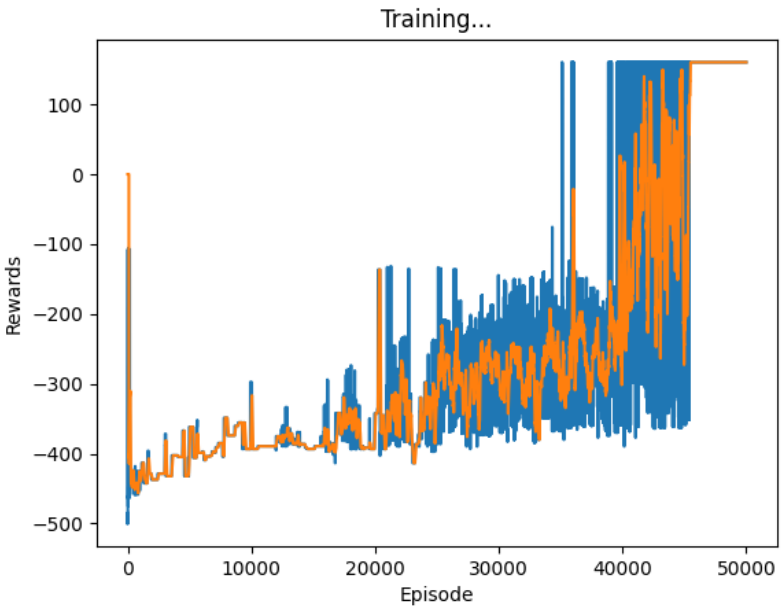

Figure 15: Rewards $\epsilon_i = 1$, $\epsilon_f = 0.1$, $N = 30k$, Total $= 50k$

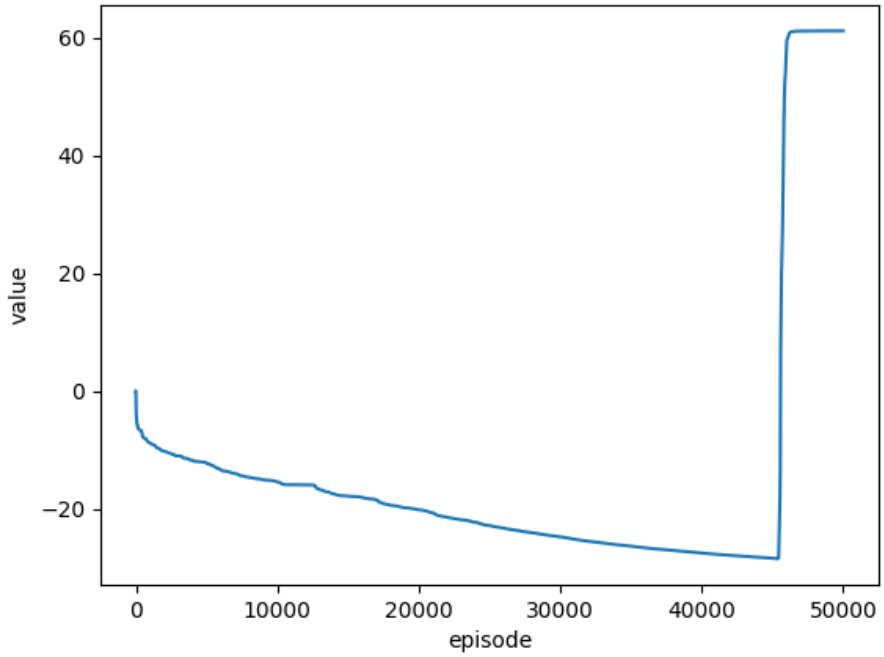

Figure 16: First State Value $\epsilon_i = 1$, $\epsilon_f = 0.1$, $N = 30k$, Total $= 50; k$

Time taken to train (including logging) $= 398.039s$

Similar to Small Environment we learn a more robust policy. But the policy is not as robust as we haven't run it for further iterations but it can be seen that further iterations the policy will significantly improve.

2.3 Comparison with Value Iteration

In Value Iteration we are given a model of the world and we just have to find the Q-Values. If we were given the model, we would have been able to learn the Q-states of all states better. Over here we aren't able to learn the Q-values of all state properly as we are limited by exploration and exploitation. So we are bound to miss out on a significant number of states and not learn their Q-value properly, but an interesting property that we see the the Q-value action plots is that the states that haven't been visited multiple times also have good policies. If we had the model, we could have run multiple iteration of Value-iteration to learn the optimum Q-values and the optimum policy but the policy here is also good. The learned policy is not exactly optimal but it would tend to optimality on running it for more iterations.

3 Deep Q-Networks

Here, we used an **MLP** with the following architecture:

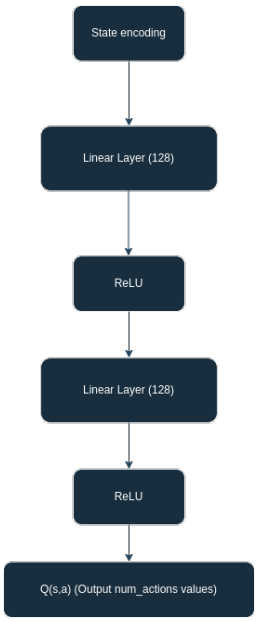

Figure 17: Architecture of network

We used the SmoothL1loss for our experiments. We used a replay buffer size of 10000 and trained the models with a batch size of 128. We use a ϵ-decay in all our training.

$\epsilon = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) * e^{\frac{-step}{decay}}$

3.1 Input

We chose to utilize a one-hot vector format instead of a singular integer format to depict the state as input for the neural network. This choice expands the dimensionality of the input space, facilitating the network's ability to discern intricate relationships and patterns within the input data more efficiently. Moreover, employing the one-hot encoding method eradicates any ordinal associations among states inherent in a solitary integer representation, enabling the network to treat states as autonomous entities without attempting to derive a mean from neighboring states.

3.2 DQN with Replay Buffer

Here, we use the following variation of DQN:

$Q_{target}(s, a) = r + \gamma maxQ(s', a', \theta_{target})$

3.3 Small Env

We trained this for 2K episodes with decay learning rate parameters $\epsilon_{start} = 1$ and $\epsilon_{end} = 0.1$ with decay = 3000. Time: $9min.34sec$

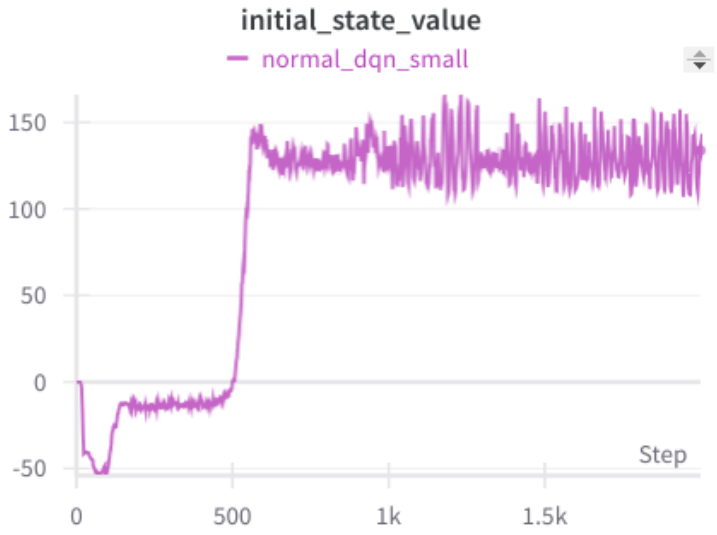

Figure 18: Start state value

Figure 19: Train reward

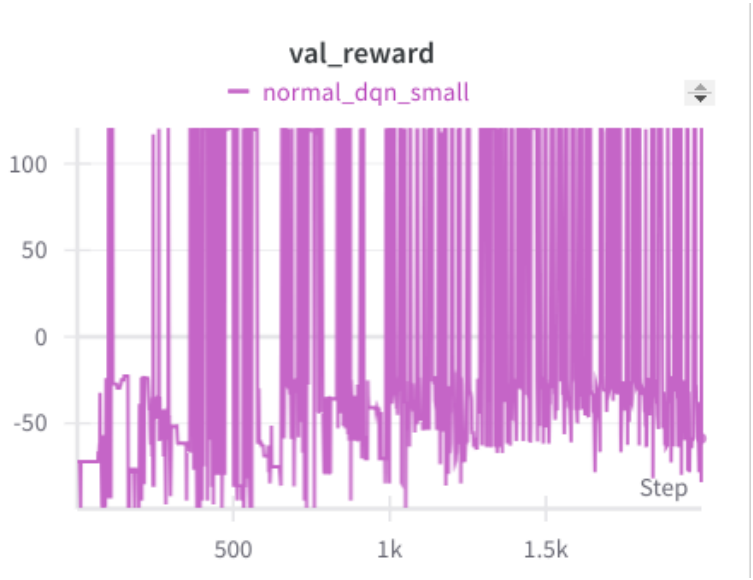

Figure 20: Val reward

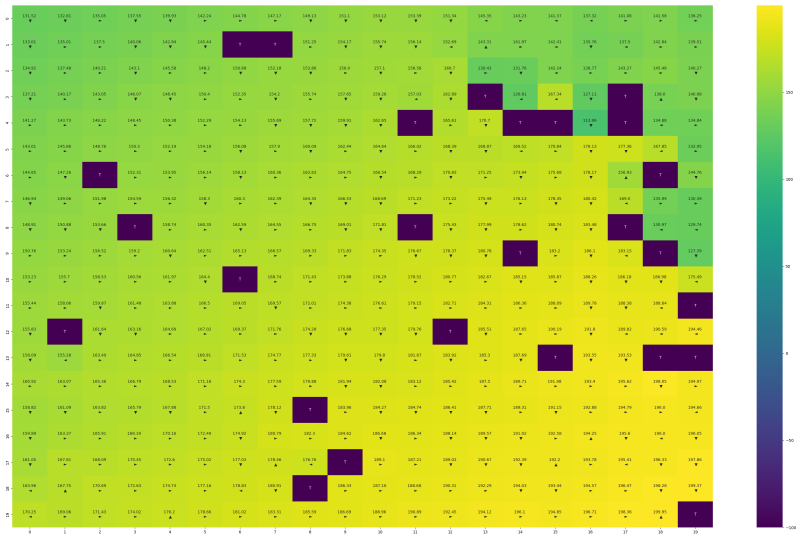

Figure 21: Qvalue with action

3.4 Large Env

We trained this for 5K episodes with decay learning rate parameters $\epsilon_{start} = 1$ and $\epsilon_{end} = 0.1$ with decay = 3000. Time: $1hr6min$

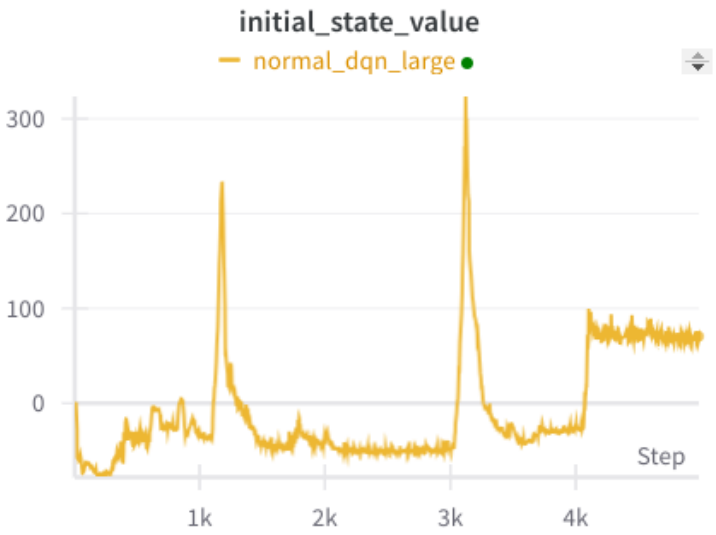

Figure 22: Start state value

Figure 23: Train reward

Figure 24: Val reward

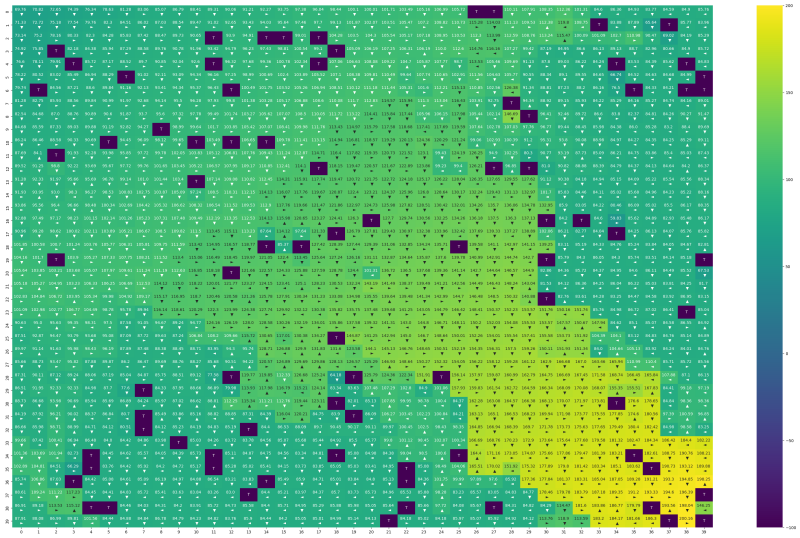

Figure 25: Qvalue with action

3.5 Double DQN

Here, we use the following variation of DQN:

$$Q_{target}(s,a) = r + \gamma Q(s', a' = argmaxQ(s', a', \theta_{main}), \theta_{target})$$

3.6 Small env

We trained this for 2k episodes with $\epsilon_{start} = 1$ and $\epsilon_{end} = 0.1$ with decay = 3000. Time: $9min$.

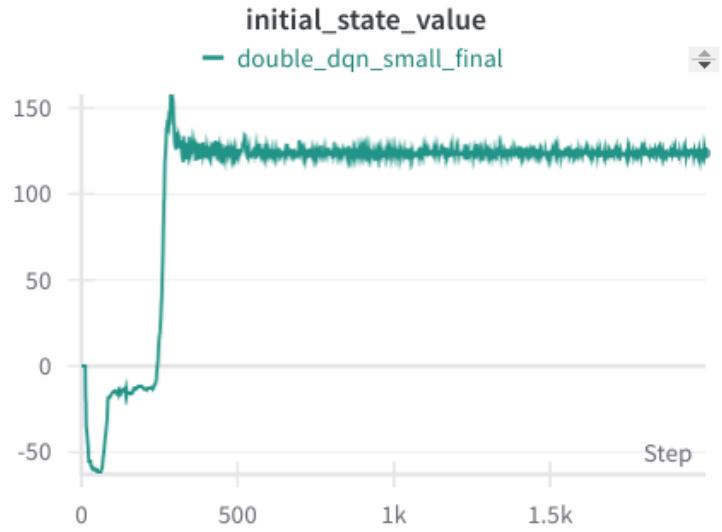

Figure 26: Start state value

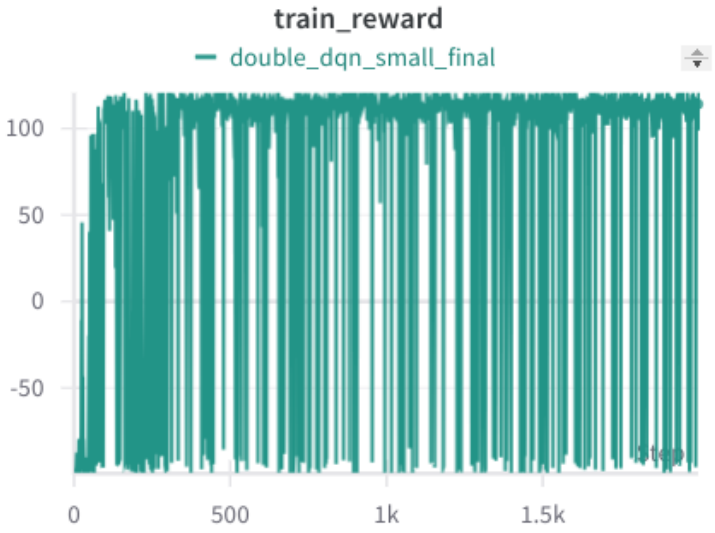

Figure 27: Train reward

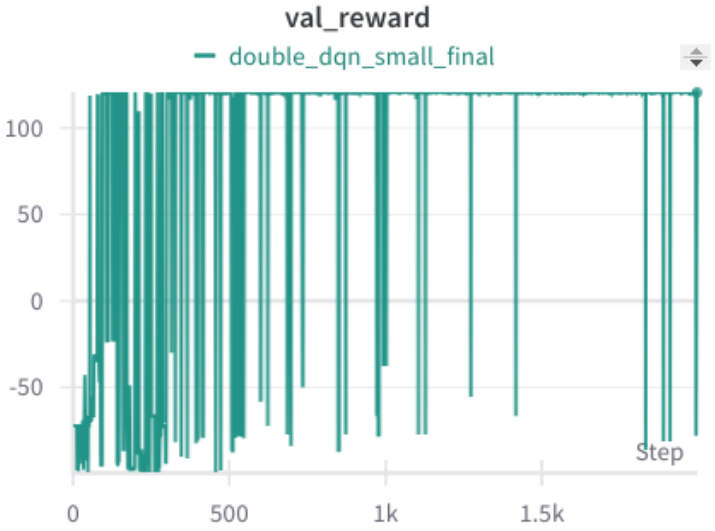

Figure 28: Val reward

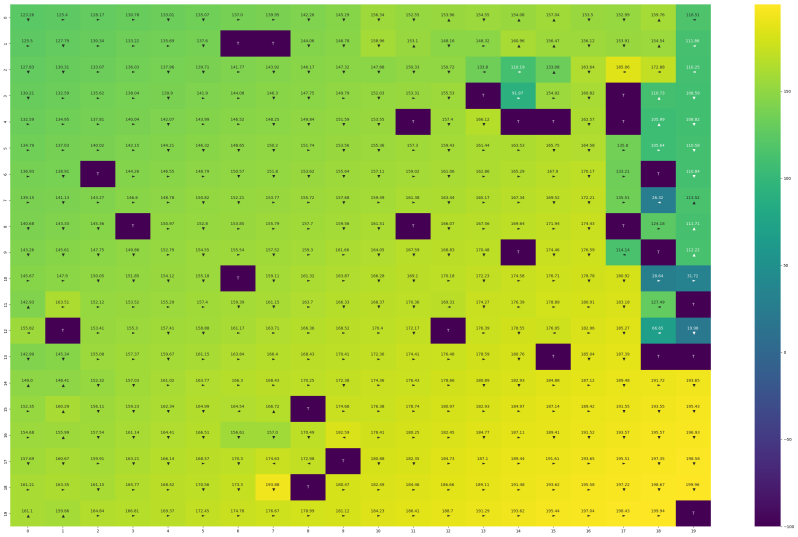

Figure 29: Q-value with actions

Training time: 2k steps, $9mins34s$

3.7 Large env

We trained the model for 15K epsiodes, with $\epsilon_{start} = 1$ and $\epsilon_{end} = 0.1$ with decay $= 3000$. Time: $2hrs$

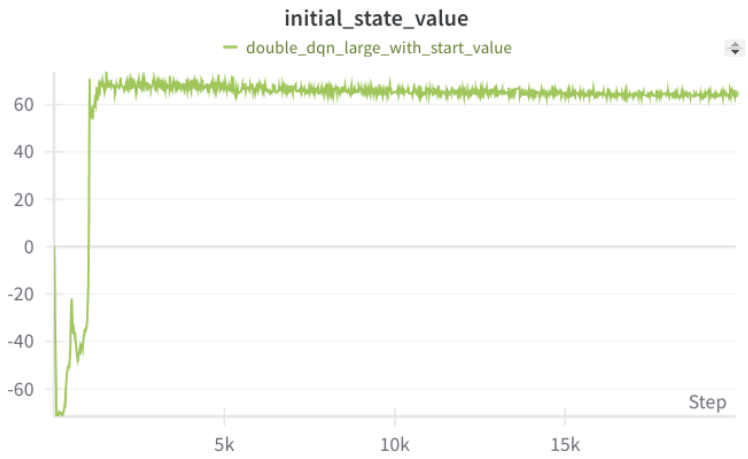

Figure 30: Start state value

Figure 31: Train reward

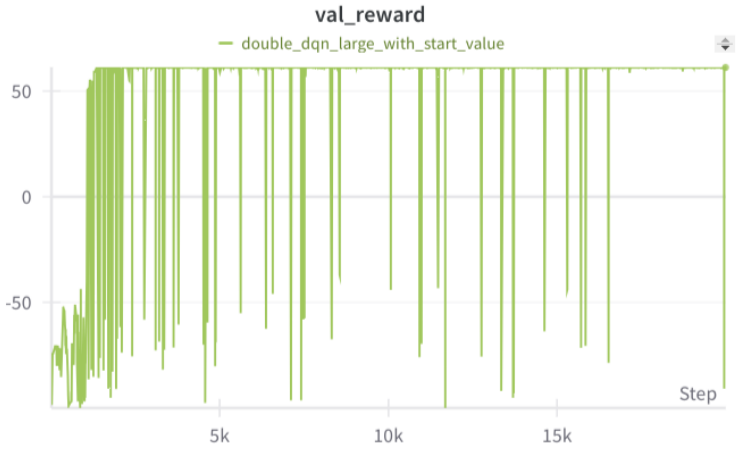

Figure 32: Val reward

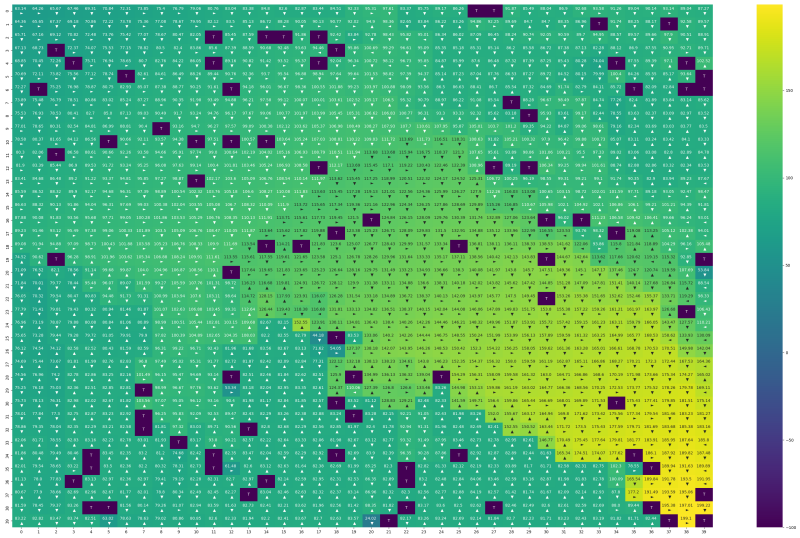

Figure 33: Q-values plot with action

3.8 Double DQN vs Single DQN

We observed that the Double DQN performs significantly better than Single DQN, and offers a much more stable training (as can be seen from first state values). In the Single Q Network, during the estimation of Q-values, there is a tendency to overestimate the true values, particularly for certain states or actions. This overestimation can lead to sub-optimal policies being learned.

However, in the Double Q Network, the overestimation is decreased by using two separate networks to evaluate actions. One network is responsible for selecting actions, while the other is used to evaluate the value of those actions. By decoupling the action selection and value estimation

processes, the Double Q Network is better able to provide more accurate Q-value estimates and thus learn more optimal policies.

In essence, the Double Q Network reduces the bias introduced by overestimation in the Single Q Network, leading to improved performance in learning tasks.

3.9 Double DQN with priority experience replay

We also performed our experiment with priority experience replay motivated by this paper. We used the following hyperparameters: $\alpha = 0.4$ and $\beta = 0$ for our experiment. We experimented only with Large environment.

We trained here for 10K episodes. Time: $1hr20min8s$. We also noted that this was able to match the policy obtained by general Q-learning in lesser number of episodes.

Figure 34: Train reward

Figure 35: Val reward

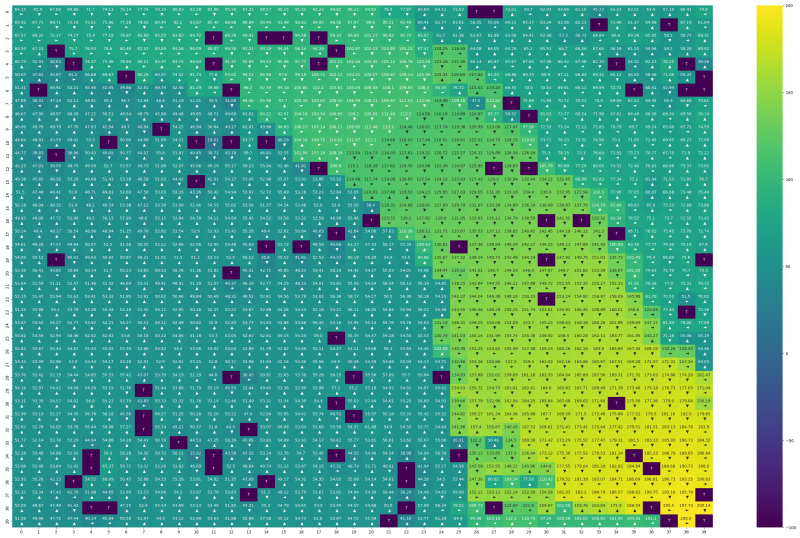

Figure 36: Q-values

3.10 Comparison of DQN and Tabular

- **Time**: Broadly speaking, we notice that Deep Q-Learning exhibits a greater demand for data since it needs to estimate function parameters and fine-tune weights through loss optimization. Consequently, it needs more time to reach convergence.

- **Policy**: The policy produced by both is approximately similar, but neural network is able to

generalize better than tabular, as it learns a function.

- **Comparison with Model-Based Learning Algorithms**: Q-learning is categorized as a Model-free approach. This characteristic renders it more suitable for environments where environment dynamics are either unknown or challenging to accurately model. The policies acquired through Q-learning tend to be nearly optimal due to extensive training. However, intuitively, if policy iteration or value iteration possess precise prior knowledge of transition and reward models, they may not require as much exploration, potentially leading to superior policies within a given time frame. Conversely, in scenarios with expansive state spaces and intricate environment dynamics that are challenging to model accurately, Q-learning might converge more swiftly. This is because it updates Q-values based on sampled experiences rather than iterating over all states and actions, as in policy iteration.

4 Work Distribution

All work was distributed equally.