COL778

ASSIGNMENT 3

Q-Learning

Parth Shah *(2020CS10380)*
Shubh Goel *(2020EE10672)*

Date of submission of Report: April 7, 2024

Team Members

Shah Parth Urveshkumar - 2020CS10380

Shubh Goel - 2020EE10672

Part I: Tabular Q-Learning

Exploration strategy and experiments with α

- After experimenting with different values of $\alpha(0.9, 0.64, 0.32, 0.15, 0.08)$, **0.15** worked the best for us.
- Discounting factor, $\gamma = 0.99$
- **Exploration strategy:**
  - The algorithm is run for 75000 episodes
  - For the first 25000 episodes we randomly select an action in a state to promote exploration
  - we then employ ϵ-greedy method to select action.
  - We start with $\epsilon = 1$ and decrease it exponentially using a factor of 0.999 till 0.5

Plots

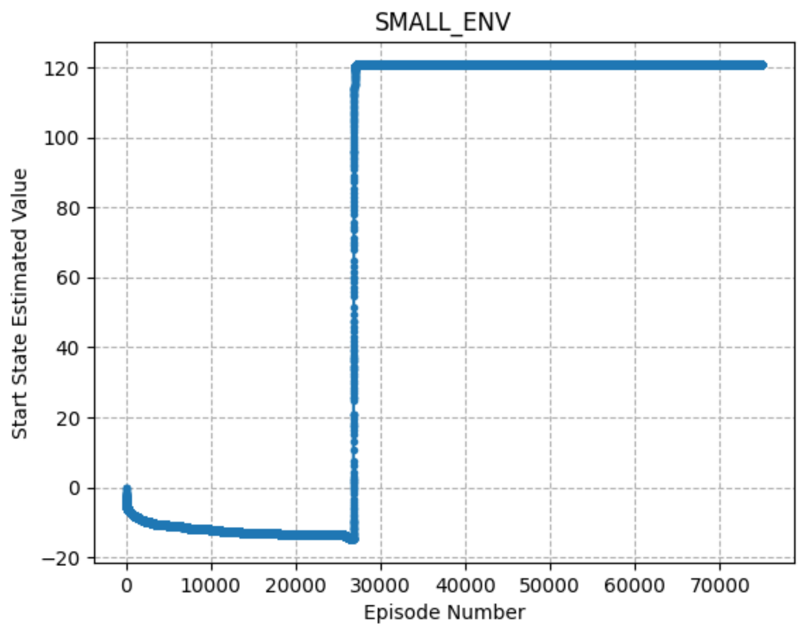

Figure 1: Starting state estimated value for small environment

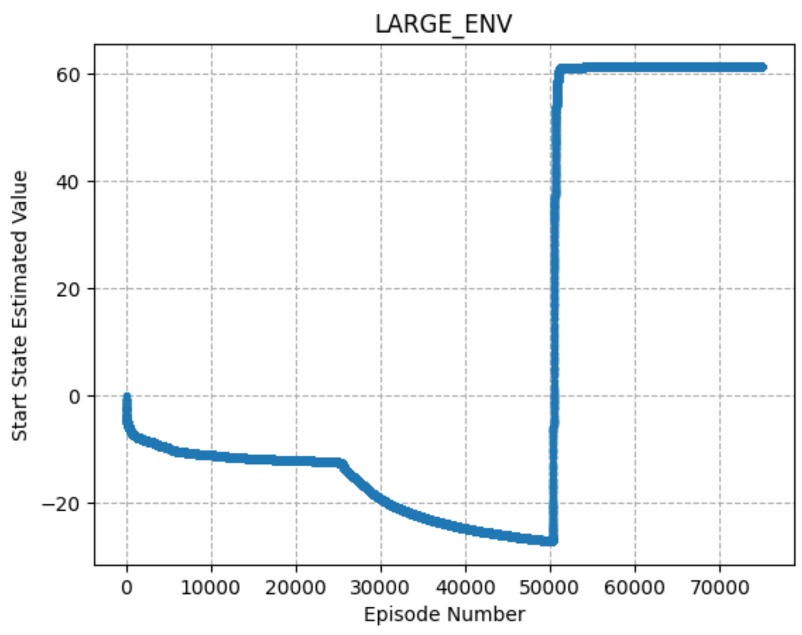

Figure 2: Starting state estimated value for large environment

- For the initial episodes the agent is in high exploration setting. As a result, the agent might explore different actions and transitions, which could lead to negative rewards or sub-optimal decisions. Hence, we see a decrease in the value associated with the initial state.

- As the agent moves towards exploitation, it starts to learn which actions lead to higher rewards. Through the Q-value updates, the agent gradually improves its estimates of the true values of state-action pairs, including the starting state.

- Eventually, with continued exploration and learning, the Q-values converge towards their optimal values. At this stage, the Q-values for the starting state stabilize because the agent has learned the best actions to take from that state to maximize cumulative rewards. As a result, the Q-values become constant, indicating that the agent has learned the optimal policy.

Heat map for small environment

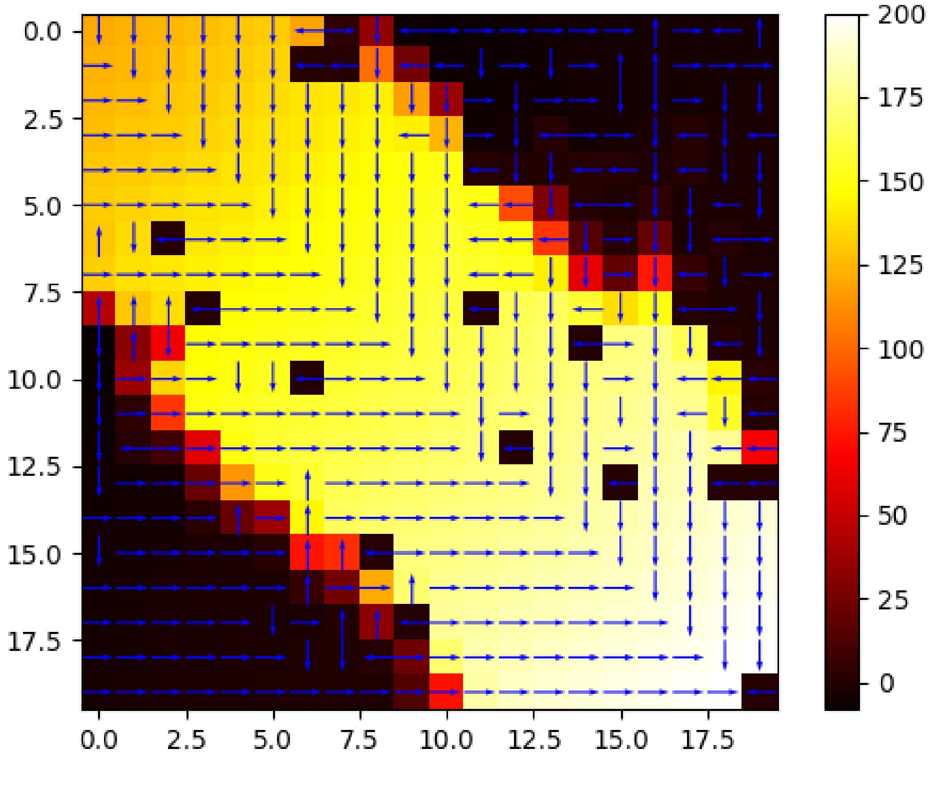

Figure 3: Small Environment

Heat map for large environment

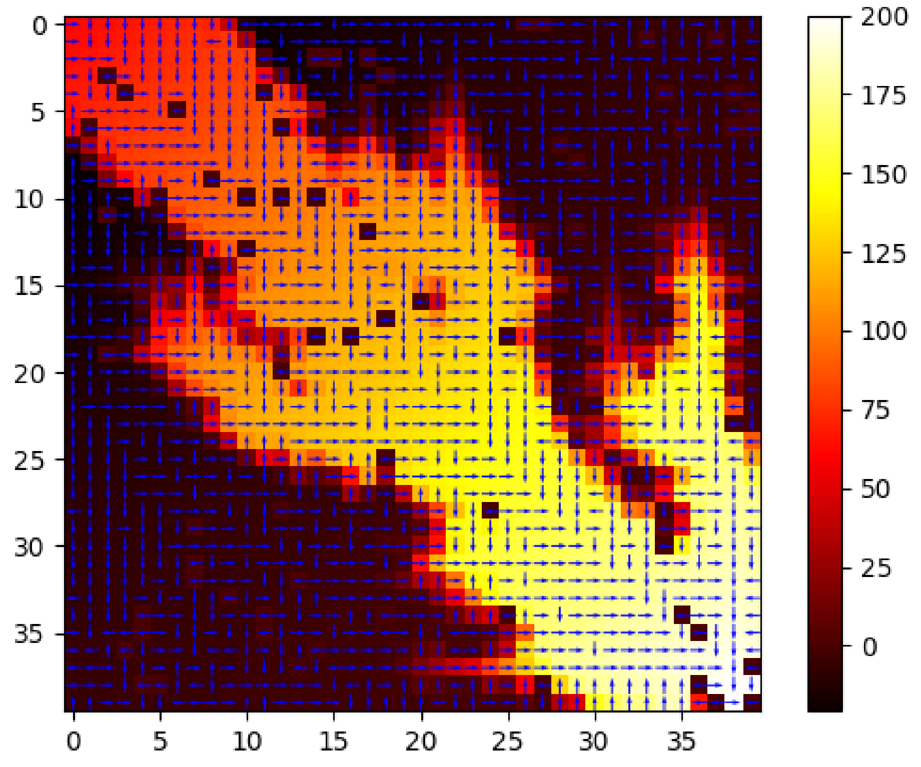

Figure 4: Large Environment

Optimality of learned policy

- As we can observe from Fig. 10, the learned policy is not goal directed from every state. Hence, it is not optimal. Although, the converged policy is such that if the agent begins from the start state, it would get to the goal state following an optimal path
- The bottom left and top right states remained unexplored. This deviation from optimal behavior is not observed in algorithms like 'Value Iteration' or 'Policy Iteration,' where every state's value is updated in each iteration, facilitating convergence to optimal policies.
- However, with Q-learning, given a sufficient number of episodes, these unexplored states would eventually be encountered and their values updated. Consequently, the agent's exploration over time could lead to improved policy convergence.

Part II: Deep Q-Learning (Neural Network as Q-Function)

Neural Network Architecture

The neural network we used for estimating the Q-values has the following specifications

- Input : Concatenation of 2 one-hot vectors one for the column and one for the row corresponding to the state

- Layers: Input x 64, 64 x 64, 64 x output
- Output : Dimensions = Number of actions, q-value estimated for each possible action for the corresponding state

Training Strategy

- We maintain 2 models, online_model(for training) and target_model(for calculating value of y_true)
- We use the experience replay method for training the neural network for predicting q-values
- For every episodes we perform the following steps:
  - Reset the environment to initial conditions
  - For every time-step till the episode terminates
  - Use ϵ-greedy method to find an action
  - Find next_state corresponding to the action previously selected
  - Add the (state, action, reward, next_state, terminated) tuple to the replay buffer
  - Choose a random batch from the experience buffer and train the model
- We copy the parameters of online_model to target_model after certain time-steps are completed
- We use exponential decay on ϵ starting from $\epsilon = 1$ till $\epsilon = 0.5$
- The random batch selected consists of 63 randomly sampled entries from the replay buffer and the recent most entry of the replay buffer
- Value of hyper-parameters:
  - $\epsilon = 1.0 \longrightarrow 0.5$
  - $\epsilon\_decay = 0.999$ after 2500 episodes
  - $\gamma = 0.9$ or 0.99
  - $training\_steps = 4$ or 1
  - $batch\_size = 64$
  - $num\_episodes = 40k$

Plots

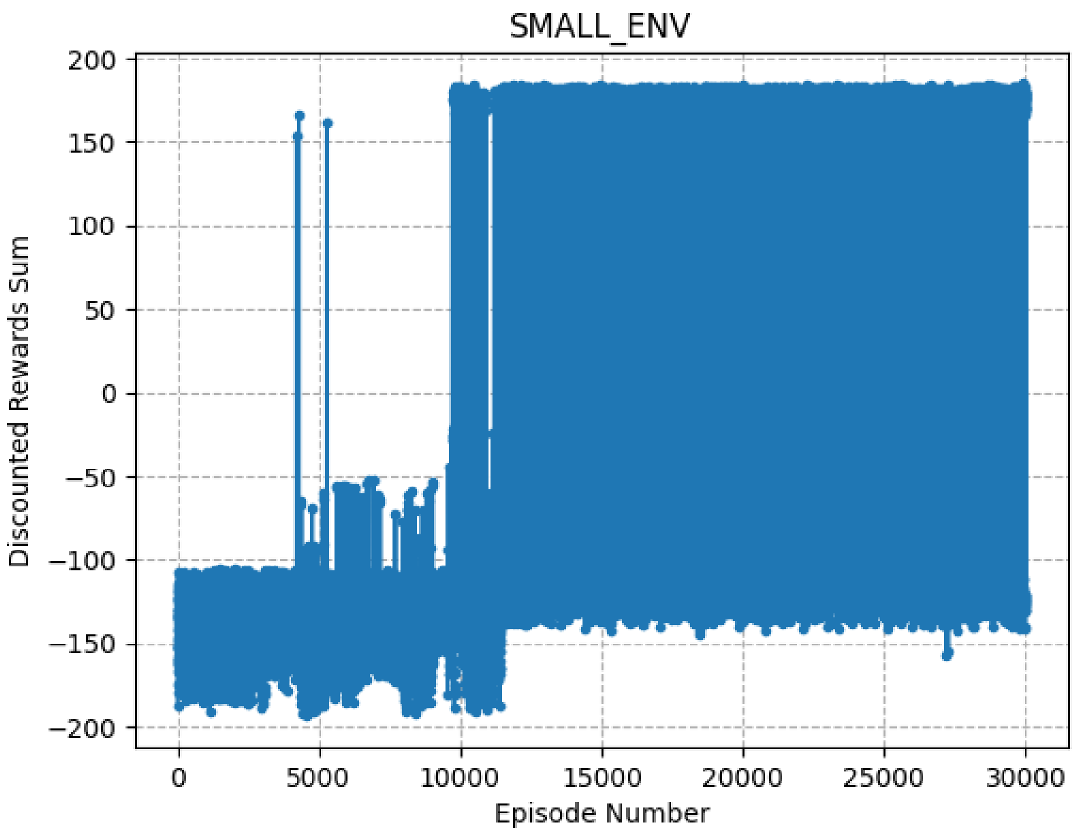

Figure 5: Discounted rewards vs. Episodes (small environment)

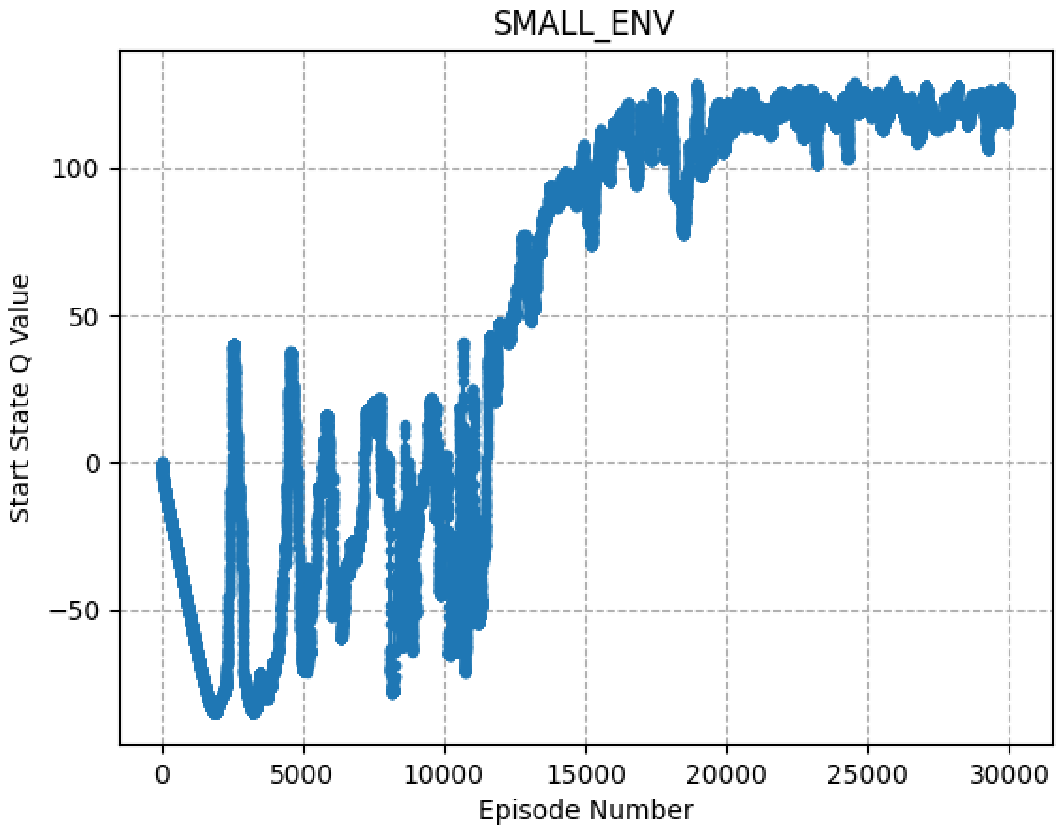

Figure 6: Initial state value vs. Episodes (small environment)

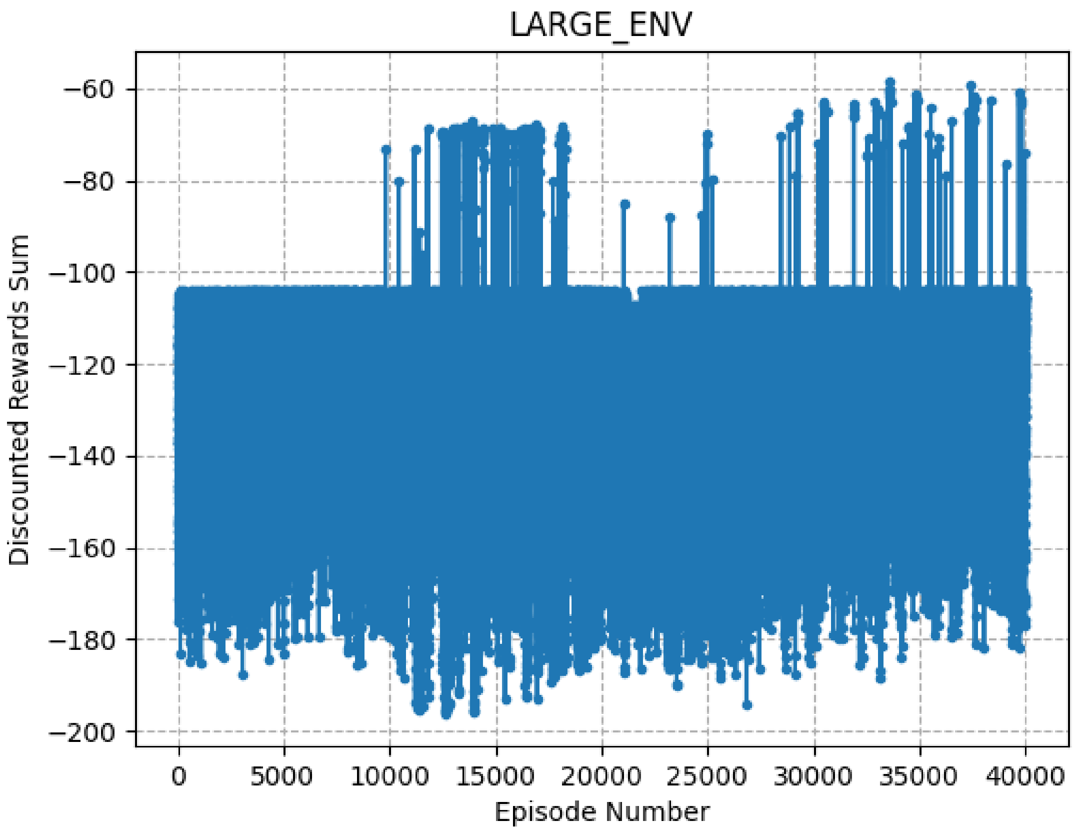

Figure 7: Discounted rewards vs. Episodes (large environment)

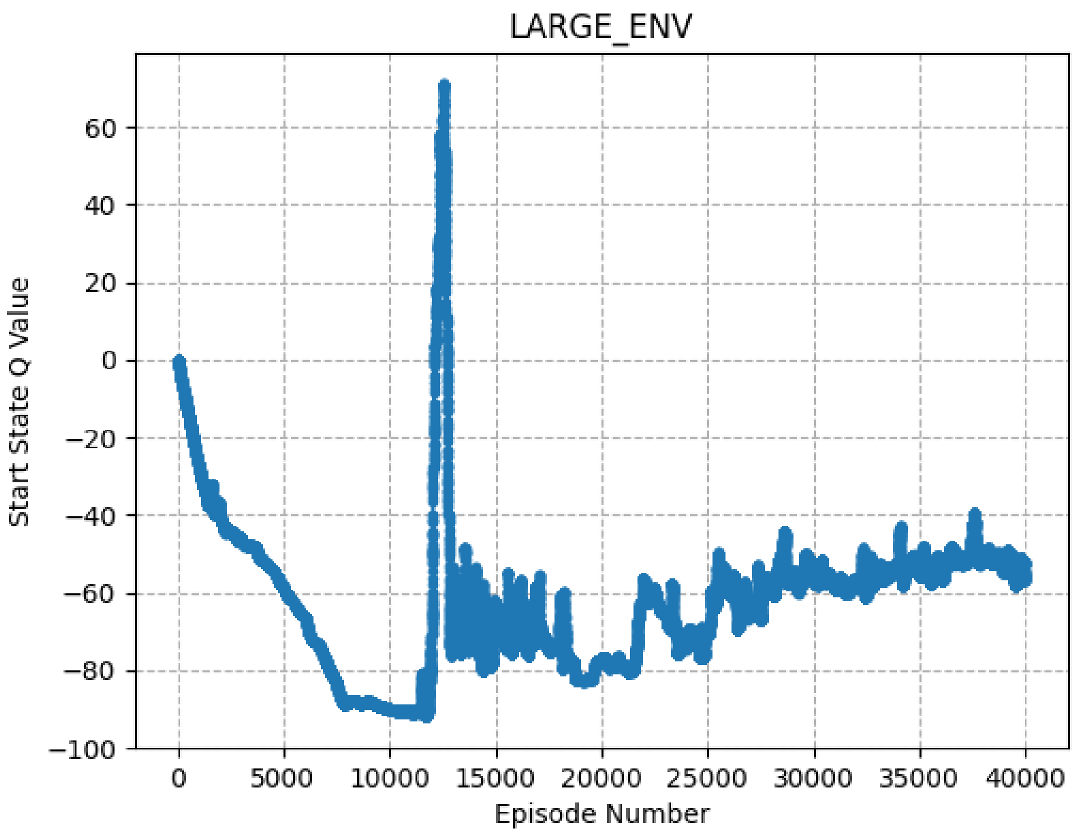

Figure 8: Initial state value vs. Episodes (large environment)

- During initial learning phase because of high value of ϵ the agent explores multiple paths and states due to which the Q-value is not very stable. We can notice such behaviour in the graphs
- As the value of ϵ decreases the agent reduces the deviation from the path towards the goal and the values of rewards become stable.

Heat Maps

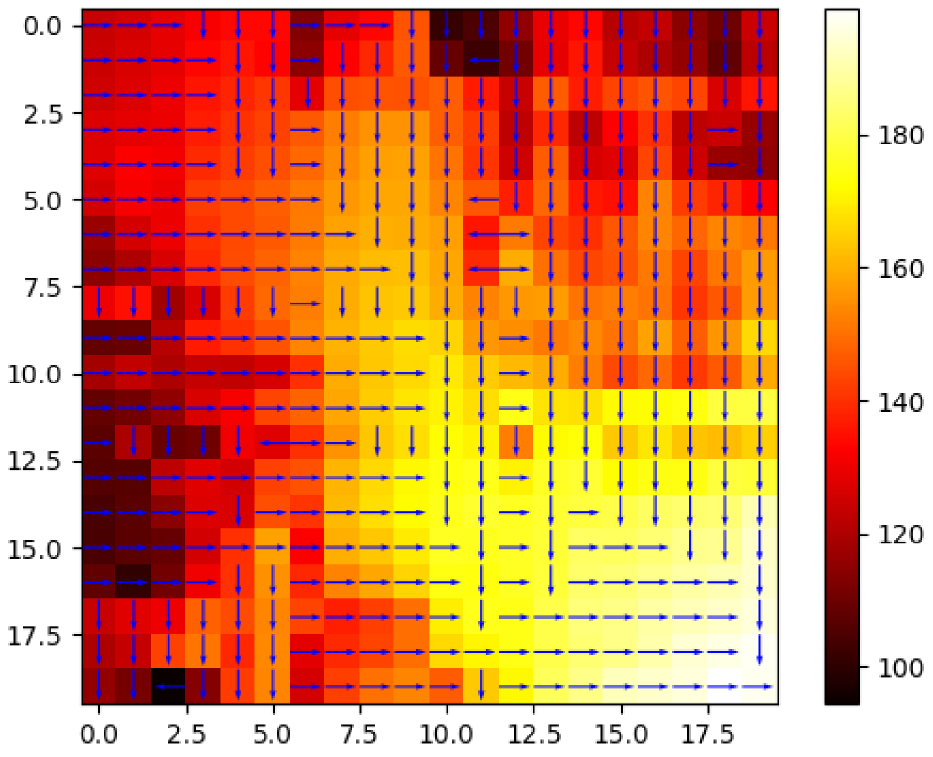

Figure 9: Small Environment gamma=0.99

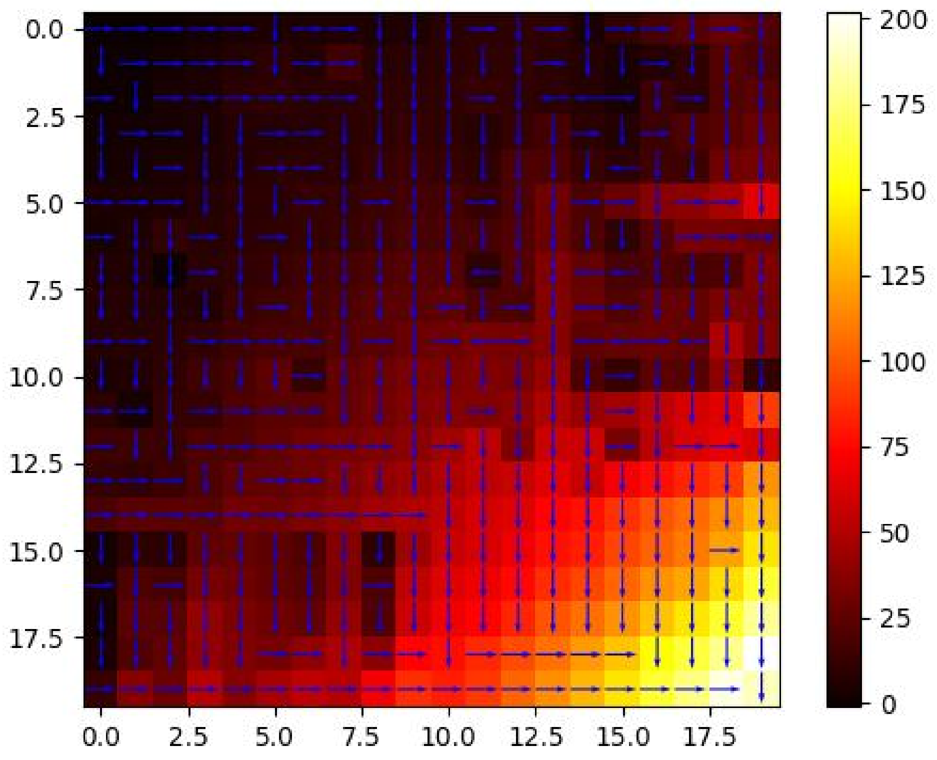

Figure 10: Small Environment gamma=0.9

Comparison with tabular method

Considering the small environment for the comparison:

	Tabular Method	**DQN**
Training time	10 mins	2.5 hrs
Num episodes	75k	30k

Table 1: Comparison of Training Time and Number of Episodes

- As there is no training of neural network required and there is less exploration in the tabular Q-learning DQN takes a much longer time to train then tabular Q-learning
- We can also observe that DQN takes less number of episodes for finding an appropriate approximation of Q-values than tabular method.
- When we consider the final policy obtained by the agents both the method helps to learn an optimal policy for the path from the initial state to the goal.
- Whereas the policy learned by the agent using DQN method for the states other than the path taken to reach the goal is much better than the policy learned in case of tabular method. This is because of two reasons:
  - We use ϵ greedy approach in DQN which helps us in exploring most of the states a sufficient number of times and thus the agent can learn their Q-values

  - Because we use neural network for estimation, the states similar to the ones already explored also keeps on learning similar Q-values.

Contributions

Both of the team members (Shah Parth Urveshkumar and Shubh Goel) contributed equally for the assignment