Q-Quest : RL Approaches in CLIF-Environment

Pratyush Bindal – 2022A7PS0119H,
Uday Sudani – 2022A7PS0136H

***Abstract*—We investigate off-policy Q-Learning-based Reinforcement Learning (RL) approaches for agent navigation in a CLIF-style environment. We emphasise the methodologies utilised and evaluated performance of the Q, Double-Q, Triple-Q, and Quadruple-Q Learning algorithms. Our basis of this report is based on the ideas of popular games such as Super Mario, Flappy Bird and Hill Climb Racing, and we tried to mimic and modify existing game formulations, and implemented a custom game-style environment. We utilised variations of Q-learning algorithms so that the agent moves through a dynamic environment filled with static and dynamic dangers, platforms, adversaries, coins, and boosts, while experiencing other stochastic aspects. The agent chooses the optimal action to reach the goal by collecting maximum number of coins and also avoiding to incur penalty by falling into hazards or other pitfalls. Traditional Q-Learning updates a single Q-table, which can lead to overestimation bias, but Double Q-Learning addresses this issue by spreading the Q-value updates across two Q-tables. Triple and quadruple Q-Learning enhance this by including more Q-tables, which reduces overestimation while increasing computing cost. Various assessment measures, such as cumulative rewards, average rewards, and training time were used to examine the effectiveness of each technique in complicated environments.**

Keywords—CLIF-style Environment, Q-Learning algorithms

I. Problem Formulation: Q-Quest

We focused on developing a reinforcement learning (RL) agent capable of operating within a dynamic and complex CLIF-based environment. The agent's objective is to reach the goal while collecting maximum number of coins while avoiding any hazards, dangers or pitfalls it faces while moving on the platform from starting point to ending point.

A. Environment Setup

The environment is represented as 2D grid with each cell representing a state where the agent can interact. There is a single agent named *Pixel Piero*, whose state is represented as a tuple of x-coordinate, y-coordinate, velocity. We implemented three versions of the game – the primitive version called *Beginner Quest*, an upgraded version called *Superior Quest*, and an advanced version called *Elite Quest*.

B. Beginner Quest Environment

Pixel Piero can take *three* actions in the Beginner Quest Environment which are moving towards left, moving towards right, and jumping up if he is on the platform. Different elements populate the grid, each introducing unique interactions and consequences for him:

1) Navi Spaces: Basic navigable spaces with no specific reward or penalty.

2) Skywalks: Pixel Piero can only move on various platforms called Skywalks of varying lengths scattered across the environment at different heights.

3) Genesis and Horizon: Pixel Piero starts from the Genesis and the aim is to move till the Horizon, that is the end position, while maximizing rewards accumulation. Reaching Horizon gives high rewards (Reward = +50)

4) Pixie Tokens: These tokens are scattered throughout the environment which Pixel Piero should collect for getting positive reward points (Reward = +5)

5) Turbo Tokens: These are the boosts which give higher reward than accumulating coins. (Reward = +10)

6) Meany Beasts: These are enemies placed randomly on the platform which walk on skywalks in predefined pattern which Pixel Piero does not know but should learn to minimise encounter. There is a penalty for encountering an enemy (Penalty = -15)

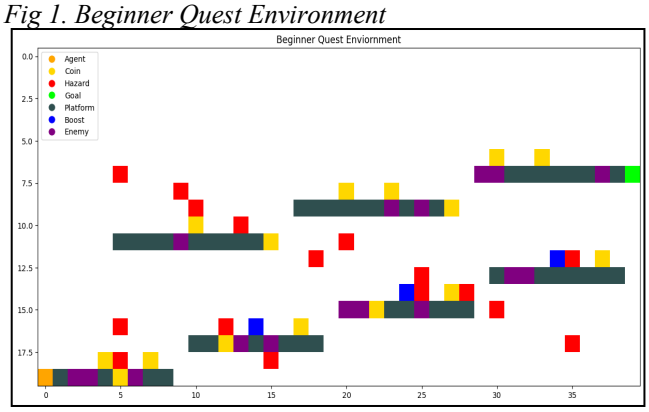
7) Fixie Frights: These are static hazards which are kept at fixed positions and do not move. Encountering them yields a penalty for Pixel Piero. (Penalty = -10)

8) Tricky Traps: These are dynamic hazards which are initially placed at fixed positions and are the moved after each episode in a pattern. Encountering them yields a penalty for Pixel Piero. (Penalty = -10)

9) Drop Zone: Pixel Piero will face penalty if it falls out of skywalk towards the ground. (Penalty = -20)

10) Gravity and Jump Strength: These are additional elements which will alter Pixel Piero's vertical velocity. (Jump Strength = 2 units and Gravity = 1 units)

Fig 1. Beginner Quest Environment



C. Superior Quest Environment

It is an upgraded version to the Beginner Quest Environment, where Pixel Piero can take *four* actions instead of three which are moving towards left, moving towards right, jumping up if he is on the platform and an additional action to kill Meany Beasts. The modifications of different elements populating the grid with respect to Beginner Quest Environment, each introducing unique interactions and consequences for Pixel Piero has been listed below:

1) Horizon: Reaching Horizon gives very high reward (Reward = +500)

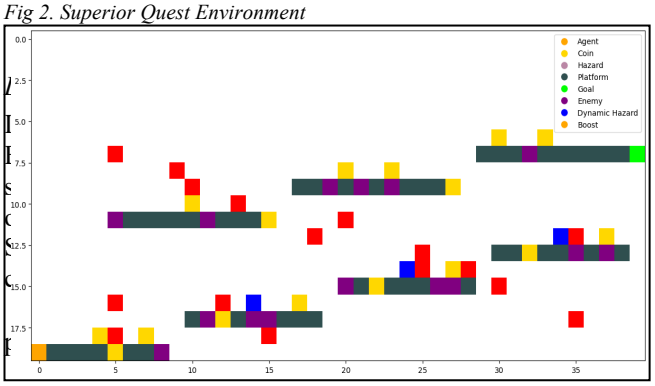
2) Danger Zone: Falling out of skywalk attracts higher penalty (Penalty = -30)

3) Meany Beasts: Pixel Piero can now kill enemies and penalty for killing now incorporates the count of Meany Beasts killed multiplied by the original penalty.

4) Survival Quest: There is an additional penalty for each step taken to reach the goal which will encourage Pixel Piero to reach the goal in minimum count of steps. (Penalty = -1)

5) Gravity and Jump Strength: Jump Strength has been enhanced to 4 units whereas the logic for incorporating gravity has been modified. If Pixel Piero is not on a platform and can fall, it enables gravity, hence, increasing its vertical velocity and checking how far it can fall before hitting a platform. If Pixel Piero reaches a platform, it stops falling instead of dropping to the ground. If Pixel Piero lands or is on a platform, his downward speed is reset to zero.

Fig 2. Superior Quest Environment



E. Elite Quest Environment

Modifications influencing Pixel Piero have been listed below:

1) Danger Zone: Fall Penalty has been increased. (Penalty = -300)

1) Pixie Tokens: Collecting Pixie Tokens yields higher rewards (Reward = +60)

2) Jump Strength: Jump Strength has been set to 3 as in Beginner Quest Environment

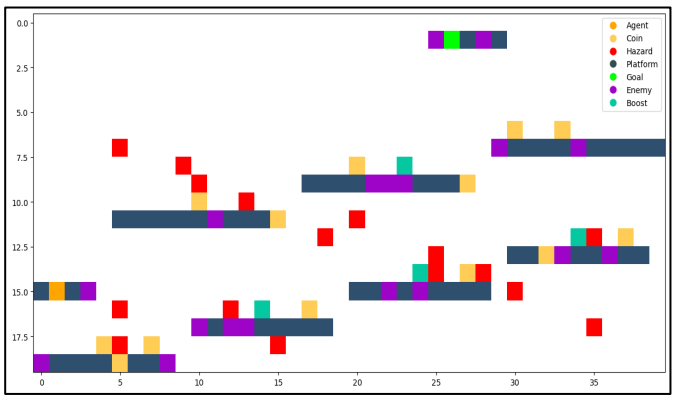
3) Turbo Tokens: Jump Strength is influenced by the amount of turbo tokens accumulated so far.

4) Meany Beasts: Pixel Piero can now kill enemies and penalty for killing now incorporates the negative of the count of Meany Beasts killed multiplied by original penalty.

5) Environment: Additional platforms, tokens and traps have been set up for rendering more complex environment.

Remaining elements incorporated in Q-Quest remains same for all the environments.

Fig 3. Elite Quest Environment



F. Pixel Piero's Objectives and Constraints

The RL agent must navigate through the environment to achieve the following:

1) Maximise Cumulative Rewards: Pixel Piero should maximise cumulative reward accumulation by collecting pixie and turbo tokens and reaching the Horizon.

2) Avoiding Penalties: Pixel Piero should avoid encounters with Meany Beasts, Fixie Frights and Tricky Traps by learning their movement pattern or remembering their fixed positions. Furthermore, Pixel Piero should avoid going into Drop Zones, as there is no recovery point from that zone, hence, the current episode would terminate. In Superior Quest Environment, Pixel Piero should also take care of count of steps to minimise the penalty for each step taken.

II. Implemented Reinforcement Learning Algorithms

We implemented the variations of the Q-Learning algorithm to analyse comparatively the difference and similarities between the variations while also selecting the best algorithm for the two environments listed above. The parameters used for the environments have been listed in Table 1.

Table 1. Hyperparameters for Game Environments

Hyperparameters	**Environment**		
	Beginner Quest Environment	***Superior Quest Environment***	***Elite Quest Environment***
Learning Rate	0.02	0.08	0.08
Dicsount Factor	0.95	0.90	0.90
Exploration Rate	1.00	1.00	1.00
Minimum Exoploration Rate	0.001	0.01	0.01
Exploration Decay	0.9994	0.9999	0.9993

A. Q-Learning Algorithm

We set up and initialise Q-table to zero for the possible actions, three or four depending on the environment. During training, it chooses actions based on an exploration strategy that balances random exploration and exploiting known rewards. For each episode, Pixel Piero receives feedback in the form of rewards, which he uses to update the Q-table and improve its decision-making for future. Update Rule:

$$Q(S,A) = Q(S,A) + \alpha\left(R + \gamma argmax_a Q(S',a) - Q(S,A)\right)$$

B. Double Q-Learning Algorithm

We enhance the standard Q-learning algorithm by aiming to reduce overestimation bias in action value estimates encountered during standard Q-learning algorithm. We set up and initialise two separate Q-tables which is updated alternately during training. When choosing an action, Pixel Piero combines the values from both Q-tables to determine the best action based on the sum of their estimates. During the update phase, he randomly selects any Q-table for updating while utilising the other table to calculate the target value. Similar update rule is utilized for Q_2 also.

$$Q_1(\mathrm{S},\mathrm{A}) = Q_1(\mathrm{S},\mathrm{A}) + \alpha(\mathrm{R} + \gamma Q_2(S', argmax_a Q_1(\mathrm{S}',\mathrm{a})) - Q_1(\mathrm{S},\mathrm{A}))$$

C. Triple Q-Learning Algorithm

Similarly, we implement Triple Q-Learning algorithm which builds upon the principles of standard Q-learning and aims to improve action value estimates by maintaining three separate Q-tables. This setup might mitigate overestimation

bias during the learning process but may lead to significant computational costs. Similar to Double Q-Learning algorithm, Pixel Piero selects actions based on the combined values from all three Q-tables. During the update phase, he randomly chooses one of the three tables for updating while utilising the maximum sum value from the other two tables to calculate the target for the chosen table. Similar update rules are utilized for Q_2 and Q_3 also.

$$Q_1(\mathrm{S},\mathrm{A}) \leftarrow Q_1(\mathrm{S},\mathrm{A}) + \alpha(\mathrm{R} + \gamma Q_2(\mathrm{S}', \mathrm{argmax}_a(Q_2(\mathrm{S}',\mathrm{a}) + Q_3(\mathrm{S}',\mathrm{a}))) - Q_1(\mathrm{S},\mathrm{A}))$$

D. Quadruple Q-Learning Algorithm

Similarly, we implement Quadruple Q-Learning algorithm, which enhances standard Q-learning algorithm by utilizing four separate Q-tables. This approach aims to reduce overestimation bias in action value estimates by updating each Q-table based on the maximum sum estimated values from the other three tables but might lead to significant computational costs. Pixel Piero chooses actions based on the combined Q-values of all four tables, and during the update phase, it randomly selects one of the tables to update. Similar update rules are utilized for Q_2, Q_3 and Q_4 also.

$$Q_1(\mathrm{S},\mathrm{A}) \leftarrow Q_1(\mathrm{S},\mathrm{A}) + \alpha(\mathrm{R} + \gamma Q_2(\mathrm{S}', argmax_a(Q_2(\mathrm{S}',\mathrm{a}) + Q_3(\mathrm{S}',\mathrm{a}) + Q_4(\mathrm{S}',\mathrm{a}))) - Q_1(\mathrm{S},\mathrm{A}))$$

Fig 4. Q-Learning Algorithm[1]

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state,\cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$
  until S is terminal

Fig 5. Double Q-Learning Algorithm[1]

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal,\cdot) = 0$
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probability:
      $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\left(R + \gamma Q_2\left(S', \mathrm{argmax}_a Q_1(S',a)\right) - Q_1(S,A)\right)$
    else:
      $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\left(R + \gamma Q_1\left(S', \mathrm{argmax}_a Q_2(S',a)\right) - Q_2(S,A)\right)$
    $S \leftarrow S'$
  until S is terminal

III. RESULTS

A. Beginner Quest Environment

We ran the variations of Q-Learning algorithm for 10000 episodes in Beginner Quest Environment and obtained the following results.

Table 2. Results for Beginner Quest Environment (10000 episodes)

Algorithm	Rewards	
	Final Total	***Average***
Q-Learning	-20	-22.0475
Double Q-Learning	-20	-21.5890
Triple Q-Learning	-20	-21.6280
Quadruple Q-Learning	-20	-21.8105

Fig 6. Total Rewards (10000 episodes)

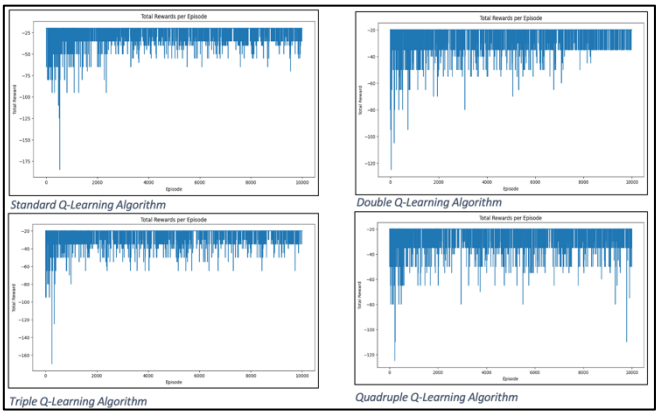
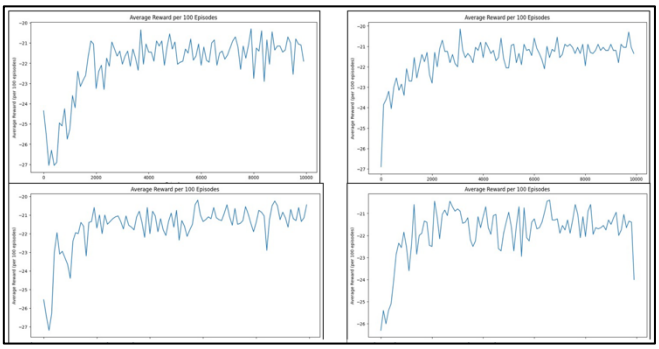


Fig 7. Average Rewards per 100 episodes (10000 episodes)



B. Superior Quest Environment

We ran the variations of Q-Learning algorithm for 100000, 200000 and 1000000 episodes for Superior Quest Environment and obtained the following results.

Table 3. Results for Superior Quest Environment (100000 episodes)

Algorithm	Rewards		Training Time (s)
	Final Total	***Average***	
Q-Learning	475	343.43073	103.51
Double Q-Learning	473	340.68952	120.77
Triple Q-Learning	476	333.24044	146.91
Quadruple Q-Learning	475	330.67928	157.58

Table 4. Results for Superior Quest Environment (200000 episodes)

Algorithm	Rewards		Training Time (s)
	Final Total	***Average***	
Q-Learning	469	396.011925	237.06
Double Q-Learning	476	397.865155	260.47
Triple Q-Learning	476	395.252725	303.45
Quadruple Q-Learning	471	391.14675	345.36

Table 5. Results for Superior Quest Environment (1000000 episodes)

Algorithm	Rewards		Training Time (s)
	Final Total	*Average*	
Q-Learning	455	427.987424	1221.36
Double Q-Learning	477	444.867258	1437.92
Triple Q-Learning	462	429.116813	1669.12
Quadruple Q-Learning	470	436.996807	1810.57

Fig 8. Total Rewards (1000000 episodes)

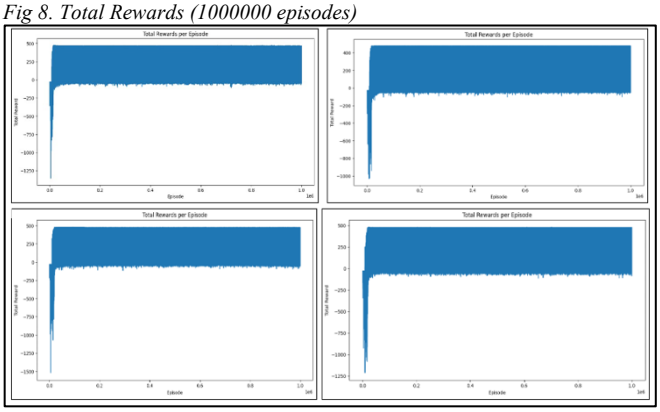
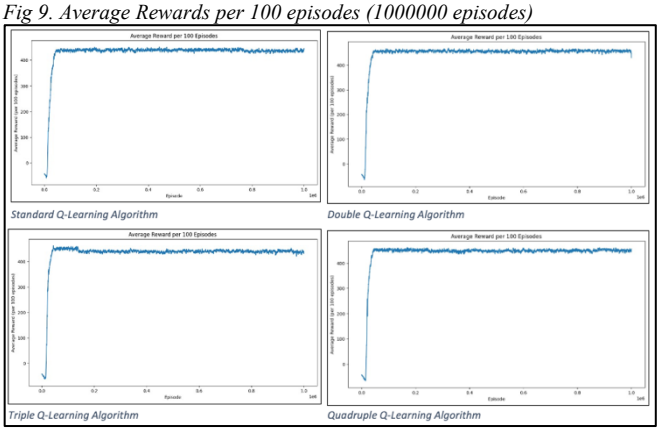


Fig 9. Average Rewards per 100 episodes (1000000 episodes)



C. *Elite Quest Environment*

We ran the variations of Q-Learning algorithm for 10000000 episodes in Elite Quest Environment and obtained the following results.

Table 6. Results for Elite Quest Environment (1000000 episodes)

Algorithm	Average Reward	Standard Deviation
Q-Learning	463.658097	249.79071394869746
Double Q-Learning	419.442759	58.79865069880431
Triple Q-Learning	469.16854	18.849378875708346
Quadruple Q-Learning	436.996807	62.176874792803794

Fig 10. Total Rewards (1000000 episodes)

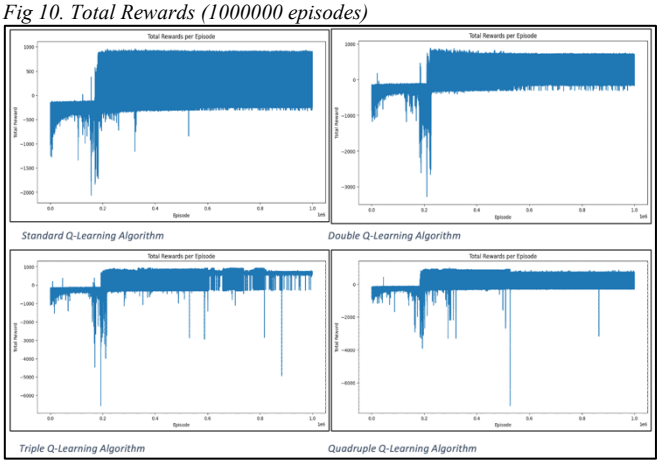
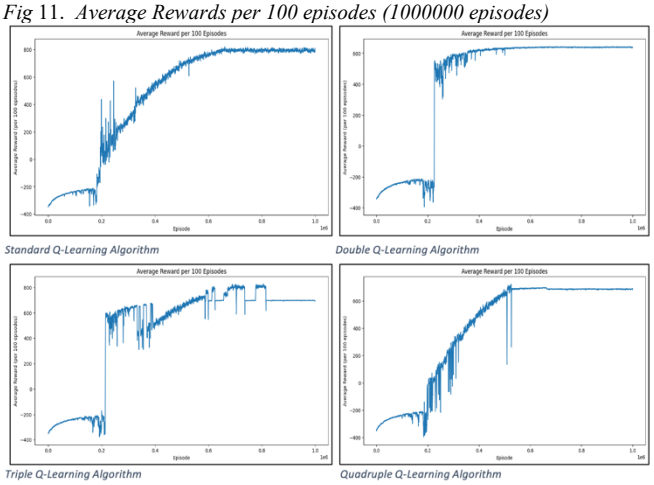


Fig 11. Average Rewards per 100 episodes (1000000 episodes)



IV. COMPARATIVE ANALYSIS

For the Beginner Quest Environment, the results demonstrated that all four algorithms achieved same final total reward, indicating that none were able to secure a positive outcome within the environment given. The average rewards varied slightly across the approaches, with Double Q-Learning exhibiting the highest average rewards, but, the overall difference in average reward accumulation remained minimal across all algorithms. These results suggest that for a simplistic environment, advanced variations of Q learning algorithm may not provide significant advantage over standard Q-learning algorithm.

Hence, we move on to upgraded Superior Quest Environment for better analysis of the implemented algorithms. We analysed the performance of variations of Q-Learning algorithms across three distinct training durations within the Superior Quest Environment. The results mentioned above indicate that all algorithms exhibit improved performance metrics with an increase in the number of training episodes. Notably, Double Q-Learning consistently achieved the highest average reward across all episode counts. Also, the training times increased significantly as we increased the episode count and moved from standard Q-Learning to Quadruple Q-Learning algorithms.

Now, while moving on to Elite Quest Environment, we observe that Q-Learning algorithm gave high average reward but the results exhibit high variability, likely due to overestimation bias. Double Q-Learning improved stability significantly by reducing the bias but increased the training time and a drop in average rewards obtained. Triple Q-Learning performed best here as it minimised variability and gave highest average rewards but the cost of training time increases due to maintaining three Q-tables instead of two in Double Q-Learning algorithm. On the contrary to expectations, Quadruple Q-Learning underperformed here which is because of increased complexity and careful coordination required among four different q-tables estimating action values. The additional complexity might have introduced computational overhead and potential redundancy or conflict in updates, amplifying noise and causing more conservative action choices without significantly enhancing reward outcomes.

V. Conclusion

A comparison of various Q-learning algorithms revealed that Double Q-Learning consistently outperforms other techniques. This advantage is attributable to its ability to reduce overestimation bias by maintaining two distinct Q-tables for action-value pairings. In the Double Q-Learning method, actions are selected based on one value function while being updated with the other, thereby minimizing the overestimation bias inherent in traditional Q-Learning. This results in more accurate and consistent learning outcomes. Furthermore, Double Q-Learning achieves an optimal balance between exploration and exploitation, which is crucial for effective learning. Double Q-Learning facilitates sufficient exploration without excessively favouring known values, hence enhancing overall performance and enabling the selection of optimal actions.

In contrast, while we anticipated that Triple and Quadruple Q-Learning algorithms would outperform both standard Q-Learning and Double Q-Learning, they underperformed in the Superior Quest Environment. However, in the Elite Quest Environment, characterized by increased complexity, maintaining three separate Q-tables by Triple Q-Learning yielded positive results, minimizing variability and achieving the highest average rewards. Conversely, Quadruple Q-Learning's underperformance might be due to increase in complexity, which could disrupt the balance between exploration and exploitation. Therefore, as the number of value estimates increases, the associated complexity may lead to diminishing returns in performance. This increased complexity of Q-tables could introduce instability during training, resulting in less satisfactory outcomes. Moreover, the additional computational burden linked to Triple and Quadruple Q-Learning may further hinder their efficiency, as the time and resources required to manage multiple value estimates can limit effective learning within a given timeframe.

In conclusion, Double Q-Learning emerges as the most suitable choice among the four algorithms for reasonably complex environments due to its effectiveness in reducing overestimation bias, achieving a balanced exploration-exploitation trade-off, and ensuring training stability, while jumping to higher order Q-Learning algorithms may be fruitful if the environment is highly complex in nature and we have large amount of computational power and costs available. The underperformance of Triple and Quadruple Q-Learning in first two environments highlights the importance of selecting algorithms specific to the challenges of the environment, showing that increased complexity does not necessarily correlate with enhanced performance.

We believe that when choosing between variations of Q-Learning algorithms in CLIF-style environments like Q-Quest, a deeper understanding of the exploration-exploitation trade-off and the need for efficient learning, without incurring excessive computational costs, is essential. By carefully selecting the most appropriate technique, we can optimize Pixel Piero's gameplay, enabling him to make optimal decisions at each state and ultimately achieving higher overall rewards.

VI. References

[1] Sutton, R. S., & Barto, A. G. (2014). Reinforcement Learning: An Introduction (Second edition, in progress). The MIT Press. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

The code developed for formulating the problem, implementing variations of Q-Learning methods along with obtained results and additional graphs and plots for other episodes mentioned in the report can be found at bPratyush/CS-F317-Reinforcement-Learning-Assignments: Coursework Assignments for CS F317 Reinforcement Learning