Assignment 2- Deep Q-Networks

Part II - Implementing DQN & Solving Various Problems:

1. Discuss the benefits of:

• Using experience replay in DQN and how its size can influence the results

Experience replay is a crucial component of Deep Q-Networks (DQN) that enhances stability and efficiency in learning.

- Breaks Temporal Correlation: In standard Q-learning, consecutive experiences are highly correlated, which can lead to inefficient updates and instability. Experience replay allows the agent to sample from a diverse set of past experiences, reducing correlation and improving generalization.
- Better Data Utilization: Instead of using each transition once and discarding it, replaying experiences multiple times helps improve learning efficiency, especially in environments with sparse rewards.
- Stabilizes Training: By maintaining a large buffer of past experiences, updates become smoother and less sensitive to small changes in the policy.

Impact of Replay Buffer Size:

- Too Small: If the buffer size is too small, the experiences will be highly correlated, reducing the benefits of replay and making the agent more prone to overfitting recent experiences.
- Too Large: If the buffer is too large, older experiences might no longer be relevant to the current policy, potentially slowing down adaptation to new strategies.
- Optimal Size: A properly tuned buffer size balances diversity and relevance, ensuring stable and efficient learning.

• Introducing the target network.

The target network is a copy of the Q-network that updates more slowly, preventing rapid fluctuations in Q-value estimates.

- Reduces Instability: Since the Q-values depend on themselves (bootstrapping), updating them with a constantly shifting target can lead to divergence. The target network smooths updates by using old Q-values for a fixed period.
- Improves Convergence: By reducing the variance in Q-value estimates, the target network prevents feedback loops where overestimated Q-values reinforce themselves.
- Acts as a Stable Reference: Instead of directly updating Q-values with estimates from the same network, the target network provides a more stable estimate, ensuring controlled learning.

• Representing the Q function as $\hat{q}(s,w)$.

Instead of representing the Q-function with a tabular approach, function approximation is used:

- Generalization Across States: Using a parameterized function q̂(s,w) (e.g., a neural network) enables learning in high-dimensional state spaces where tabular methods fail.
- Efficient Storage: Rather than storing Q-values for all possible state-action pairs, the function learns a mapping, making it scalable to complex environments.
- Enables Deep Learning: The function q̂(s,w) can be a deep neural network, allowing for non-linear representations and better feature extraction from high-dimensional inputs.

2. Briefly describe ‘CartPole-v1’, the second complex environment, and the grid-world environment that you used (e.g. possible actions, states, agent, goal, rewards, etc). You can reuse related parts from your Assignment 1 report.

CartPole-v1: CartPole is a classic reinforcement learning (RL) environment where the goal is to balance a pole on a moving cart.

- State (Observation Space):

  The environment provides a continuous state space represented by:

    - Cart position (x-coordinate)
    - Cart velocity
    - Pole angle
    - Pole angular velocity
- Actions (Action Space): The agent can take two discrete actions:
  - Move cart left (0)
  - Move cart right (1)
- Goal: Prevent the pole from falling by keeping it balanced for as long as possible.
- Rewards: The agent receives +1 reward for every time step the pole remains upright. The episode ends if the pole falls past a certain angle or if the cart moves out of bounds.
- Termination Conditions:
  - The pole tilts more than ±12 degrees.
  - The cart moves more than ±2.4 units from the center.
  - The agent survives for 500 time steps (solved threshold).

2. FrozenLake-v1: FrozenLake is a grid-based RL environment where an agent navigates a slippery surface to reach a goal while avoiding holes.

- State (Observation Space): The environment consists of a 4X4 grid where each tile is either:
  - Start (S) – The agent’s starting position.
  - Frozen (F) – A safe tile where the agent can move.

    - Hole (H) – A dangerous tile where the agent falls and the episode ends.
    - Goal (G) – The target destination.

The state is a single discrete integer representing the agent’s position in the grid (from 0 to 15).

- Actions (Action Space): The agent can take four discrete actions:
    - Move left (0)
    - Move down (1)
    - Move right (2)
    - Move up (3)
- Goal: Navigate from the start tile (S) to the goal tile (G) without falling into holes.
- Rewards:
    - +1 reward for reaching the goal (G).
    - 0 reward for stepping on frozen tiles (F).
    - Episode ends immediately if the agent falls into a hole (H).
- Stochasticity: The agent’s movements are not deterministic. There is a chance it will slip and move in an unintended direction, adding complexity to the learning process.

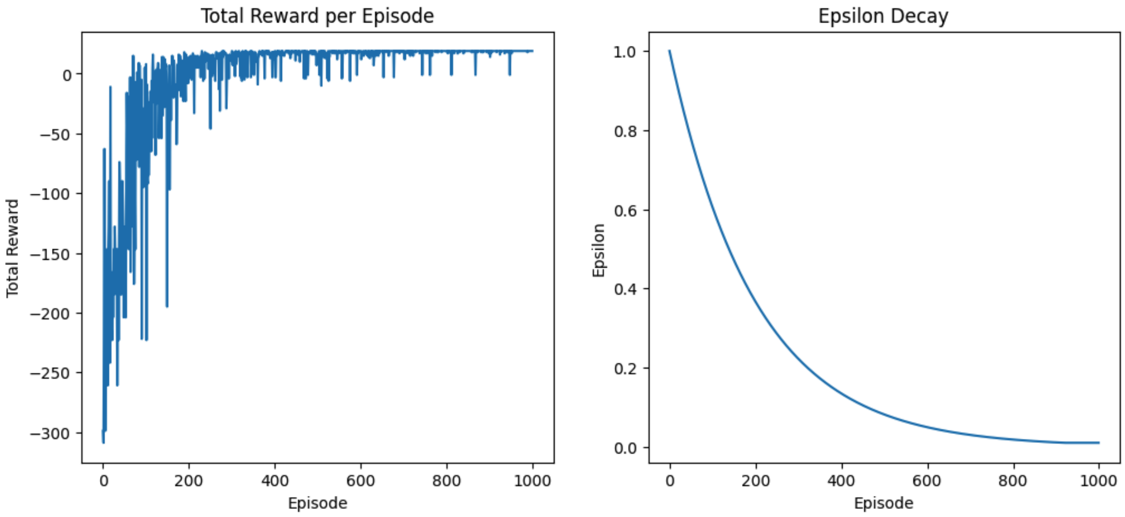
3. Custom Grid-World Drone Delivery (Env_3_Drone_Delivery): This is a custom grid-based environment where a drone picks up and delivers a package while avoiding obstacles.

- State (Observation Space):
    - Drone’s x, y position in the grid.
    - Whether the drone has picked up the package (0 or 1).
- Actions (Action Space): The drone can take six discrete actions:
    - UP (0)
    - DOWN (1)
    - LEFT (2)
    - RIGHT (3)
    - PICKUP (4) (at the warehouse)
    - DROPOFF (5) (at the delivery location)
- Goal: The agent must navigate from the warehouse (pickup location) to the customer (delivery target) while avoiding no-fly zones.
- Rewards:
    - -1 per step (encourages efficiency).
    - +10 for picking up the package.
    - +20 for successfully delivering the package.
    - -20 penalty for entering a no-fly zone.
- Additional Features:
    - No-fly zones act as obstacles.
    - Stochasticity: A 10% chance the drone moves in an unintended direction.
    - Episode ends after 100 steps or successful delivery.

3. Show and discuss your results after applying your DQN implementation on the three environments. Plots should include epsilon decay and the total reward per episode.

(The below results are achieved after implementing changes suggested in Part 2 Feedback)

1. Custom Grid-World Drone Delivery (Env_3_Drone_Delivery)



Interpretation of plots:

The results suggest that the DQN agent has successfully learned a good policy for the Custom Grid-World Drone Delivery environment.

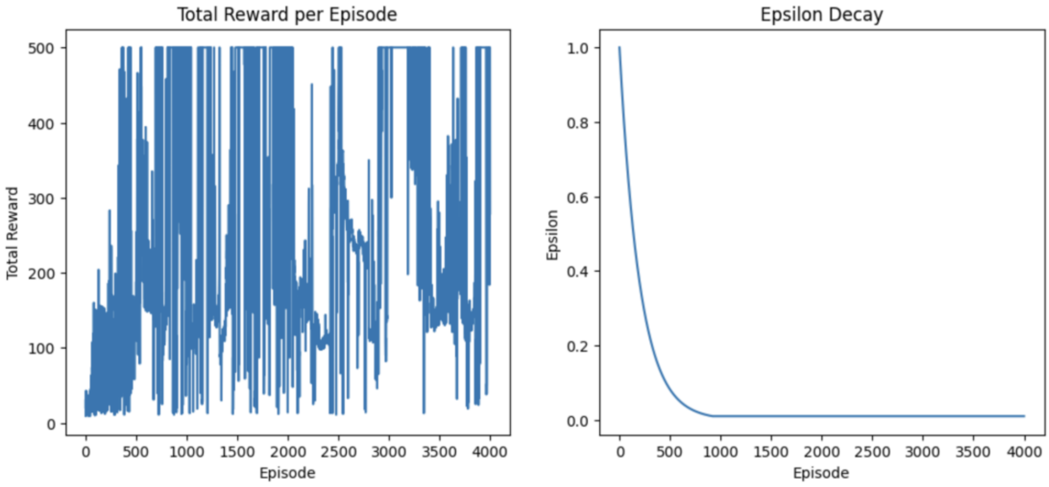
Total Reward Per Episode Plot:

1. **Initial Exploration:** The total reward per episode shows significant fluctuations and generally low values in the initial episodes (roughly the first 200-300 episodes). The agent explores the environment and is learning the consequences of its actions. The agent is likely trying out different strategies, some of which lead to very negative rewards.
2. **Learning Phase:** After the initial exploration phase, there's a noticeable and relatively rapid increase in the total reward per episode. This indicates that the DQN agent is starting to learn a better policy for navigating the grid-world and delivering packages.
3. **Convergence:** Around episode 300-400, the total reward appears to stabilize around a higher value, generally staying above 0 and often reaching positive values. This suggests that the DQN agent has learned a near-optimal or a reasonably good policy for the task. The fluctuations become smaller, indicating more consistent performance.

Epsilon Decay Plot:

1. Exploration-Exploitation Trade-off: This plot shows the decay of the epsilon (ε) value over episodes. Epsilon determines the probability of the agent taking a random action (exploration) versus taking the action with the highest estimated Q-value (exploitation). The graph shows a typical exponential decay of epsilon. It starts at a high value (presumably 1.0 or close to it), meaning the agent initially explores the environment almost randomly.
2. Gradual Shift to Exploitation: As the number of episodes increases, epsilon gradually decreases. This signifies a shift from exploration to exploitation. The agent becomes more confident in its learned Q-values and starts to rely more on the actions it believes will yield the highest reward.
3. Low Final Epsilon: By the end of the training (around 1000 episodes), epsilon has decayed to a very low value (close to 0). This means the agent is primarily exploiting its learned policy and rarely takes random actions.

2. Cartpole-v1 Environment



Interpretation of plots:

The results clearly demonstrate that the DQN agent has successfully learned to solve the CartPole-v1 environment.

Total Reward per Episode Plot:

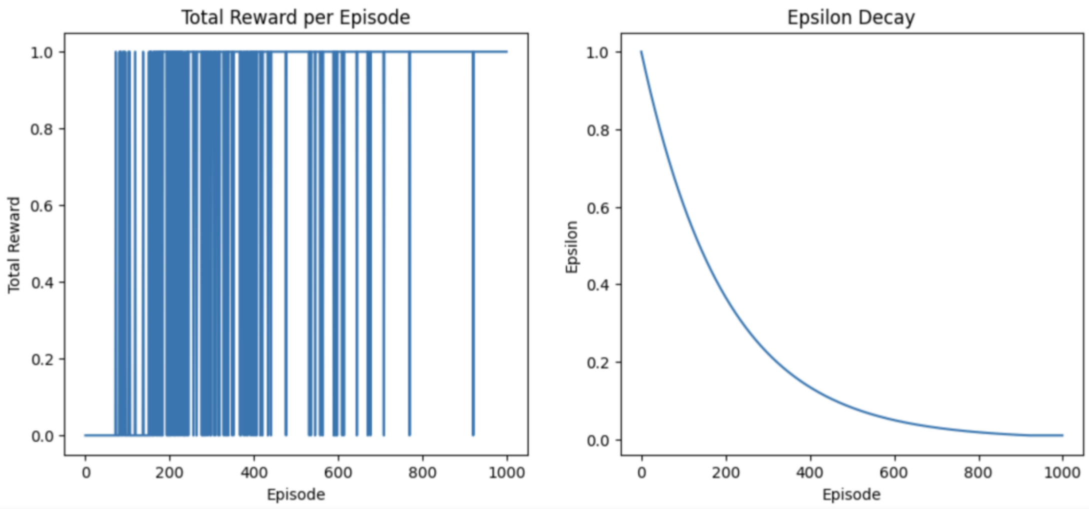
1. Initial Learning Phase: In the beginning episodes (roughly the first 100-300), the total reward per episode is relatively low and shows significant fluctuations. The agent is randomly trying actions and hasn't yet learned which actions lead to positive outcomes (keeping the pole balanced for a longer duration). The low rewards are indicating frequent episode terminations due to the pole falling or the cart moving out of bounds.

2. Rapid Performance Improvement: Following the initial exploration, there is a clear and rapid increase in the total reward per episode. This signifies that the DQN agent is starting to learn a successful policy for balancing the pole.
3. Reaching Maximum Reward: The total reward quickly climbs and starts to consistently reach the maximum possible reward for CartPole-v1, which is 500. The environment gives a reward of +1 for each timestep the pole remains balanced and the episode ends when the pole falls, the cart goes out of bounds, or the episode reaches 500 timesteps.
4. Sustained High Performance: After achieving the maximum reward, the total reward per episode remains consistently at or very close to 500 for the majority of the remaining episodes. This indicates that the DQN agent has learned a robust and effective policy to keep the pole balanced for the maximum duration. The agent has successfully solved the CartPole-v1 environment.

Plot of Epsilon Decay:

1. Exploration-Exploitation Trade-off: This plot shows the decay of the epsilon (ε) value over episodes. Epsilon determines the probability of the agent taking a random action (exploration) versus taking the action with the highest estimated Q-value (exploitation). The graph shows a typical exponential decay of epsilon. It starts at a high value (presumably 1.0 or close to it), meaning the agent initially explores the environment almost randomly.
2. Gradual Shift to Exploitation: As the training progresses and the number of episodes increases, the epsilon value gradually decreases. This signifies a shift from exploration towards exploitation. The agent becomes more confident in its learned Q-values/policy and increasingly relies on its policy.
3. Low Final Epsilon: By the end of the training (around 1000 episodes), the epsilon value has decayed to a very small value (close to 0). This implies that the agent is primarily exploiting its learned policy and rarely takes random actions. This is expected as the agent has learned a successful strategy.

3. Frozen Lake Environment



Interpretation of plots:

Total Reward per Episode Plot:

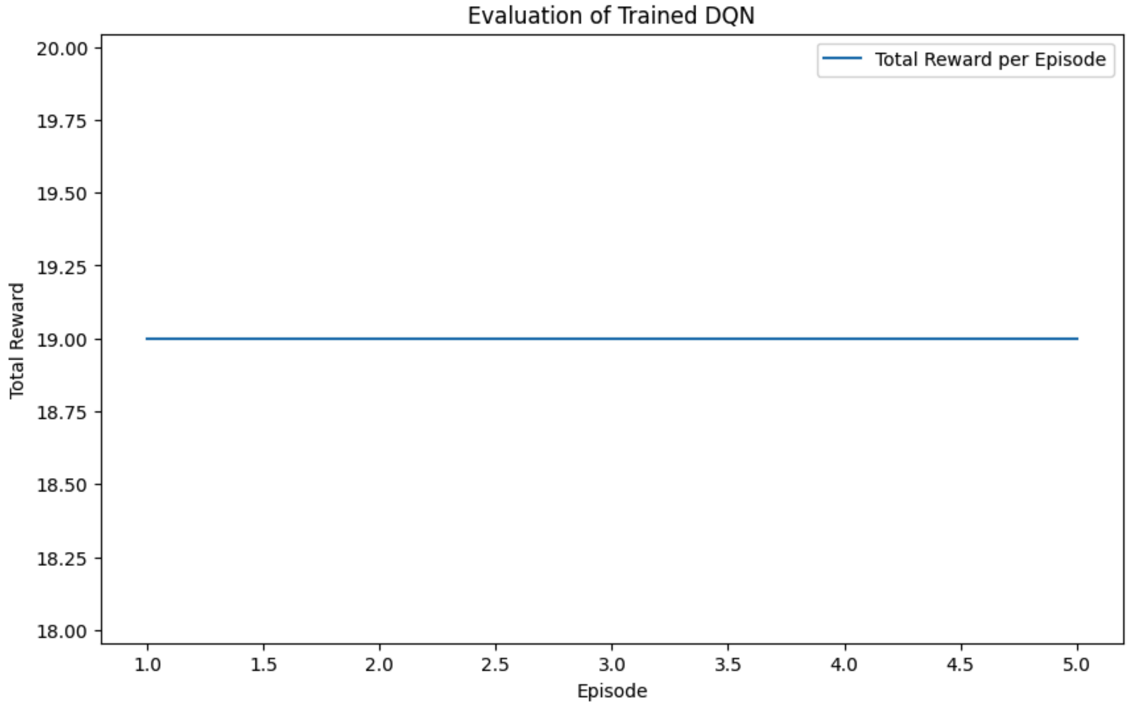
1. Initial Struggle: In this FrozenLake environment, the total reward per episode oscillates between 0 and 1. In the early episodes (roughly the first 150), the total reward is almost always 0. This indicates that the agent is primarily exploring randomly and frequently falling into holes or not reaching the goal within the episode limit.
2. Learning and Improvement: Around episode 150, we start to see episodes with a reward of 1. This signifies that the agent has begun to learn a policy that occasionally leads it to the goal. The frequency of successful episodes (reward = 1) gradually increases over time.
3. Fluctuating Performance: Even after the agent starts achieving the goal, the performance remains somewhat fluctuating. There are still many episodes where the reward is 0, and some episodes with episodes where the reward is 1. This is due to the stochastic nature of the FrozenLake environment (slippery ice). Even with a good policy, there's a chance of slipping and falling into a hole.
4. Gradual Increase in Success Rate: Over the later episodes (especially after episode 500), the proportion of successful episodes (reward = 1) appears to increase. While there are still failures, the agent is reaching the goal more consistently. This indicates that the learning process is ongoing and the policy is improving.
5. No Full Convergence: The total reward per episode never consistently stays at 1. This implies that the agent hasn't learned a deterministic optimal policy that avoids all risks of falling into a hole in the stochastic environment. It's possible that the learned policy has a high probability of reaching the goal but still carries some risk due to the slippery nature of the ice.

Epsilon Decay Plot:

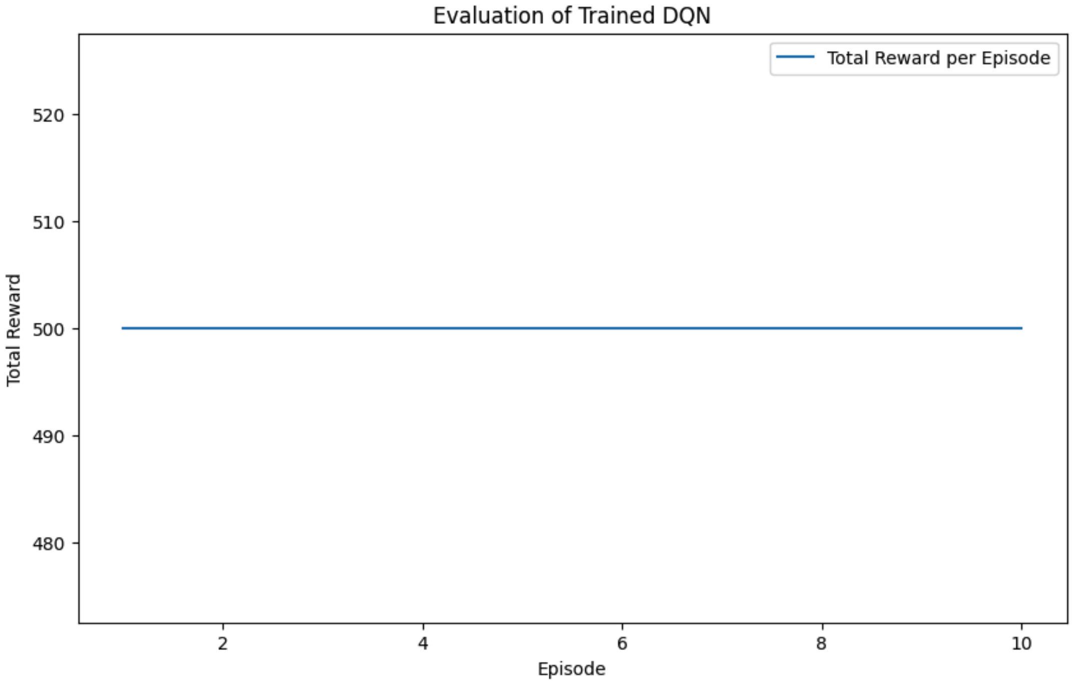
1. Exploration-Exploitation Trade-off: This plot shows the decay of the epsilon (ε) value over episodes. Epsilon determines the probability of the agent taking a random action (exploration) versus taking the action with the highest estimated Q-value (exploitation). The graph shows a typical exponential decay of epsilon. It starts at a high value (presumably 1.0 or close to it), meaning the agent initially explores the environment almost randomly.
2. Shift from Exploration to Exploitation: As the training progresses and the number of episodes increases, the epsilon value gradually decreases. This signifies a shift from exploration towards exploitation. The agent becomes more confident in its learned Q-values/policy and increasingly relies on its policy.

4. Provide the evaluation results. Run your agent on the three environments for at least 5 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.

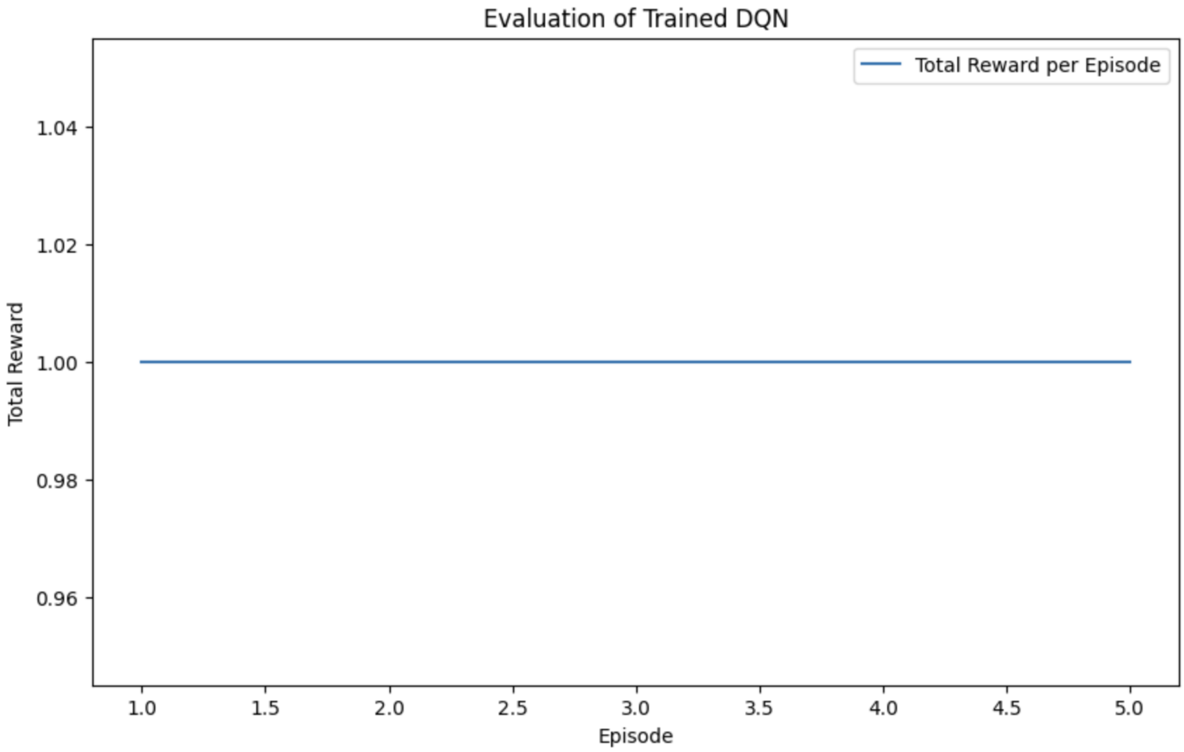
1. Custom Grid-World Drone Delivery (Env_3_Drone_Delivery)



2. Cartpole-v1 Environment

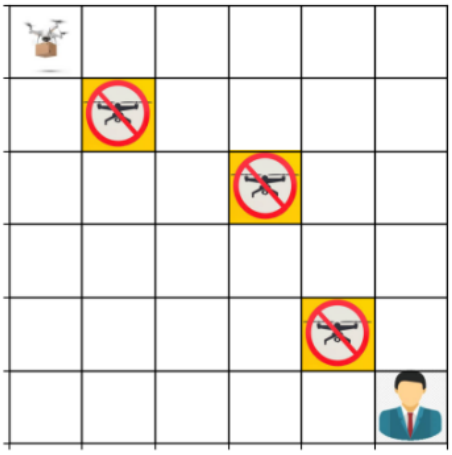


3. Frozen Lake Environment

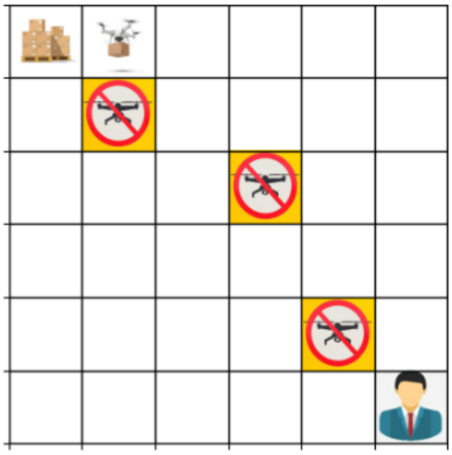


5. Grid World environments only: Run your environment for 1 episode where the agent chooses only greedy actions from the learned policy. Render each step of the episode and verify that the agent has completed all the required steps to solve the environment. Save this render and include it in your report as clearly ordered screenshots or as a clearly named video file in your submission.

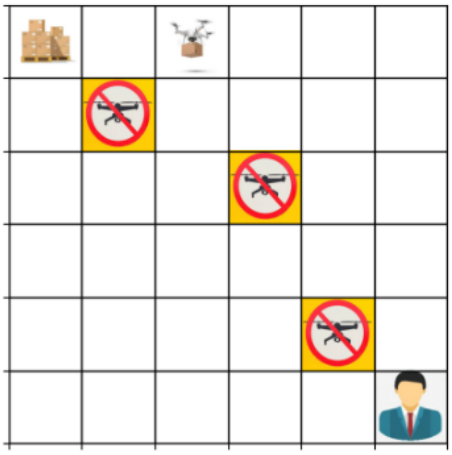
```
Step 1: Action PICKUP, Reward: 9, Position: (0, 0, True)
```



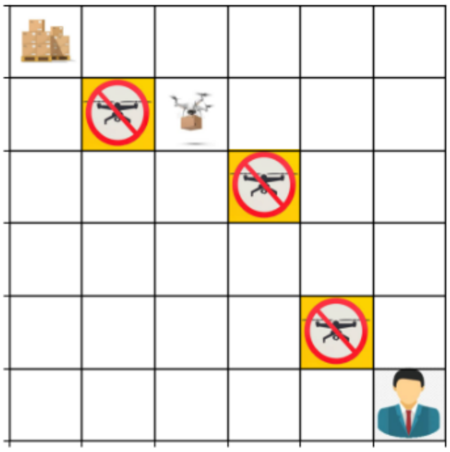
```
Step 2: Action RIGHT, Reward: -1, Position: (0, 1, True)
```



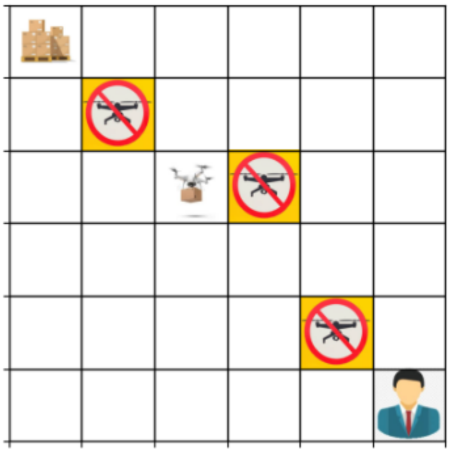
```
Step 3: Action RIGHT, Reward: -1, Position: (0, 2, True)
```



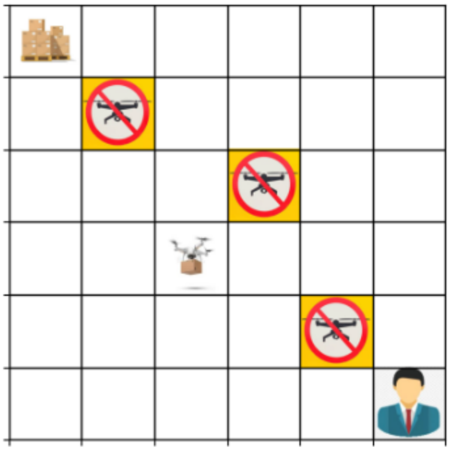
```
Step 4: Action DOWN, Reward: -1, Position: (1, 2, True)
```



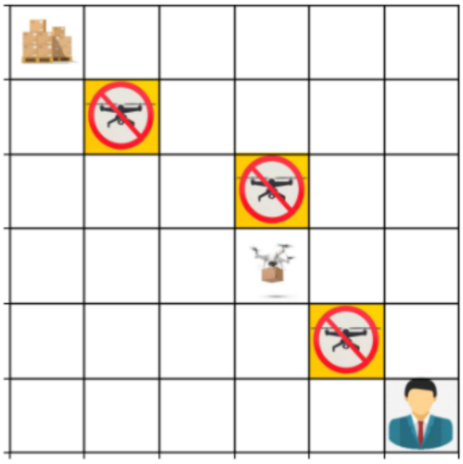
```
Step 5: Action DOWN, Reward: -1, Position: (2, 2, True)
```



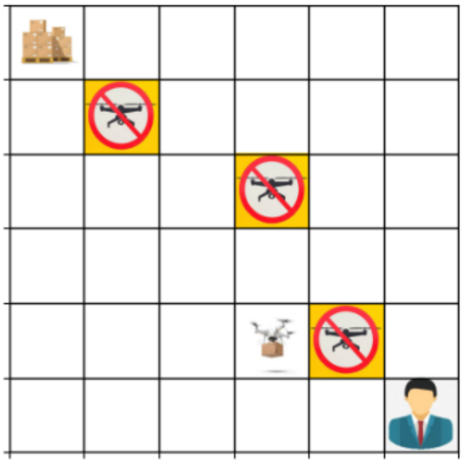
```
Step 6: Action DOWN, Reward: -1, Position: (3, 2, True)
```



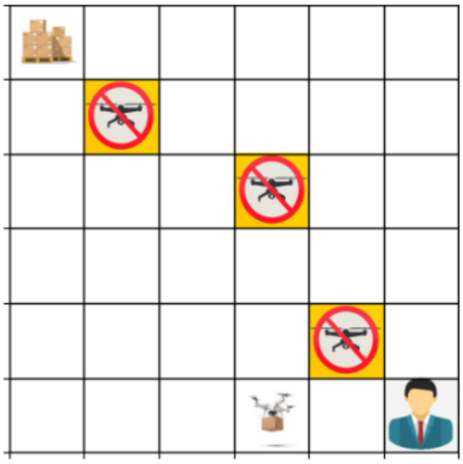
```
Step 7: Action RIGHT, Reward: -1, Position: (3, 3, True)
```



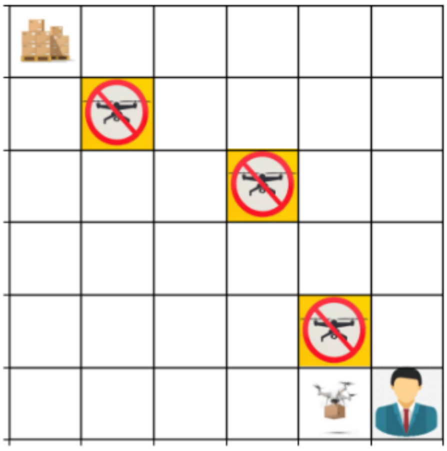
```
Step 8: Action DOWN, Reward: -1, Position: (4, 3, True)
```



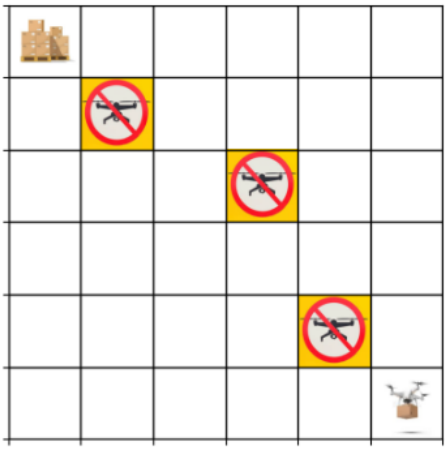
```
Step 9: Action DOWN, Reward: -1, Position: (5, 3, True)
```



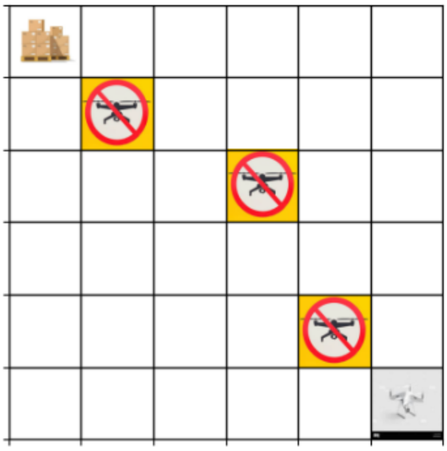
```
Step 10: Action RIGHT, Reward: -1, Position: (5, 4, True)
```



```
Step 11: Action RIGHT, Reward: -1, Position: (5, 5, True)
```



```
Step 12: Action DROPOFF, Reward: 20, Position: (5, 5, False)
```



```
Simulation finished after 12 steps with total reward 19
```

6. Provide your interpretation of the results. E.g. how the DQN algorithm behaves on different envs.

The DQN behaves differently across different environments.

1. Custom Grid-World Drone Delivery (Env_3_Drone_Delivery): The graphs showed initial struggle followed by a significant improvement in total reward, suggesting DQN learned a good policy for delivery. The stabilization of reward and epsilon decay indicate convergence.
2. CartPole-v1: DQN performed very well, quickly achieving and sustaining the maximum reward. This demonstrates DQN's effectiveness in a low-dimensional, continuous state space with a small, discrete action space and clear reward signal. The rapid learning and stable high reward are typical for DQN on this environment.
3. FrozenLake: DQN showed learning progress, with the agent reaching the goal more frequently over time. However, the reward fluctuated between 0 and 1, indicating the learned policy wasn't perfectly consistent. This is expected in the stochastic FrozenLake environment where the agent can slip. The epsilon decay shows a typical shift from exploration to exploitation.

7. Referenecs:

- https://gymnasium.farama.org/environments/classic_control/cart_pole/
- https://gymnasium.farama.org/environments/toy_text/frozen_lake/
- CSE546 RL Assignment 1 submitted by ruthvikv
- https://gymnasium.farama.org/
- Lecture slides
- https://arxiv.org/abs/1511.05952
- https://arxiv.org/abs/1509.06461
- https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

Part III - Improving DQN & Solving Various Problems

1. Discuss the algorithm you implemented.

Dueling DQN is an improvement over the standard Deep Q-Network (DQN) algorithm that helps with more efficient learning, especially in environments where some actions do not significantly impact the reward. It achieves this by decomposing the Q-value into two separate streams:

- Value Stream: Estimates the overall value of being in a given state.
- Advantage Stream: Estimates the relative importance of each action in that state.

These two streams are combined to compute the final Q-value using:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|A|}\sum_{a'} A(s,a')\right)$$

This formulation ensures that the advantage values are centered around zero, preventing the overestimation of Q-values.

Key Features:

1. Neural Network Architecture:
   a. Two fully connected layers (256 neurons each).
   b. Separate Value and Advantage streams to calculate Q-values.
2. Replay Buffer:
   a. Stores past experiences and samples minibatches to break correlations in data.
   b. Allows for more stable and efficient learning.
3. Target Network:
   a. A separate network that updates periodically to stabilize Q-value updates.
   b. Helps prevent the issue of chasing a moving target in Q-learning.
4. Epsilon-Greedy Policy:
   a. Balances exploration vs. exploitation.
   b. Decays over time to encourage exploitation of learned strategies.

Advantages of Using Dueling DQN:

- Better Generalization: Helps in environments where the choice of action matters less in certain states.
- More Stable Learning: Reduces variance in Q-value estimates, leading to smoother training.
- Improved Sample Efficiency: Prioritizes learning the value of states more effectively, speeding up convergence.

Application to Grid World & CartPole:

- In Grid World (Drone Delivery), this helps the agent learn which states are more valuable (e.g., near delivery locations) without relying heavily on the effect of individual actions.
- In CartPole, it improves stability by separating the value of balancing from the effect of individual tilts.

2. What is the main improvement over the vanilla DQN?

The main improvement of Dueling DQN over Vanilla DQN is the separation of state-value and action-advantage functions, which allows the network to better estimate the value of each state independently of specific actions.

Key Improvements Over Vanilla DQN:

1. Better Value Estimation in Irrelevant Actions:
   a. In many states, the action taken has little impact on the future outcome (e.g., standing still in Grid World).

    b. Dueling DQN explicitly learns a separate state-value function V(s), helping the agent identify good states without relying on action selection.
2. More Stable Learning & Faster Convergence:
    a. By reducing variance in Q-value estimates, it leads to faster and more stable training compared to DQN.
    b. The advantage function ensures that only the necessary part of the Q-value function is updated, preventing overestimation of unnecessary actions.
3. Efficient Policy Learning:
    a. In Vanilla DQN, every Q-value for each action is learned independently, making learning slower.
    b. Dueling DQN efficiently shares information between similar actions through the advantage function, improving learning speed.

Mathematical Difference:

Vanilla DQN:

$$Q(s,a) = f(s,a)$$

Dueling DQN:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|A|}\sum_{a'} A(s,a')\right)$$

- V(s) learns how good it is to be in state s, regardless of action.
- A(s,a) learns the importance of each specific action in state s.

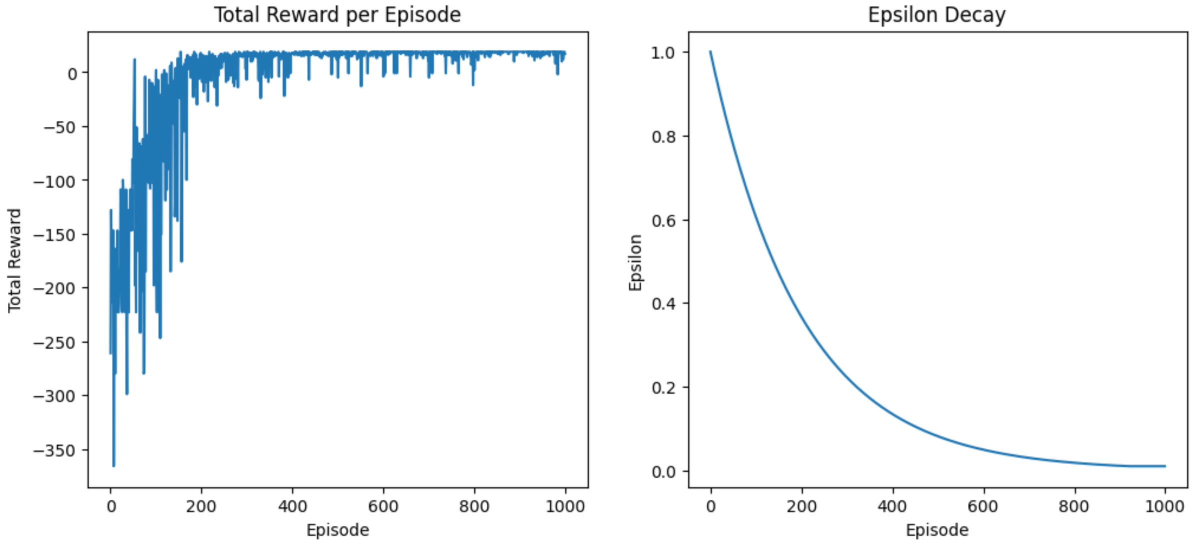
This separation makes the agent more efficient in learning which states are valuable before learning which actions are best.

Practical Benefits:

- Dueling DQN works best in environments where many actions lead to similar outcomes, such as Grid World or Atari games.
- In contrast, Vanilla DQN struggles when many actions don’t significantly change the state value.

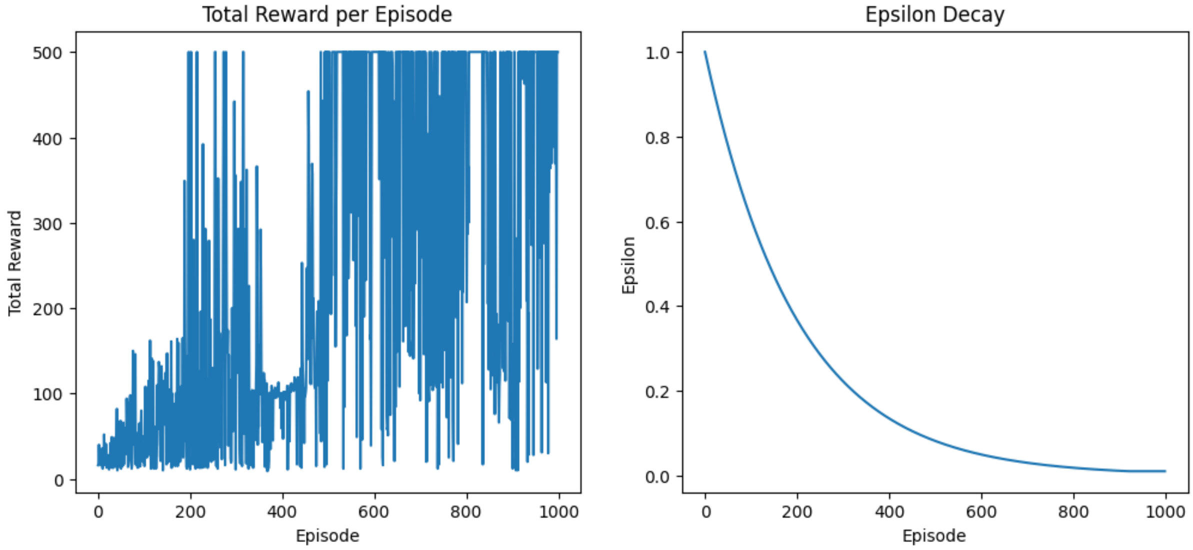
3. Show and discuss your results after applying your the two algorithms implementation on the environment. Plots should include epsilon decay and the total reward per episode.

a. Gridworld:



- Total Reward per Episode: The curve shows a sharp improvement in early episodes with fluctuations stabilizing over time. The agent successfully learns to navigate the grid and optimize its reward.
- Epsilon Decay: The agent starts with high exploration, then gradually shifts towards exploitation, leading to improved performance.
- Observations: The training seems effective but occasional drops in reward indicate that the agent still explores suboptimal paths at times.

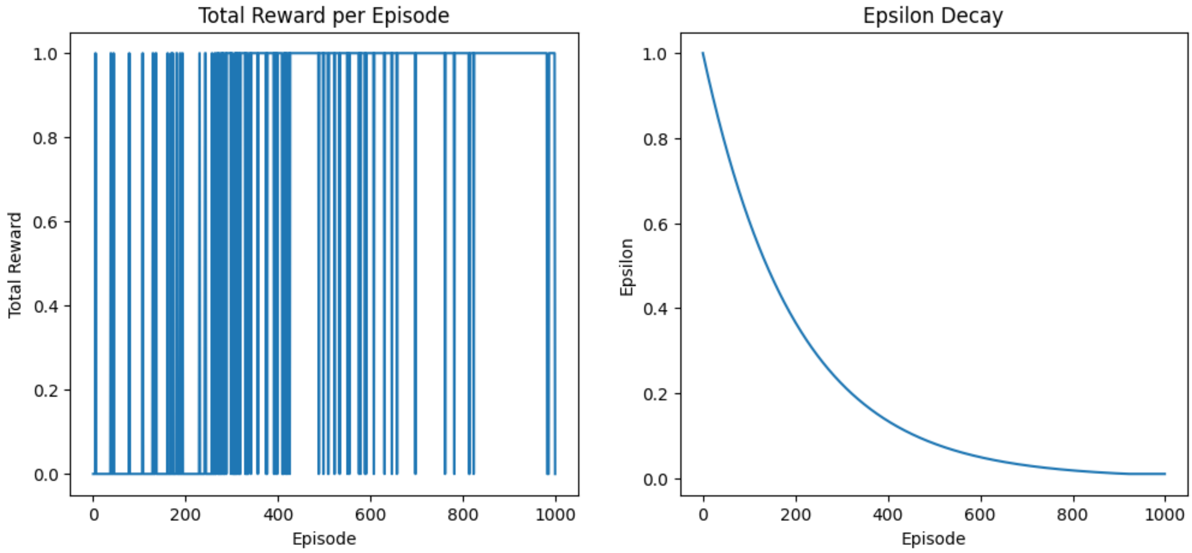
b. Cartpole-v1:



- Total Reward per Episode: The plot shows an increase in rewards, with significant fluctuations, especially after episode 400. This suggests the agent is learning but still making mistakes.
- Epsilon Decay: The agent reduces exploration over time but the instability in rewards suggests that it occasionally deviates from the optimal policy.

- Observations: The variance in performance indicates that tuning hyperparameters such as learning rate and target network update frequency could improve stability.

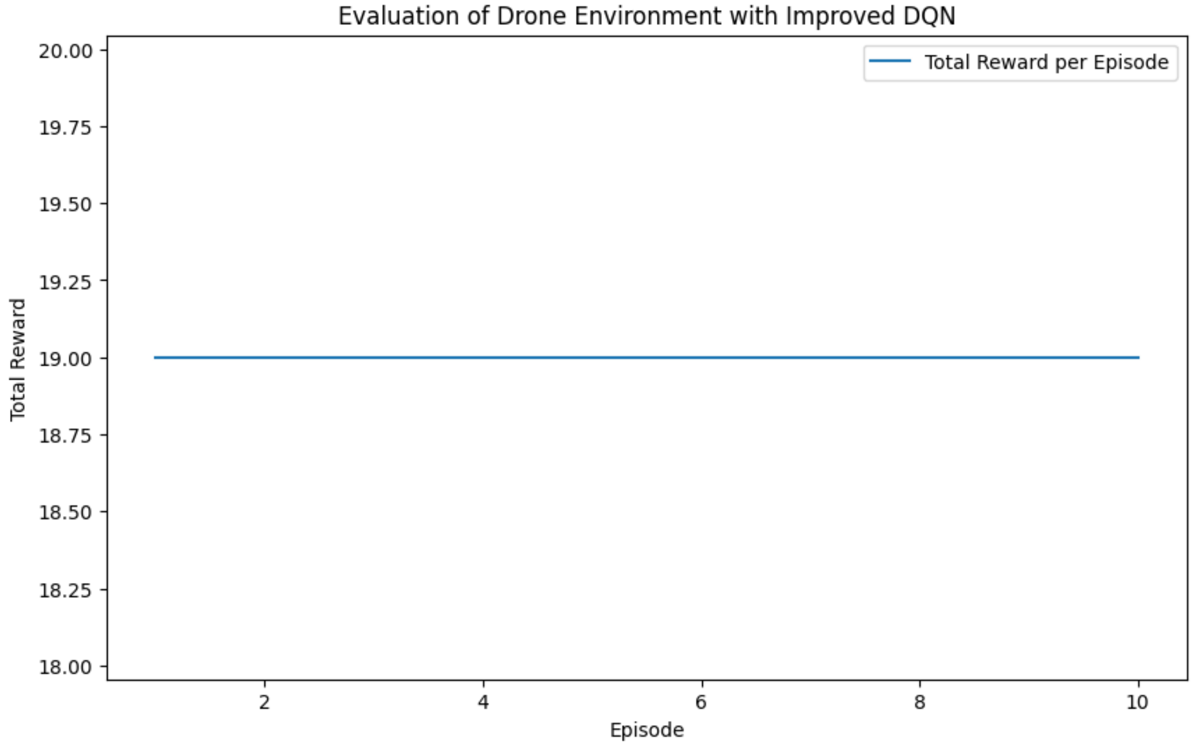
c. FrozenLake:



- Total Reward per Episode: The reward graph shows a binary pattern, episodes either achieve success (reward = 1) or fail (reward = 0). This is expected in a deterministic FrozenLake setup, where an optimal policy exists and consistently reaches the goal once learned. The learning curve suggests that the agent initially struggles but, after around episode 200–300, it begins finding the correct path more frequently.
- Epsilon Decay: Epsilon decays gradually, reducing exploration over time. Once epsilon is low, the agent mostly exploits the best-known path instead of randomly testing alternatives. The success rate stabilizes, as seen in the later episodes where the agent mostly succeeds.
- Observations: A possible improvement would be to use Prioritized Experience Replay (PER) or incorporate reward shaping to help the agent differentiate good policies.

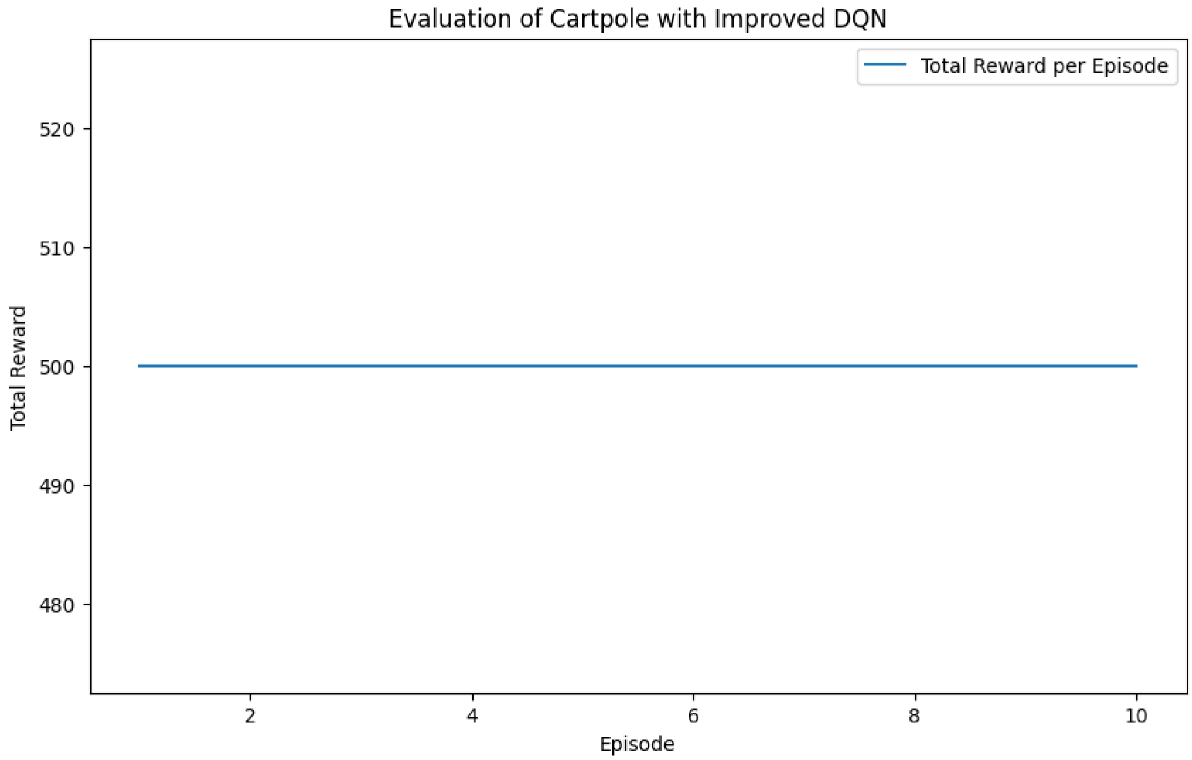
4. Provide the evaluation results. Run your environment for at least 5 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.

a. Gridworld:



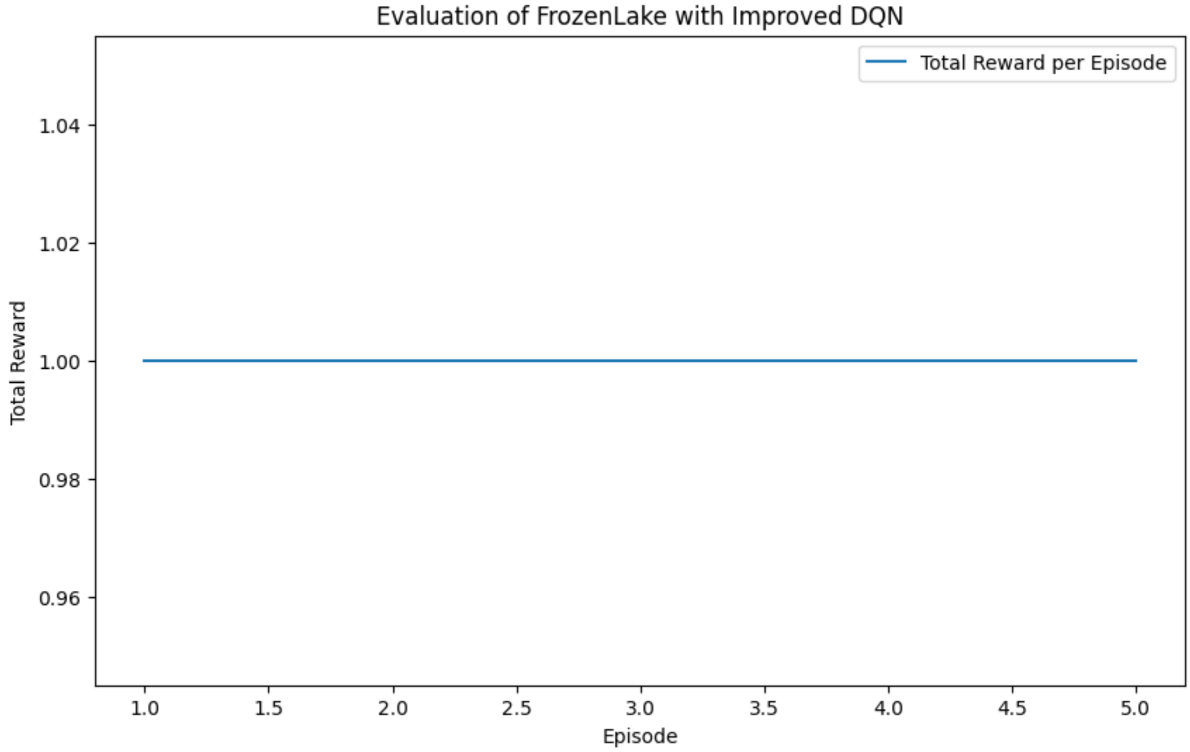
In the Drone environment, the improved DQN agent achieves a total reward of 19 in every evaluation episode, showing strong consistency and stability. This result suggests that the agent has effectively learned a deterministic and repeatable strategy that performs well across episodes. While the uniformity of rewards is encouraging pointing toward convergence to a good policy, it also raises the question of whether the environment might be too simple or predictable. Another possibility is that the agent always takes the same optimal path with no variability in outcomes. Either way, the behaviour looks solid and its a clear improvement over a trained agent. Still, it might be worth adding a bit of noise or running with different seeds to confirm that the policy is robust and not just overfitting to a specific scenario.

b. Cartpole-v1:



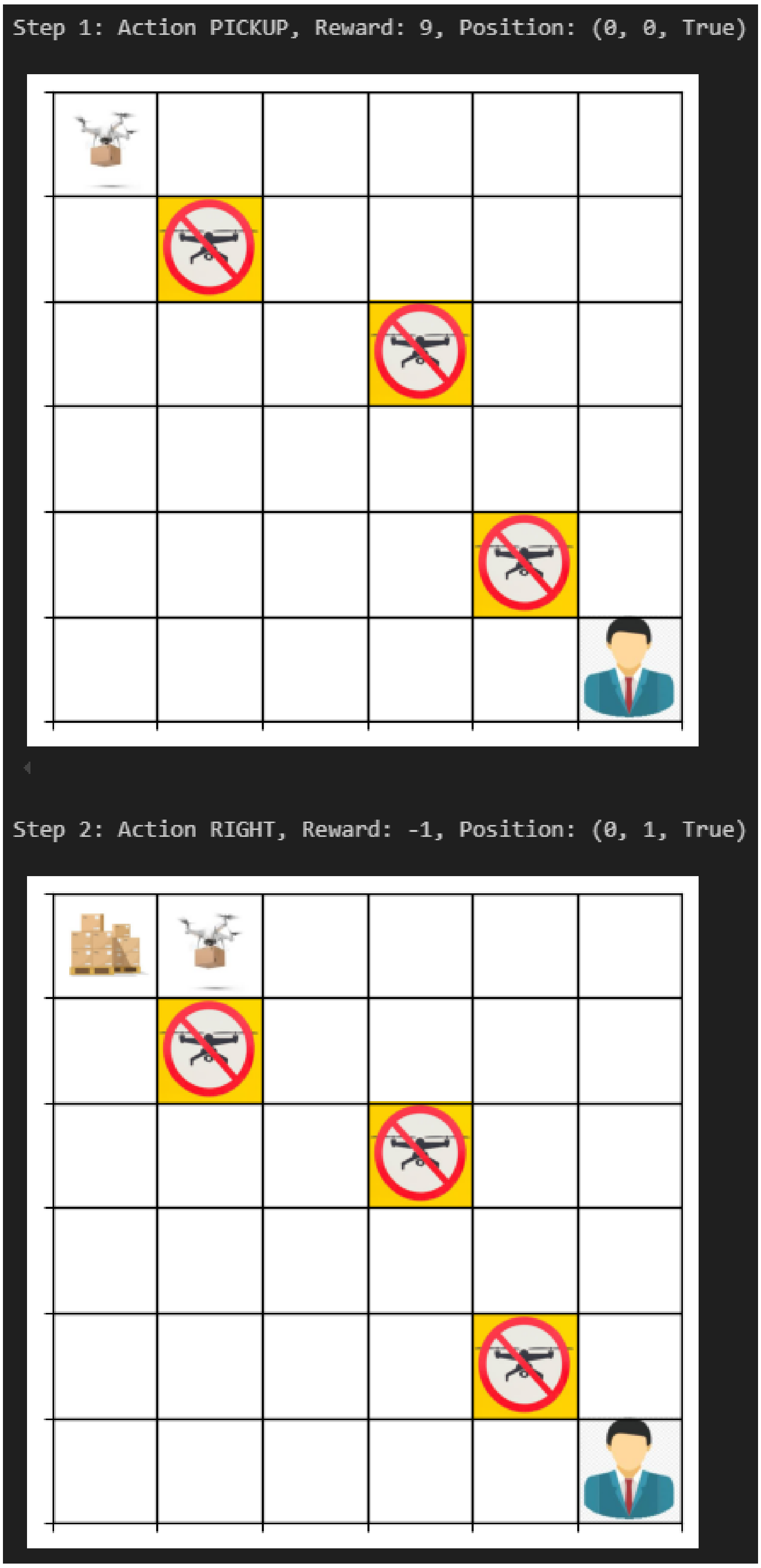
The results in the CartPole environment are particularly impressive, the agent scores the maximum possible reward of 500 in every episode evaluated. This indicates that the DQN agent has not only learned a successful policy but one that is essentially perfect, balancing the pole for the full duration in all test runs. CartPole is often considered a benchmark for verifying if a reinforcement learning algorithm is working properly and achieving this level of performance shows that the model architecture, training dynamics and hyperparameters are all well-tuned. The lack of reward variability also implies that the agent is reliably avoiding terminal states and recovering from potential imbalances. While this may look too good to be true, the nature of CartPole makes such plateauing reasonable once the policy is refined enough. Nonetheless, extended testing over more episodes would help ensure it’s not a fluke due to deterministic physics or static starting conditions.

c. FrozenLake:



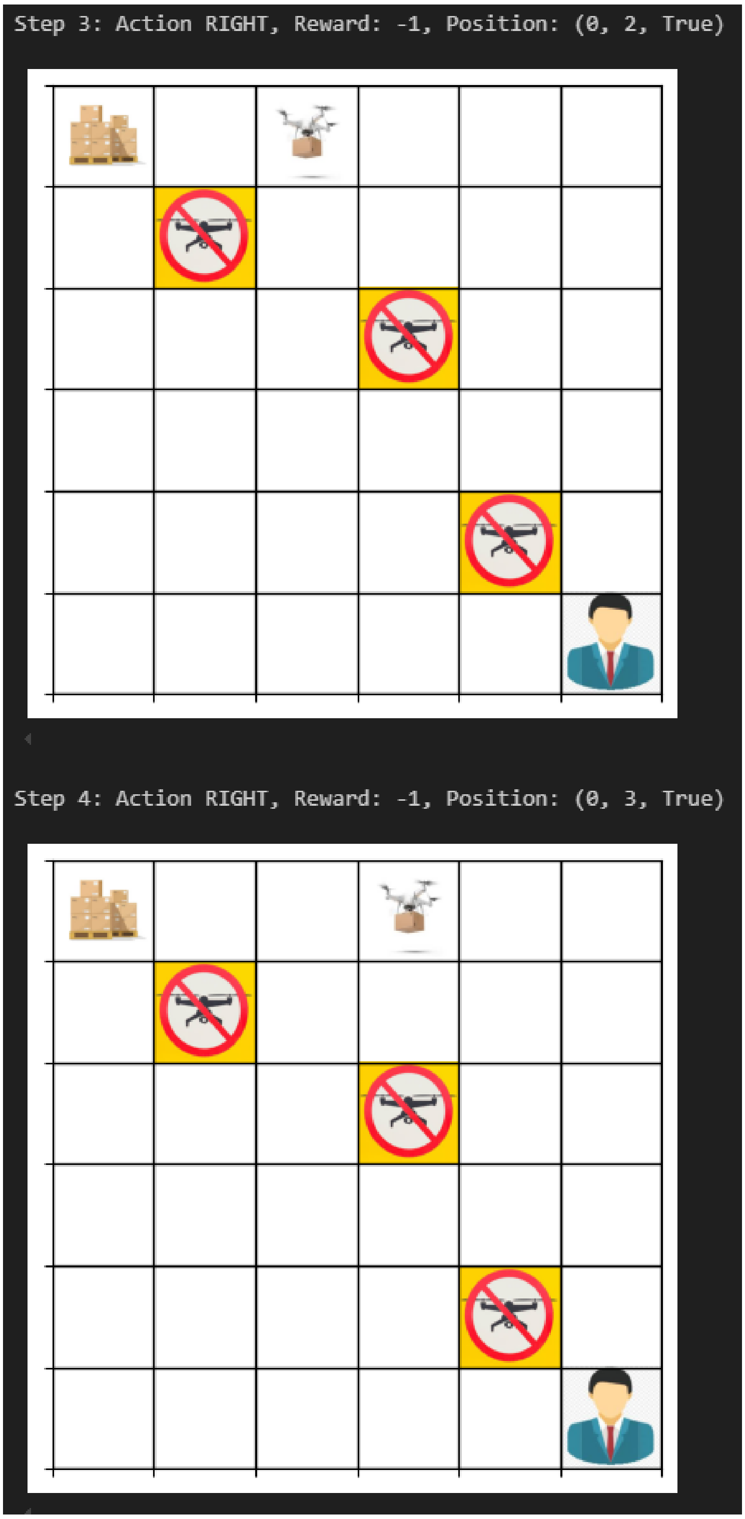
In the FrozenLake environment, the agent achieves a reward of 1.0 in every one of the five test episodes which implies that it reaches the goal state successfully every time. FrozenLake is notoriously tricky because it offers sparse rewards and punishes missteps harshly, slipping into holes or moving away from the goal. For a DQN agent to navigate this reliably suggests a high-quality learned value function and effective exploitation of learned Q-values in greedy mode. However, due to the small number of test episodes, it's hard to assess whether the agent's policy generalizes well across various random starts or map layouts. Running evaluations over more episodes and including randomized or custom maps could help verify the robustness of the solution. Still, hitting the goal every time under default conditions is a strong sign of intelligent, learned behaviour.

5. Grid World environments only: Run your environment for 1 episode where the agent chooses only greedy actions from the learned policy. Render each step of the episode and verify that the agent has completed all the required steps to solve the environment. Save this render and include it in your report as clearly ordered screenshots or as a clearly named video file in your submission.



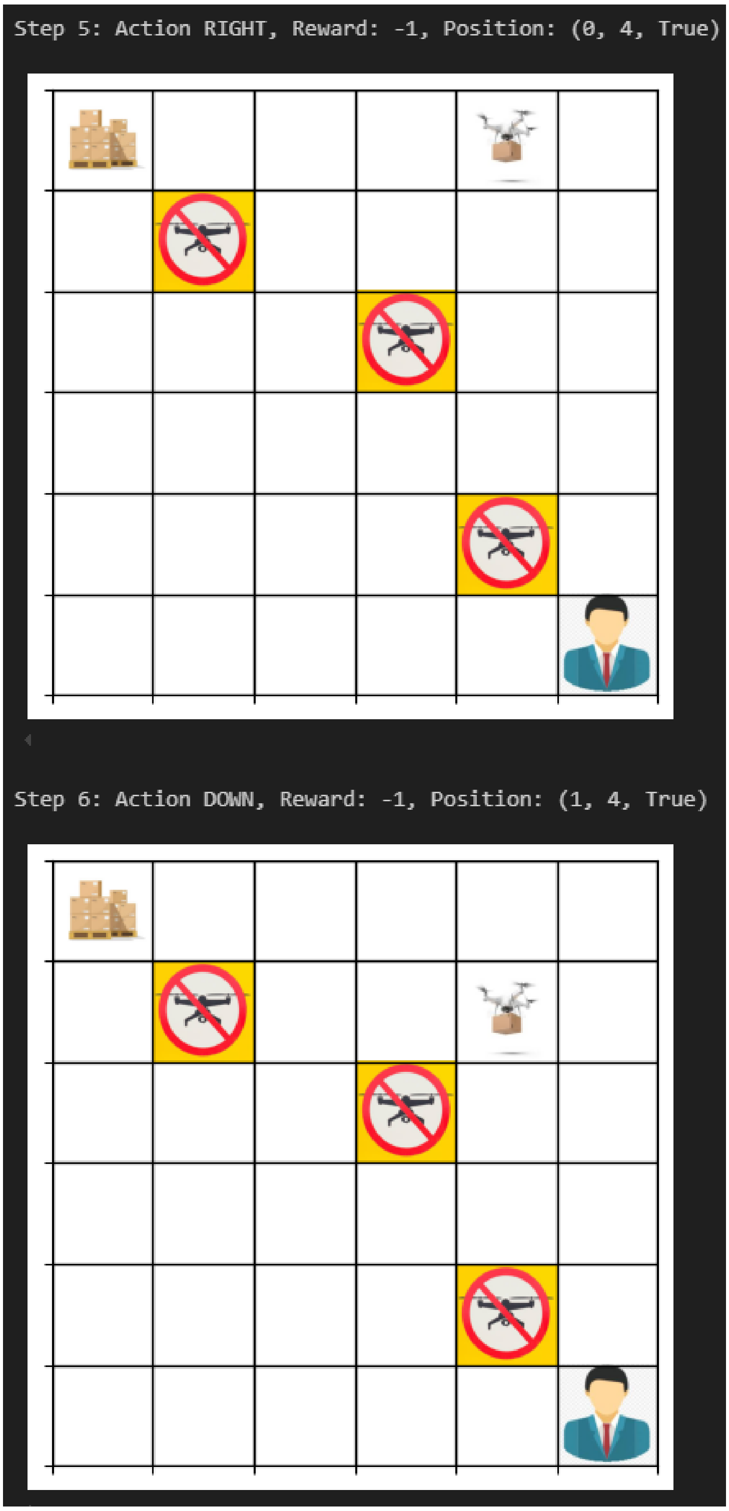
```
Step 1: Action PICKUP, Reward: 9, Position: (0, 0, True)
```

```
Step 2: Action RIGHT, Reward: -1, Position: (0, 1, True)
```



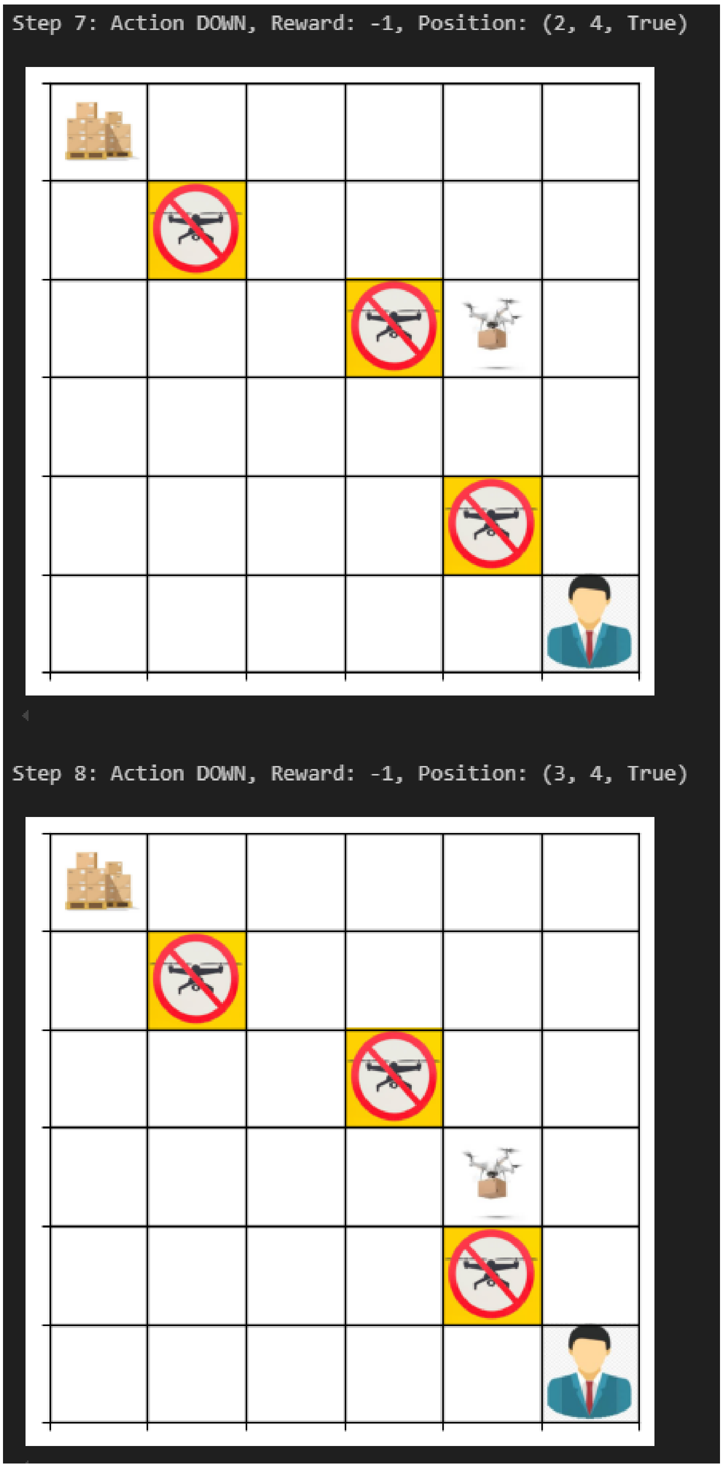
```
Step 3: Action RIGHT, Reward: -1, Position: (0, 2, True)
```

```
Step 4: Action RIGHT, Reward: -1, Position: (0, 3, True)
```



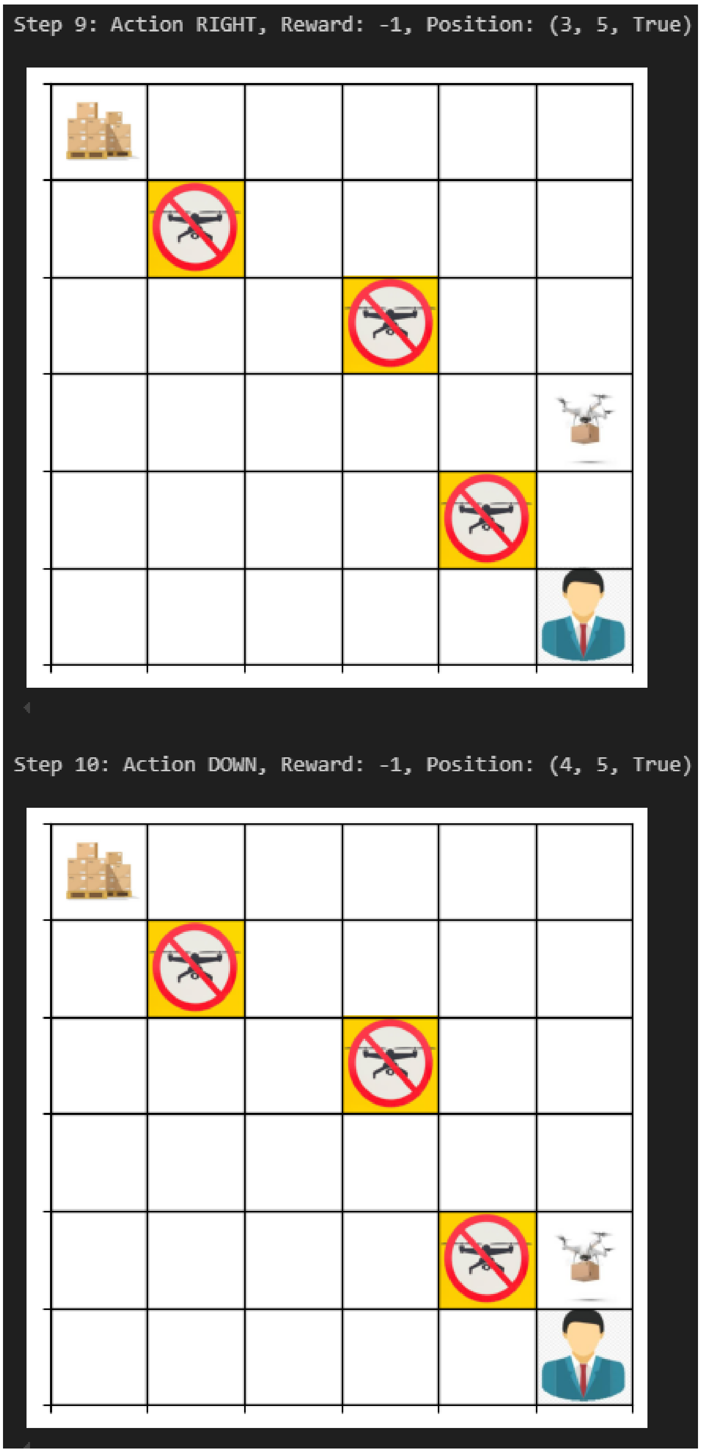
```
Step 5: Action RIGHT, Reward: -1, Position: (0, 4, True)
```

```
Step 6: Action DOWN, Reward: -1, Position: (1, 4, True)
```



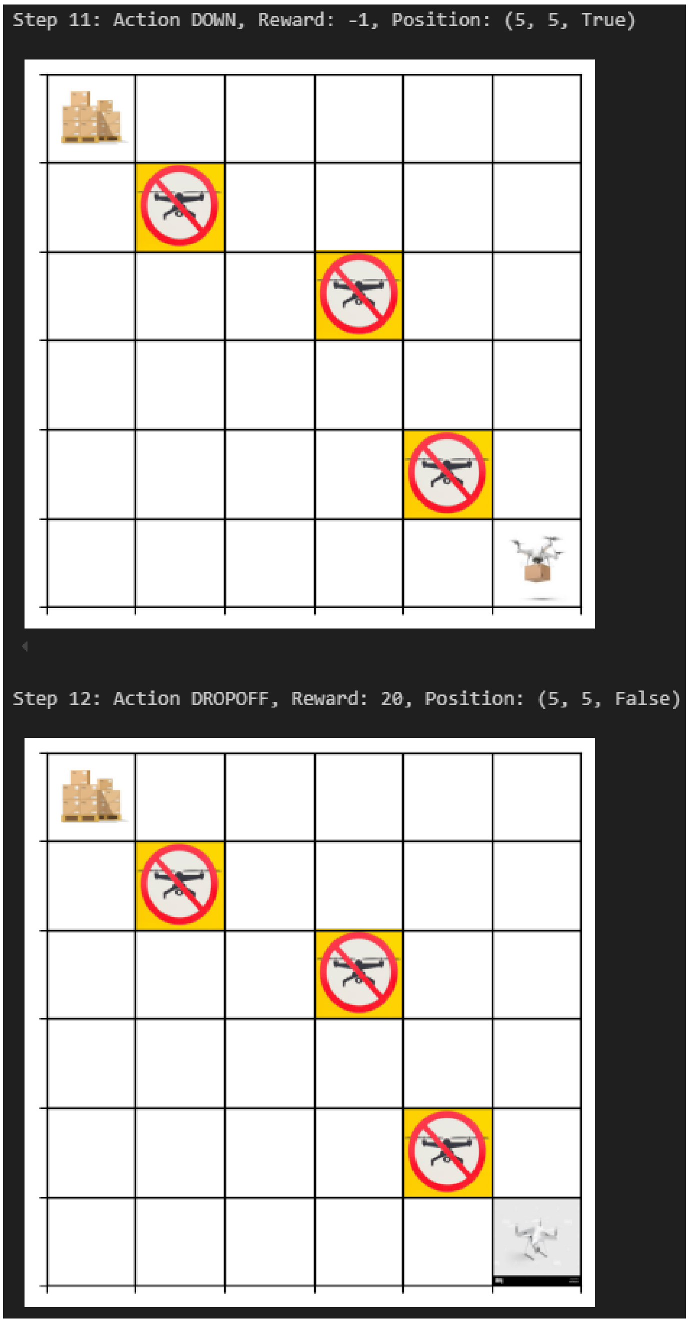
```
Step 7: Action DOWN, Reward: -1, Position: (2, 4, True)
```

```
Step 8: Action DOWN, Reward: -1, Position: (3, 4, True)
```



```
Step 9: Action RIGHT, Reward: -1, Position: (3, 5, True)
```

```
Step 10: Action DOWN, Reward: -1, Position: (4, 5, True)
```

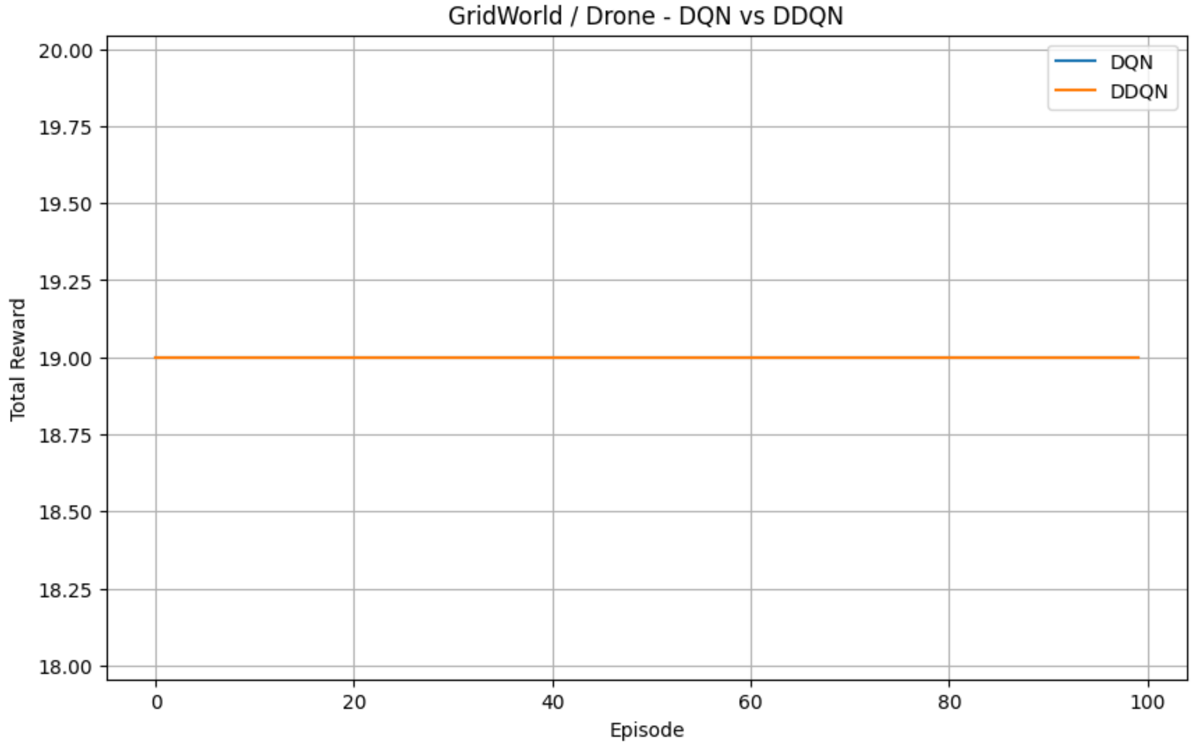


```
Step 11: Action DOWN, Reward: -1, Position: (5, 5, True)
```

```
Step 12: Action DROPOFF, Reward: 20, Position: (5, 5, False)
```

6. Compare the performance of both algorithms (DQN & Improved version of DQN) on the same environments (e.g. show one graph with two reward dynamics) and provide your interpretation of the results. Overall three rewards dynamics plots with results from two algorithms applied on:

- Grid-world environment:



DQN:

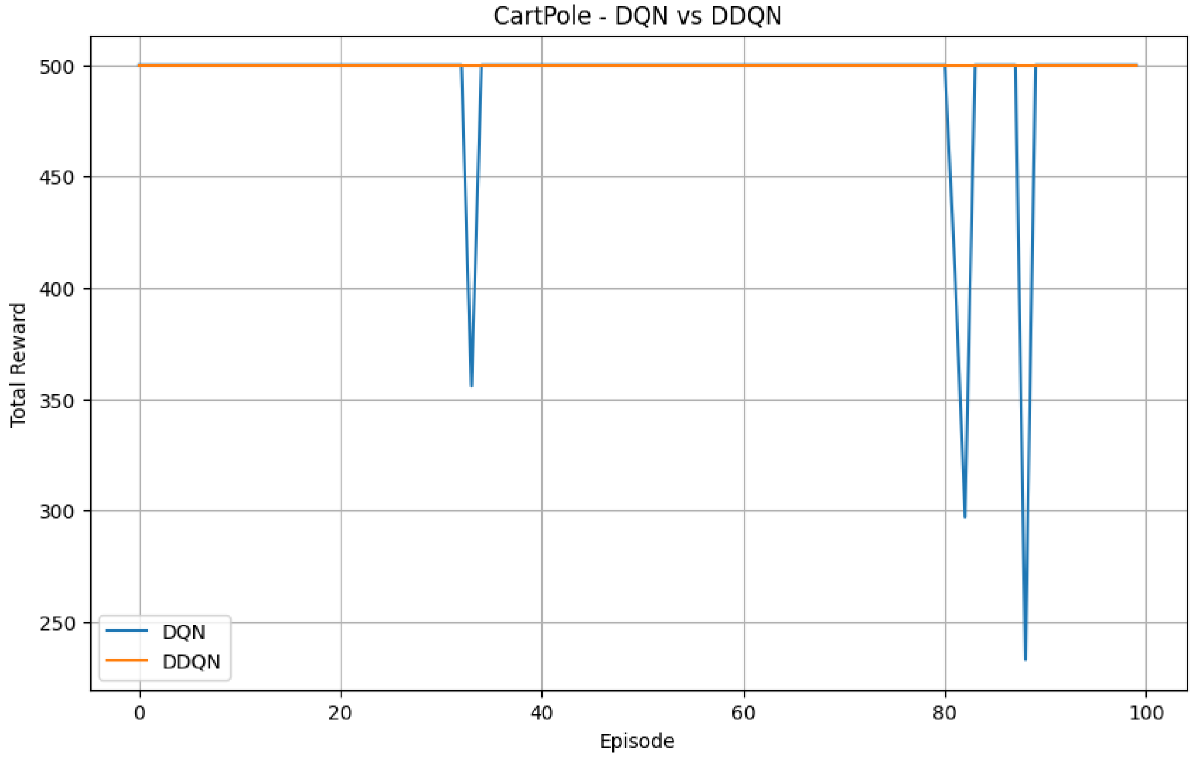
- Achieves a total reward of 19 consistently (overlapping with DDQN).
- Learns the optimal policy but does not show a visible difference from DDQN.

DDQN:

- Also achieves a total reward of 19 across all episodes.
- Likely reduces Q-value overestimation, but in this simple environment, it does not provide a major advantage.

Conclusion: Both algorithms perform equally well.

- CartPole-v1:



DQN:

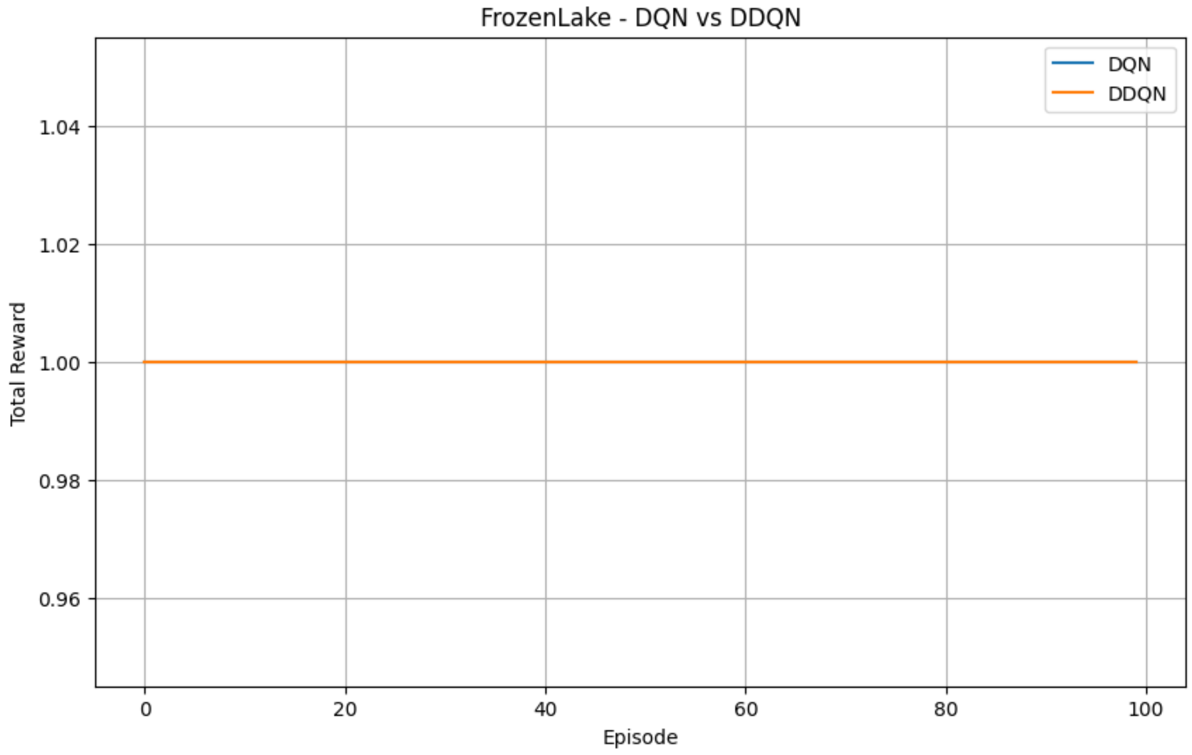
- Reaches near-optimal performance (500 reward).
- However, sharp drops in performance occur at episodes ~35, 81, and 90, indicating instability.

DDQN:

- Maintains a perfect reward (500) consistently across all episodes.
- Prevents sudden performance drops, showing better stability.

Conclusion: DDQN is the winner due to its stability and robustness.

• FrozenLake:



DQN:

- Achieves a total reward of 1.0 consistently after learning the optimal policy.
- Performs as expected in a deterministic and low-variance environment.

DDQN:

- Also achieves a total reward of 1.0, indicating no significant difference in this task.

Conclusion: Both DQN and DDQN perform identically in this environment.

7. Provide your interpretation of the results. E.g., how the same algorithm behaves on different environments, or how various algorithms behave on the same environment.

DQN Across Environments:

- Performs perfectly in FrozenLake (simple and deterministic).
- Unstable in CartPole, with sudden drops in performance, suggesting overestimation bias affects learning.
- In GridWorld, it performs well, but not distinguishable from DDQN (likely due to environment simplicity).

Insight: DQN struggles with environments that require precise value estimation (CartPole) but performs well in deterministic settings (FrozenLake).

DDQN Across Environments:

- Consistently stable and optimal in all environments.
- Prevents sudden performance drops in CartPole, which is crucial for control problems.
- Performs similarly to DQN in environments with low variance (GridWorld, FrozenLake).

Insight: DDQN is more stable and robust across different environments, particularly in those where Q-value overestimation is problematic.

GridWorld / Drone Environment:

- Both DQN and DDQN perform identically, reaching the optimal policy.
- Since GridWorld is less susceptible to Q-value overestimation, DDQN does not provide a noticeable advantage.

Conclusion: DDQN is unnecessary for this simple environment, as DQN performs just as well.

CartPole-v1:

- DQN reaches high rewards but suffers from instability, occasionally dropping performance.
- DDQN consistently maintains the maximum reward, avoiding performance fluctuations.

Conclusion: DDQN is clearly superior in environments that require precise Q-value estimation.

FrozenLake:

- Both DQN and DDQN reach the perfect policy, performing identically.
- The environment’s low variance and deterministic nature mean overestimation bias is not an issue.

Conclusion: DDQN does not provide additional benefits in simple, deterministic environments.

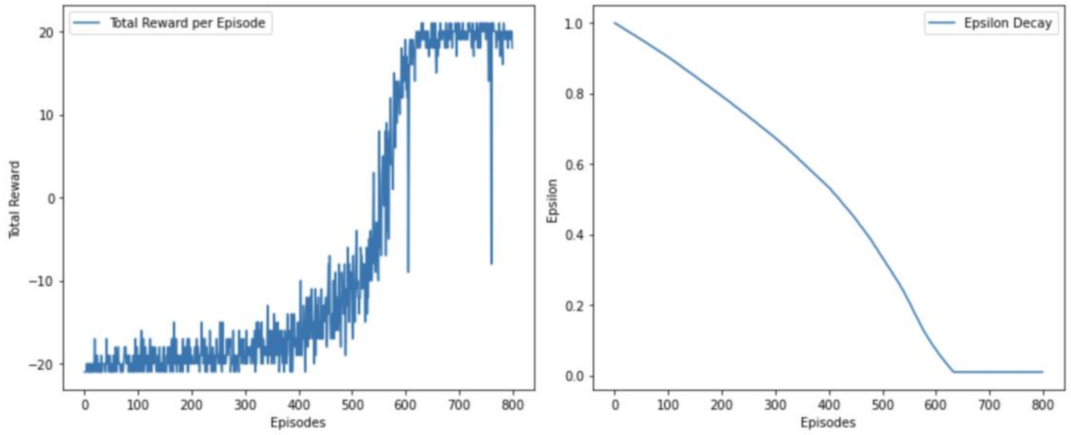
8. Include all the references that have been used to complete this part

- https://medium.com/@sainijagjit/understanding-dueling-dqn-a-deep-dive-into-reinforcement-learning-575f6fe4328c
- https://gymnasium.farama.org/environments/classic_control/cart_pole/
- https://gymnasium.farama.org/environments/toy_text/frozen_lake/
- CSE546 RL Assignment 1 submitted by ruthvikv
- https://gymnasium.farama.org/
- https://ale.farama.org/multi-agent-environments/pong/
- Lecture slides

Bonus Part - Solving Image-based Environment

Atari Pong (‘PongNoFrameskip-v4’) was chosen to solve.

Using Vanilla DQN



Inference

We can observe the Pong environment took around 600 episodes or 1,060,742 time steps to reach convergence i.e. a mean reward of 15 over past 100 episodes.

We can also observe something similar in the epislon decay plot as well, we see as the training progresses and the number of episodes increases, the epsilon value gradually decreases. This signifies a shift from exploration towards exploitation. The agent becomes more confident in its learned Q-values/policy and increasingly relies on its policy.

Results -

Training

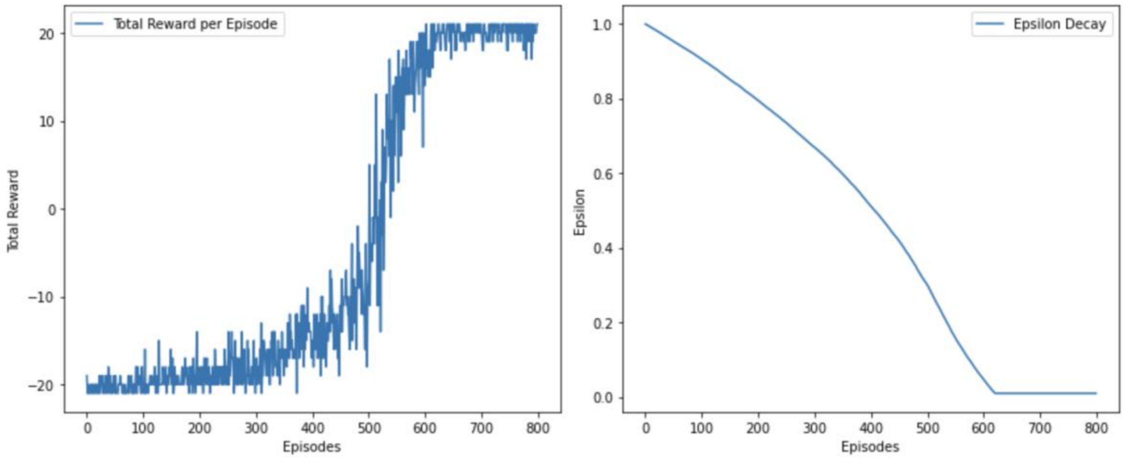
Training time - 6773.53 seconds

Total Episodes: 800 | Total Steps: 1298852

Evaluation-

Average Reward over 10 episodes: 20.70

Using Dueling DQN



Inference:

We can observe the Pong environment took around 631 episodes or 1,017,424 time steps to reach convergence i.e. a mean reward of 15 over past 100 episodes.

We can also observe something similar in the epislon decay plot as well, we see as the training progresses and the number of episodes increases, the epsilon value gradually decreases. This signifies a shift from exploration towards exploitation. The agent becomes more confident in its learned Q-values/policy and increasingly relies on its policy.

Results -

Training

Training time - 7117.92 seconds

Total Episodes: 800 | Total Steps: 1311077

Evaluation-

Average Reward over 10 episodes: 21.00

Bonus Part - RL Libraries

Cartpole using DQN in Stable-baselines3

The following were the hyperparameters used for DQN in Stablebaseline3
1. learning_rate: 0.0002 - Step size for weight updates.

2. buffer_size: 1000 - Size of the experience replay buffer.

3. learning_starts: 1000 - Timesteps before training starts.

4. batch_size: 128 - Number of experiences per training step.

5. gamma: 0.99 - Discount factor for future rewards.

6. train_freq: 256 - Timesteps between each training step.

7. gradient_steps: 128 - Number of updates per training step.

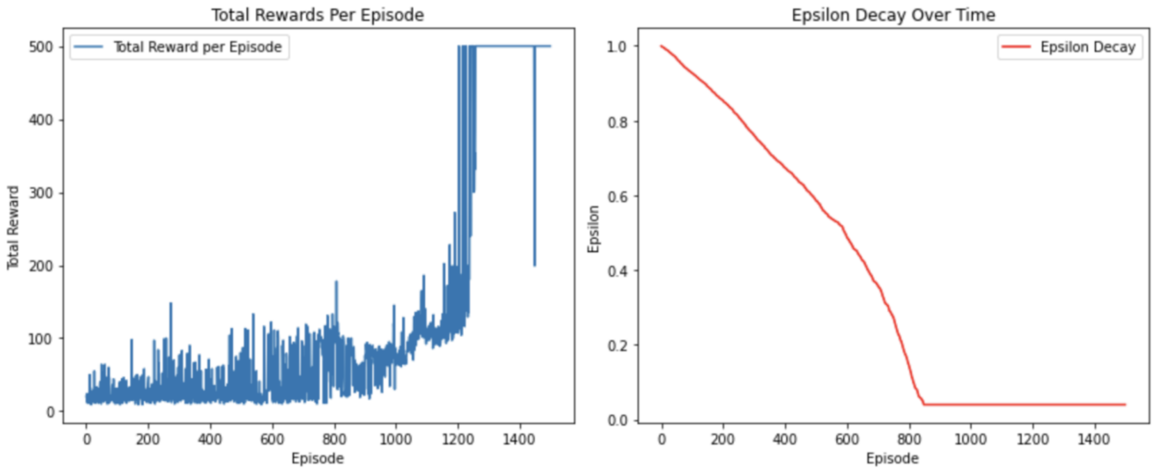
8. target_update_interval: 10 - Timesteps between target network updates.

9. exploration_fraction: 0.16 - Fraction of training for exploration.

10. exploration_final_eps: 0.04 - Final exploration rate.

11. policy_kwargs: net_arch=[256, 256, 128] - Neural network architecture.

Agent was trained for – 200000 time steps



Inference:

Training Time: 442.21 seconds

Average Reward from Evaluation: 500.00

Drone Environment using DQN in Stable-baselines3

Hyperparameters for the model used:

1. learning_rate: 1e-3 - Step size for weight updates.

2. buffer_size: 10,000 - Size of the experience replay buffer.

3. batch_size: 64 - Number of experiences per training step.

4. learning_starts: 1000 - Timesteps before training starts.

5. gamma: 0.99 - Discount factor for future rewards.

6. train_freq: 4 - Timesteps between each training step.

7. target_update_interval: 1000 - Timesteps between target network updates.

8. exploration_fraction: 0.1 - Fraction of training for exploration.

9. exploration_final_eps: 0.05 - Final exploration rate.

Agent was trained for – 100000 timesteps

Mean Reward on evaluation (10 episodes) – 19.0

Training time – 114 seconds

Frozen Lake using DQN in Stable-baselines3

The following were the hyperparameters used for DQN in Stablebaseline3:
1. learning_rate: 0.0002 - Step size for weight updates.

2. buffer_size: 1000 - Size of the experience replay buffer.

3. learning_starts: 1000 - Timesteps before training starts.

4. batch_size: 128 - Number of experiences per training step.

5. gamma: 0.99 - Discount factor for future rewards.

6. train_freq: 256 - Timesteps between each training step.

7. gradient_steps: 128 - Number of updates per training step.

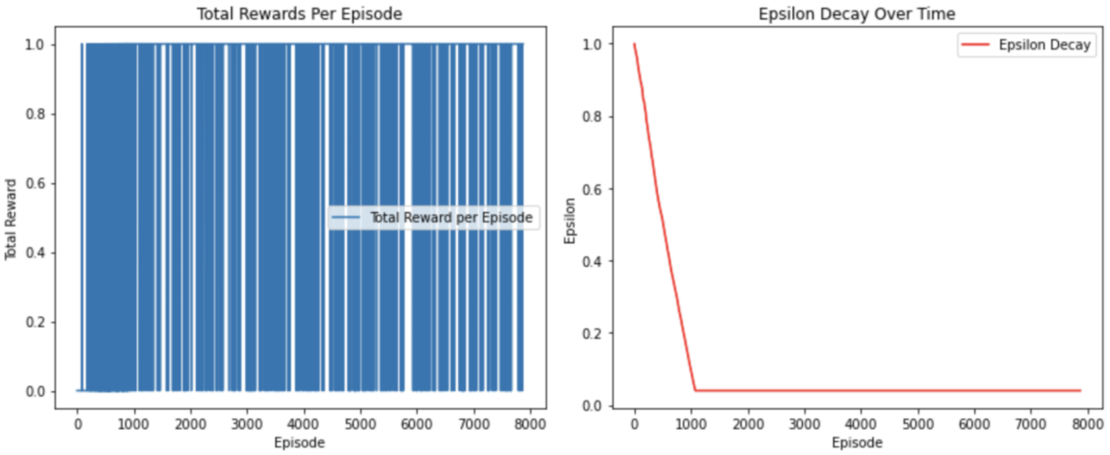
8. target_update_interval: 10 - Timesteps between target network updates.

9. exploration_fraction: 0.16 - Fraction of training for exploration.

10. exploration_final_eps: 0.04 - Final exploration rate.

11. policy_kwargs: net_arch=[256, 256, 128] - Neural network architecture.

Agent was trained for – 50000 time steps



Inference

Training Time: 114.52 seconds

Average Reward from Evaluation: 1.00

References:

1. https://ale.farama.org/multi-agent-environments/pong/
2. https://medium.com/nerd-for-tech/reinforcement-learning-deep-q-learning-with-atari-games-63f5242440b1

Contribution Table:

Team Member	Assignment Part	Contribution (%)
Ruthvik Vasantha Kumar	1, 2, 3, bonus	50%
Shaurya Mathur	1, 2, 3, bonus	50%