**Part 1:**

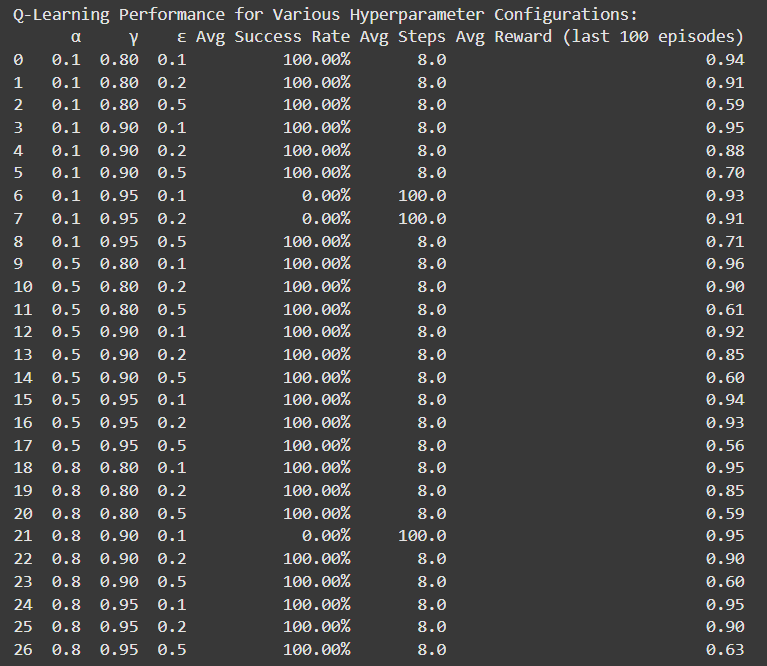
**Q-Learning and Policy Iteration on the Frozen Lake Environment**

In the Q-learning algorithm, adjusting the learning rate (α) significantly impacts the agent's learning behaviour. A low learning rate (α = 0.1) results in slower, more cautious updates to Q-values, which prolongs convergence but ensures stability and eventual success with sufficient training and exploration (e.g., ε = 0.1 or 0.2). However, this may lead to slightly lower average rewards due to delayed policy refinement. A medium learning rate (α = 0.5) offers a balanced trade-off between learning speed and stability, leading to faster convergence and consistently high rewards across various ε and γ settings. In contrast, a high learning rate (α = 0.8) enables rapid updates and quicker convergence but increases the risk of instability, especially when combined with poor exploration or high discount factors. When carefully tuned with appropriate ε and γ values, a high α can yield excellent performance, though it requires more precise parameter selection.

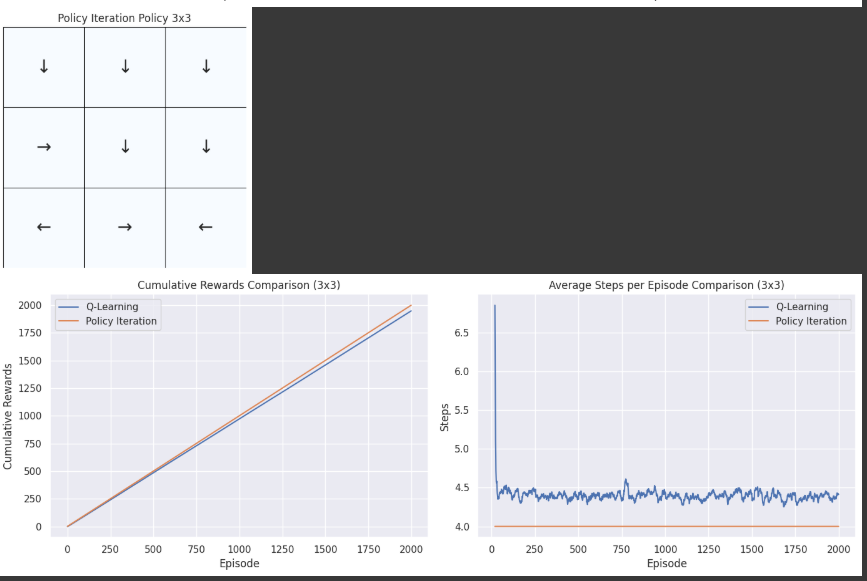
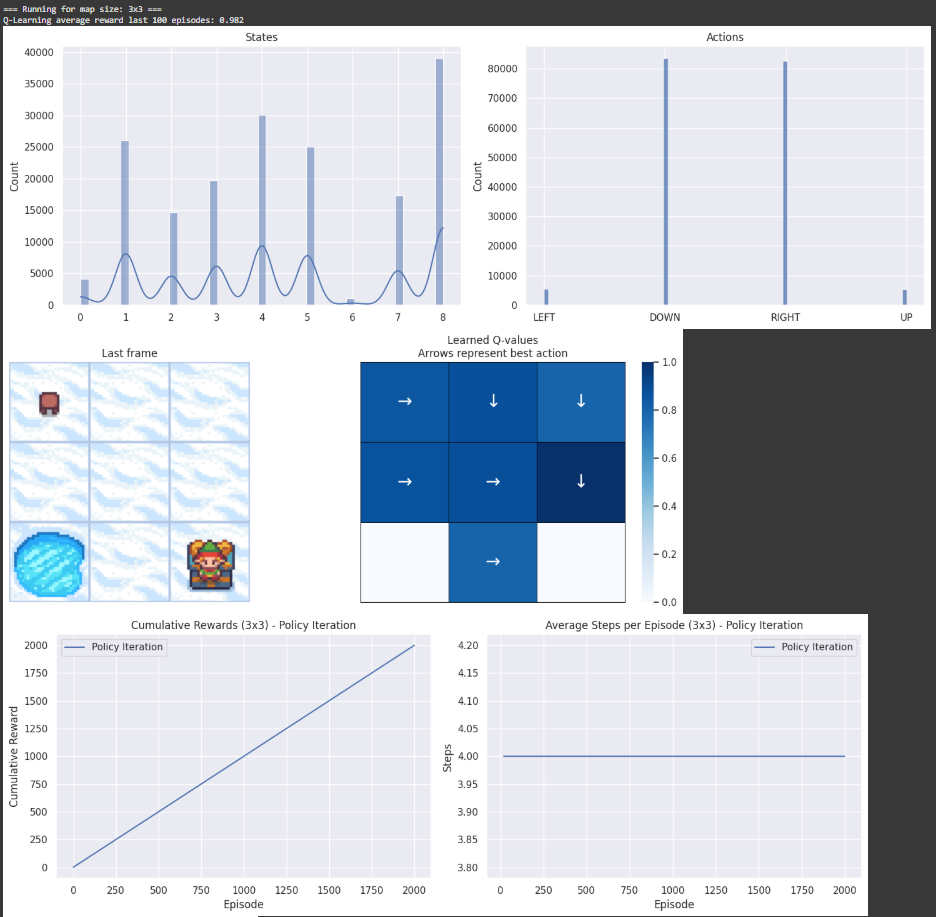
The discount factor (γ) greatly affects how an agent balances short- and long-term rewards in Q-learning. A lower γ (0.80) Favors immediate rewards and works well for short-horizon tasks, offering stable and consistent performance. A medium γ (0.90) provides the best balance, often leading to the highest average rewards and showing robustness across different hyperparameter settings. A higher γ (0.95) emphasizes long-term rewards but can hinder goal achievement in stochastic environments unless paired with sufficient exploration (ε ≥ 0.2). Overall, γ should be carefully tuned alongside ε and α to optimize performance.

The exploration rate (ε) significantly influences how the agent balances exploration and exploitation in Q-learning. A low ε (0.1) generally delivers the best performance by favouring exploitation of the learned policy while still allowing minimal exploration. It is ideal once a reliable policy is in place but can lead to premature convergence on suboptimal paths if the initial policy is poor or γ is high (as seen in row 21). A moderate ε (0.2) offers a balanced approach, providing enough exploration during training without severely impacting performance. It typically results in slightly lower rewards than ε = 0.1 but outperforms higher exploration settings. A high ε (0.5), on the other hand, introduces too much exploration, which can degrade policy quality and reward outcomes, despite achieving full success rates. This setting is only effective when combined with ε-decay strategies that reduce exploration over time.

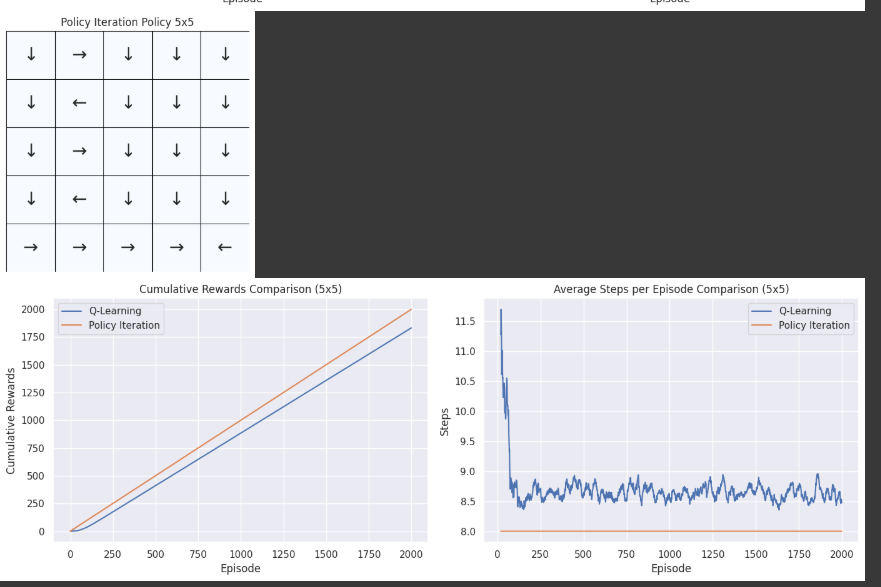
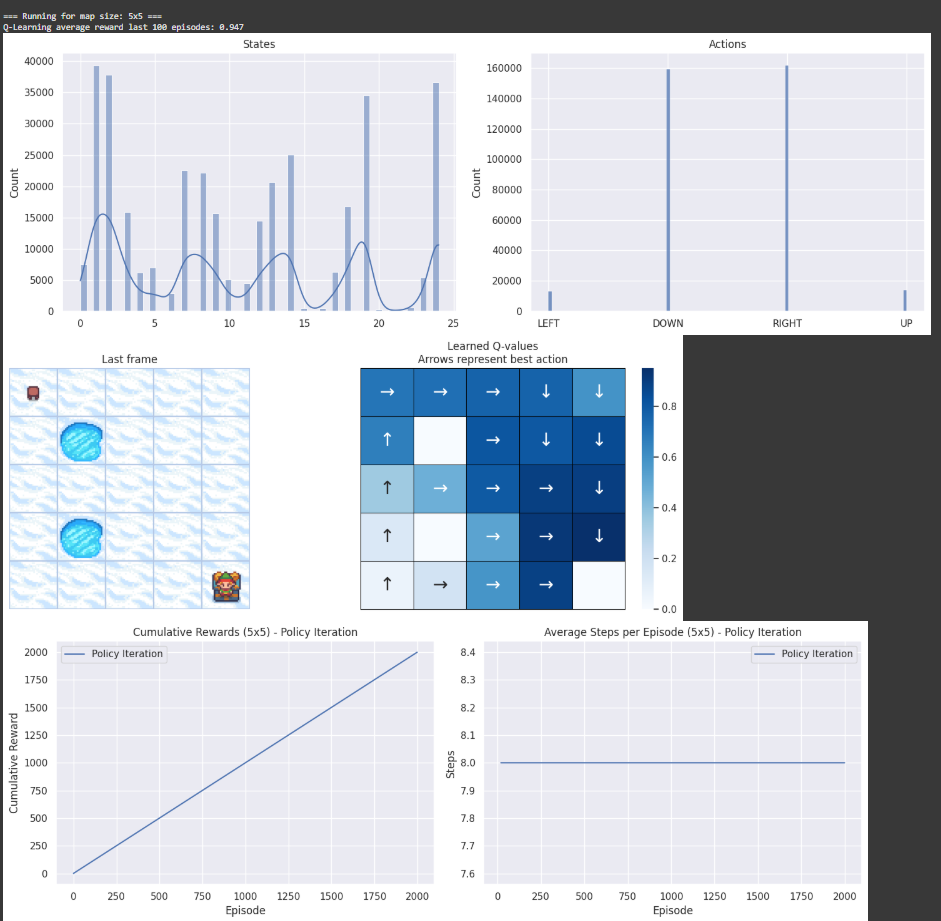
Over exploration with a high ε throughout the entire training phase can hinder long-term performance, even if the success rate stays high. Similarly, a high γ can make the agent overly cautious, causing it to take safe actions that avoid the goal, even though rewards might still appear high. The best results typically come from balanced configurations, such as α = 0.5, γ = 0.9, and ε = 0.1, which lead to high rewards, quick convergence, and consistent success. For stochastic environments like Frozen Lake, moderate exploration is beneficial early on, but as the agent learns, the policy should shift toward exploitation. This makes lower ε values or ε decay strategies ideal for improving performance as training progresses.



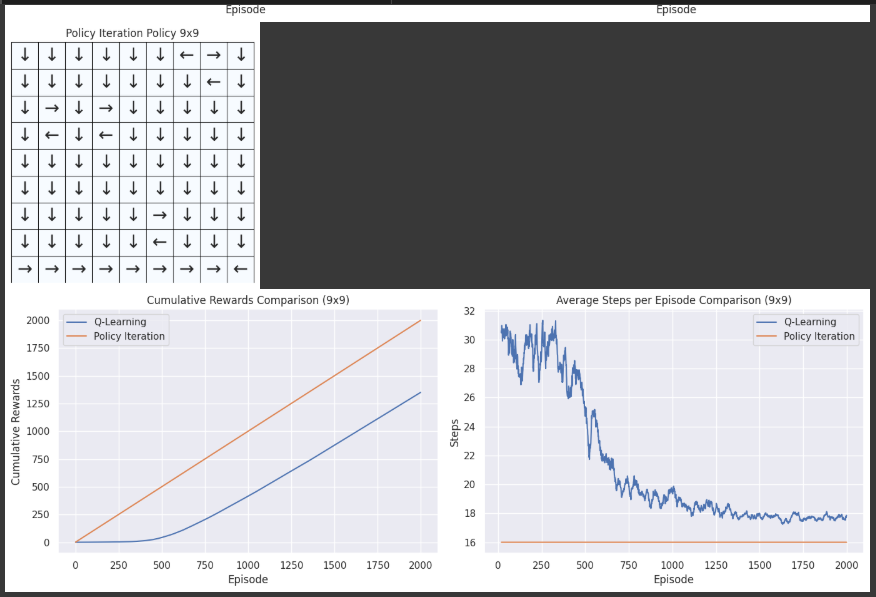
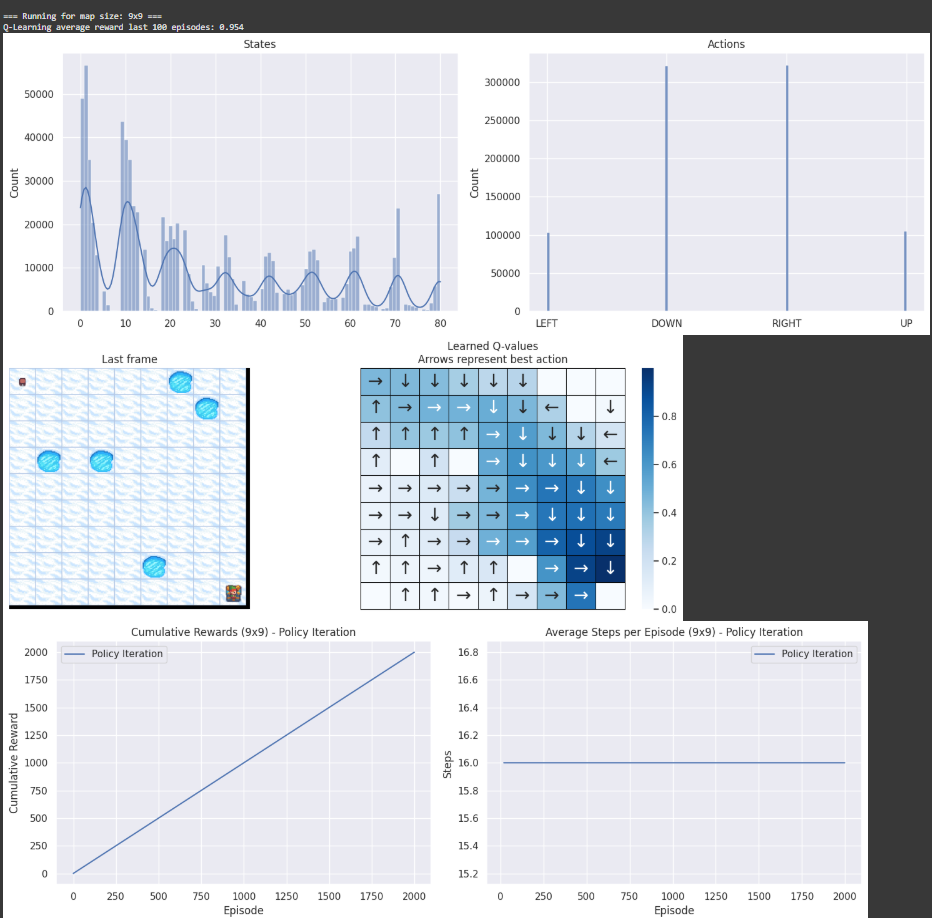
3x3



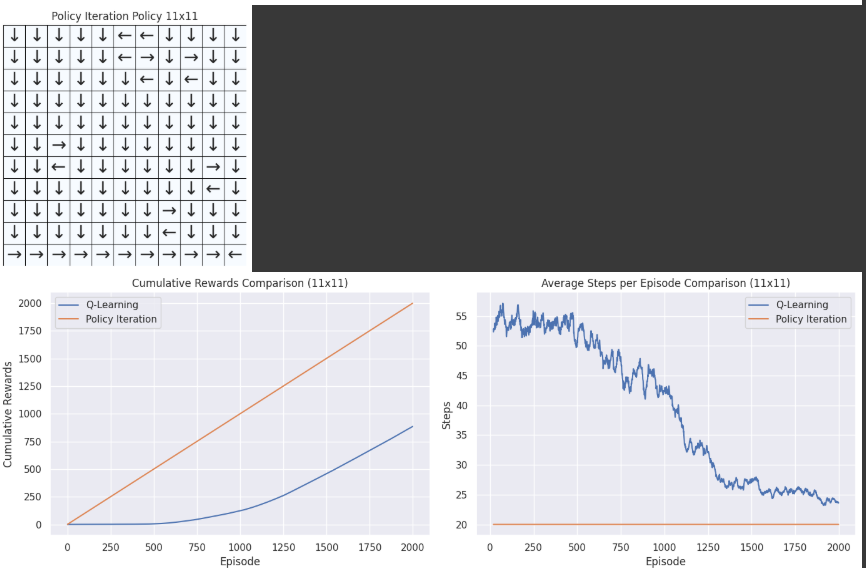
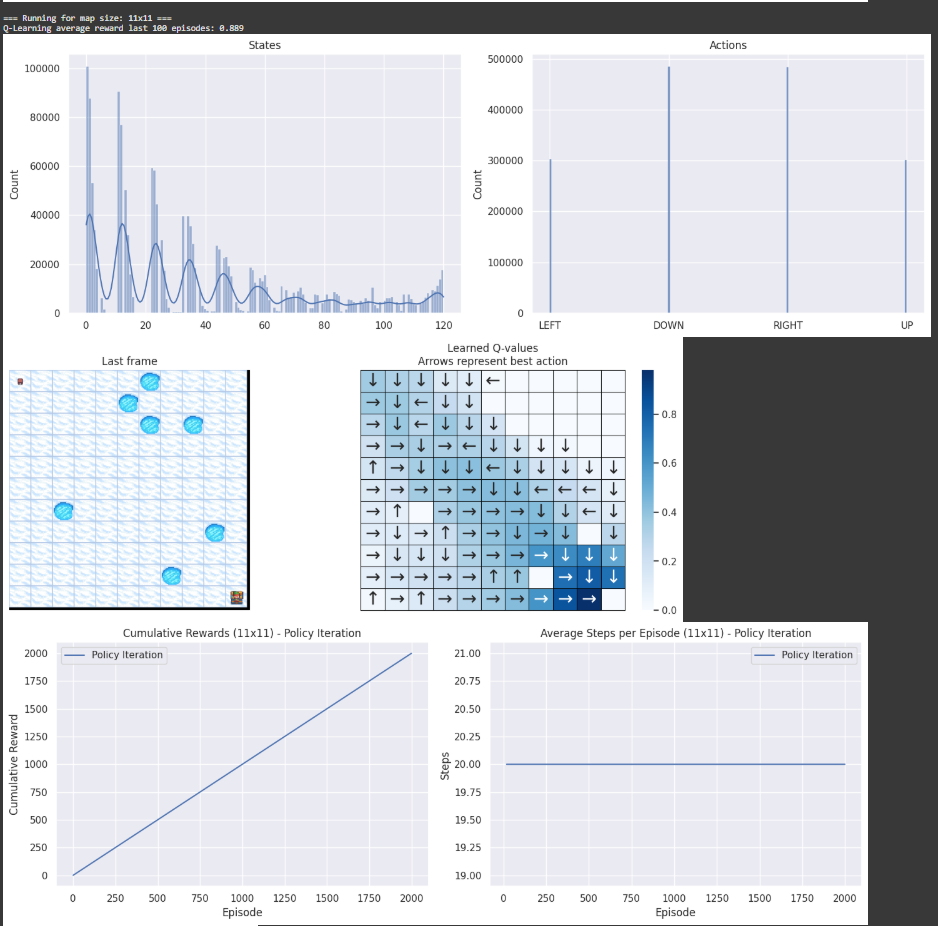
5x5

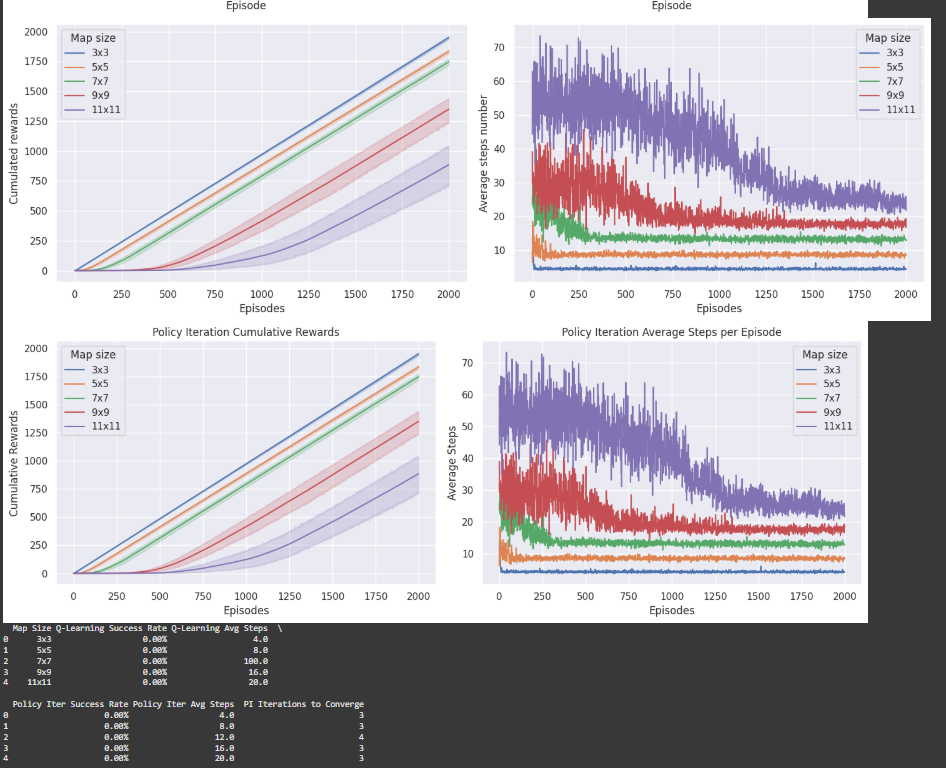


9x9



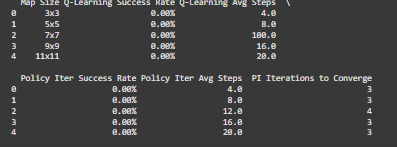
11x11





**Implement the policy iteration algorithm and compare its performance to Q-Learning.**

Both Q-Learning and Policy Iteration (PI) show a 0.00% success rate across all map sizes in the table, though they differ in their performance characteristics. Q-Learning requires increasing numbers of steps for larger maps (e.g., 4 steps for 3x3 and 20 for 11x11). It is model-free, flexible, and can learn incrementally in unknown environments, but it suffers from slow convergence, sensitivity to hyperparameters, and inefficiency in larger state spaces. On the other hand, Policy Iteration also shows an increase in steps (4 for 3x3, 20 for 11x11) and converges in 3-4 iterations. While it offers stable convergence and guarantees optimal policies in smaller, deterministic environments, it requires a complete model of the environment, struggles in stochastic settings, and can be computationally expensive. In conclusion, Q-Learning is more adaptable to unknown environments but struggles with slow learning, while Policy Iteration converges more quickly but relies on having a full model and is less effective in stochastic or larger state spaces. Both methods need careful tuning for optimal performance.



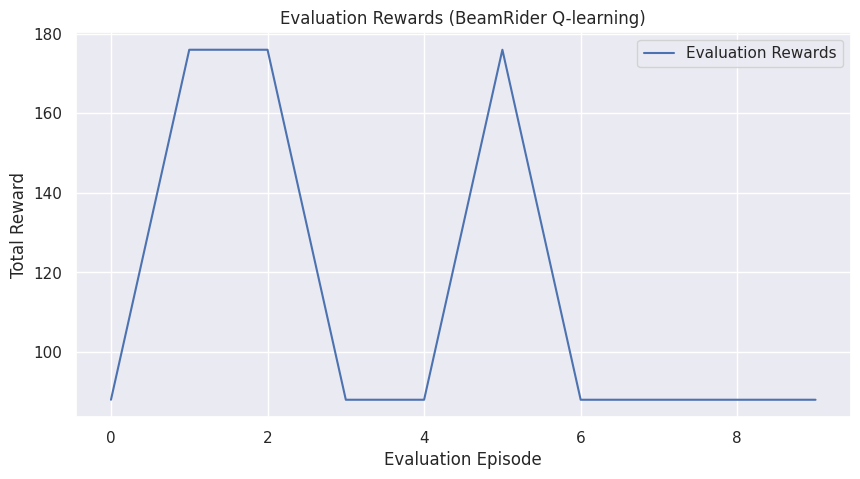
**Part 2:**

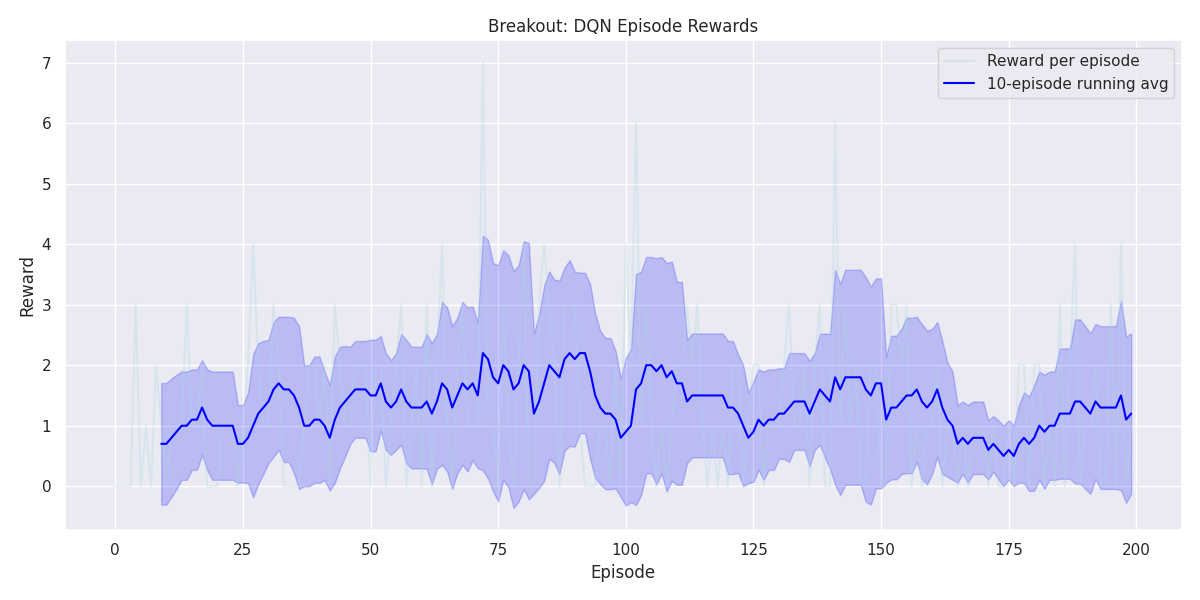
**Deep Q-Learning on an Atari Game Environment**

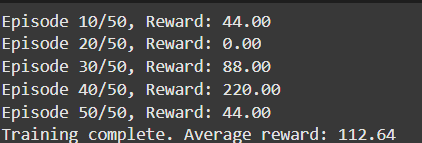
Here we implement a Q-learning agent for the *Beam Rider* game using a tabular approach. It preprocesses game frames by converting them to grayscale and resizing them for easier processing. The agent discretizes these pre-processed observations into bins and uses a Q-table to estimate action values. Through the Q-learning algorithm, the agent explores the environment using an epsilon-greedy policy, updating its Q-table based on rewards. Over multiple episodes, the agent shifts from exploration to exploitation as epsilon decays. After training, the agent’s policy is evaluated by running it through test episodes without exploration. The code also plots the training and evaluation rewards to visualize the agent's learning progress.

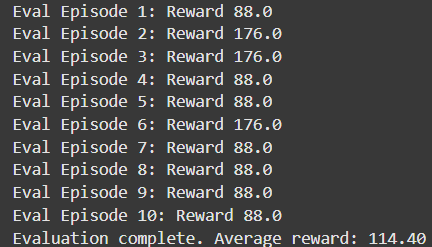
Beam Rider is a classic 3D arcade shooter where players control a spaceship, moving along five beams, to destroy enemy ships, dodge obstacles, and collect extra lives while facing increasing difficulty. The Deep Q-Learning (DQN) algorithm applied to this environment replaces a traditional Q-table with a convolutional neural network (CNN) to predict action values from game frames. The agent uses a replay buffer to store experiences and learns by sampling random batches to update its policy. An epsilon-greedy strategy balances exploration and exploitation, while periodic updates to a target network stabilize learning. The goal is for the agent to maximize its score by optimizing its actions to survive longer and destroy enemies effectively.











Here we see that initially the total reward per episode is low and fluctuating, indicating the agent is exploring and has not yet learned a stable policy. There is slight imporvment in reward after 100 episodes which tell us that the model is learning to keep the pole balanced for longer durations and we see that we have an average reward of about 114 .