

CSE 4/546: Reinforcement Learning

Spring 2025

Instructor: Alina Vereshchaka

Final Course Project - Checkpoint report

TEAM 25

Title: Multi-Agent Reinforcement Learning for Drone Delivery Coordination and Simple Spread (v3) from PettingZoo

Short Summary:

This project investigates Multi-Agent Reinforcement Learning (MARL) in two distinct multi-agent transportation environments:

1. A custom-designed autonomous drone delivery system, where drones must cooperate to deliver packages across a city grid while avoiding no-fly zones.
2. The Simple Spread v3 environment from PettingZoo, a benchmark cooperative MARL scenario.

Both environments pose complex multi-agent coordination challenges, requiring agents to learn cooperative, competitive, and communication strategies to achieve optimal task completion. These scenarios reflect real-world applications such as urban drone delivery networks and autonomous vehicle coordination, offering practical test beds for evaluating MARL algorithms.

Objectives:

- Develop a multi-agent environment with at least two agents (expanding from Assignment 1).
- Solve the environment using tabular reinforcement learning methods (e.g., Q-Learning, SARSA, Double Q-Learning).
- Implement deep reinforcement learning algorithms (e.g., DQN, QMIX) for multi-agent learning and compare their performance with tabular methods.
- Apply the implemented algorithms to Simple Spread v3 from PettingZoo and evaluate their effectiveness.

NOTE: We changed the second environment from the highway merging scenario to Simple Spread v3 (PettingZoo).

DEFINING THE ENVIRONMENT

Custom MARL environment - DroneDeliveryMARLEnv

DroneDeliveryMultiAgentEnv is a custom multi-agent environment simulating a drone delivery system on a 6x6 grid world. The environment is designed for centralized training with decentralized execution (CTDE), where multiple drones must pick up packages from specified locations and deliver them to designated drop-off points while avoiding static obstacles, no-fly zones, collisions, and border violations.

This environment provides a challenging multi-agent decision-making problem combining navigation, task allocation, collision avoidance, and safety constraints.

Environment Details

Attribute	Value
Observation Space	Tuple(Discrete(6), Discrete(6), Discrete(2)) per agent
Action Space	Discrete(6) per agent
Number of Agents	2 (default)
Grid Size	6 x 6
Max Steps	100
Packages	2
No-Fly Zones	7 static coordinates

Observation Space

Each agent's observation is a tuple of 3 discrete values:

(x, y, carrying)

- x: Agent's x-coordinate (0 to 5)
- y: Agent's y-coordinate (0 to 5)
- carrying: Binary indicator (0: not carrying, 1: carrying a package)

Agents observe their own position and carrying status; no direct observation of other agents or environment state. A centralized state (global info) can be obtained via `get_centralized_state()` for centralized training.

Reward Structure

Rewards are provided per agent but calculated as a shared cumulative reward per timestep:

Event	Reward
Successful pickup	+25
Successful delivery	+100
Step penalty	-1
Entering no-fly zone	-100
Collision with other agent	-10
Invalid pickup/drop	-25
Hitting grid border	-50

Agents receive the same shared reward for any agent's action to encourage cooperative behavior.

Termination Conditions

An episode terminates when either:

- All packages are successfully delivered, or
- Maximum number of steps (100) is reached.

Each agent's termination flag indicates whether it completed delivery or was truncated by time.

Safety & Hazards Modeled

The environment includes safety-critical elements simulating real-world drone constraints:

- No-fly zones (static hazard zones)
- Collision avoidance (agents penalized for moving into same cell)
- Border restrictions (penalty for moving outside grid)
- Dynamic shared obstacles (violated no-fly zones are remembered across steps)
- Invalid operation penalties (attempting to pick up/drop off at incorrect locations)

This makes it suitable for studying safe reinforcement learning, multi-agent coordination under safety constraints, and learning cooperative behaviors under sparse reward signals.

Key Objectives for Agents

Agents must:

- Plan paths to packages avoiding no-fly zones
- Pick up and deliver their assigned (or available) packages
- Avoid collisions with other agents
- Minimize penalties while maximizing reward
- Coordinate implicitly or explicitly to achieve successful delivery under shared reward

Environment visualization

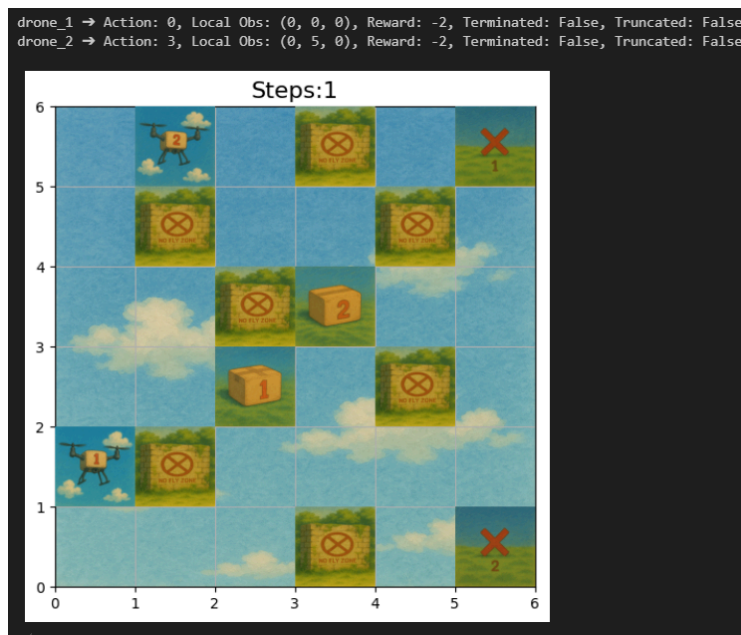


Fig: Environment visualization

Applying Tabular Methods

Q-Learning Implementation

We implemented a multi-agent Q-learning algorithm to solve the custom environment. Each drone (agent) maintains its own Q-table with state-action mappings based on its local observation, represented as a tuple of (x, y, carrying) coordinates and status. The action space consists of 6 discrete actions (move up, down, left, right, pick up, deliver).

The Q-learning agents follow an epsilon-greedy policy with a decaying epsilon to balance exploration and exploitation. The Q-table is updated using the standard Q-learning update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

The learning rate (α), discount factor (γ), and epsilon decay parameters were set to promote stable learning while encouraging early exploration.

Training Setup

The agents were trained for 500 episodes with the following hyperparameters:

- Learning rate (α): 0.1
- Discount factor (γ): 0.99
- Initial epsilon: 1.0
- Minimum epsilon: 0.01
- Epsilon decay factor: 0.995
- Maximum time steps per episode: 1000

During training, the agents gradually reduced exploration as epsilon decayed, leading to more greedy behavior over time. The environment provides dense feedback through shaped rewards for pickups, deliveries, collisions, illegal actions, and no-fly zone penalties.

Results and Evaluation

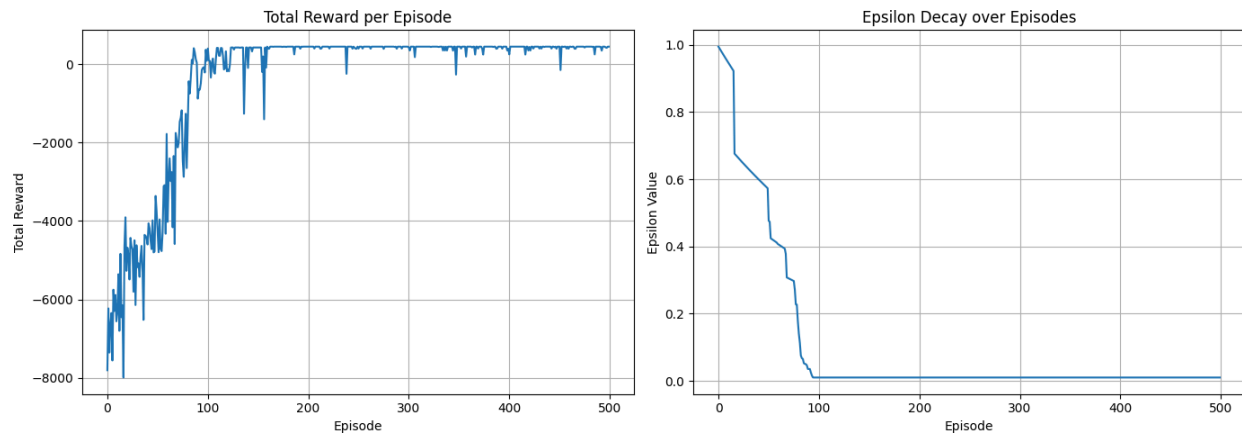
The training performance is summarized in the following figures:

1. Total Reward per Episode:

The agent starts with poor performance, accumulating rewards around -8000 per episode. As training progresses, rewards increase steadily, surpassing zero around episode 90 and stabilizing close to +450 after episode 100. This indicates successful learning of effective policies to deliver packages while avoiding penalties.

2. Epsilon Decay over Episodes:

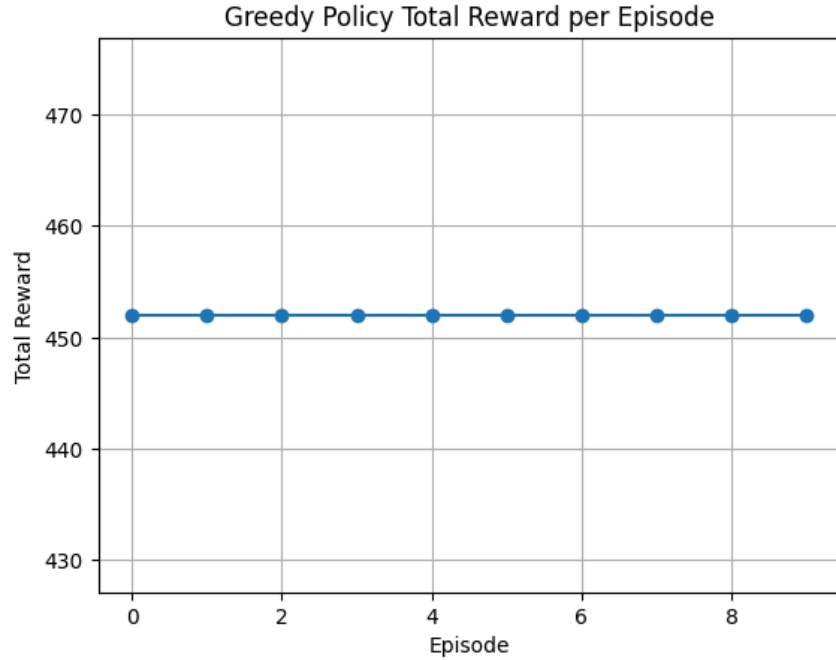
Epsilon decays sharply during the first 100 episodes, reaching the minimum epsilon value (0.01) around episode 100. The rapid decrease allowed for early exploration and faster convergence to exploitation.



During evaluation, the agents were tested under a greedy policy ($\epsilon=0$). The results across 10 evaluation episodes are:

```
Greedy Evaluation Episode 1: Total Reward = 452
Greedy Evaluation Episode 2: Total Reward = 452
Greedy Evaluation Episode 3: Total Reward = 452
Greedy Evaluation Episode 4: Total Reward = 452
Greedy Evaluation Episode 5: Total Reward = 452
Greedy Evaluation Episode 6: Total Reward = 452
Greedy Evaluation Episode 7: Total Reward = 452
Greedy Evaluation Episode 8: Total Reward = 452
Greedy Evaluation Episode 9: Total Reward = 452
Greedy Evaluation Episode 10: Total Reward = 452
```

This shows that the learned policy consistently achieves the maximum possible reward of 452 across episodes, indicating high reliability and successful task completion.



The results demonstrate that the Q-learning agents were able to learn an optimal or near-optimal policy in the DroneDeliveryMultiAgentEnv within 500 episodes. The environment's shaped reward function provided sufficient feedback to guide learning despite the large negative penalties for collisions and illegal moves.

The early convergence, reflected by the plateau in the reward plot and the low variance during greedy evaluation, suggests that the environment dynamics and the epsilon decay schedule supported efficient policy discovery.

SARSA Implementation

In addition to Q-learning, We implemented a multi-agent SARSA (State-Action-Reward-State-Action) algorithm in the same DroneDeliveryMultiAgentEnv. Each drone maintains an independent Q-table that is updated following the on-policy SARSA update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma Q(s',a') - Q(s,a)]$$

Unlike Q-learning, SARSA updates its estimate using the action actually taken in the next state, enabling a more conservative learning approach. Similar to Q-learning, agents use an epsilon-greedy policy with decaying epsilon for exploration-exploitation balance.

Training Setup

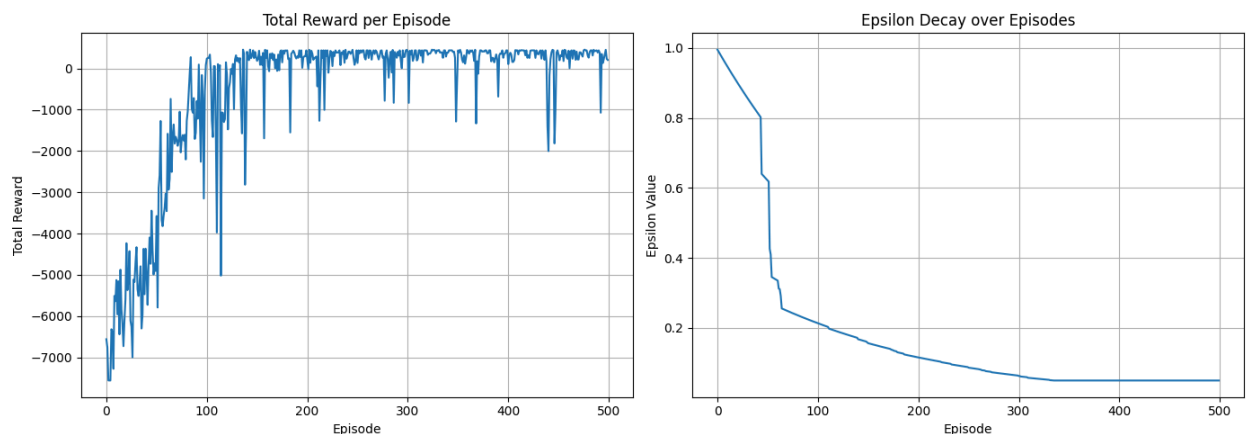
The SARSA agents were trained under the following settings:

- Episodes: 500
- Learning rate (α): 0.2
- Discount factor (γ): 0.95
- Initial epsilon: 1.0
- Minimum epsilon: 0.05
- Epsilon decay factor: 0.995
- Maximum timesteps per episode: 500

Results and Evaluation

The training performance is visualized in the following plots:

1. Total Reward per Episode:
The agent begins with poor performance (~ -7000 per episode) but improves steadily, surpassing zero around episode 100. The reward stabilizes close to +438 in later episodes, although with more fluctuations compared to Q-learning.
2. Epsilon Decay over Episodes:
Epsilon decays more slowly compared to Q-learning, reaching 0.05 after about 300 episodes. This prolonged exploration allowed the agent to discover better paths but also caused more variability in rewards.



During greedy policy evaluation ($\epsilon=0$), the learned policy consistently achieved a total reward of 438 across 10 evaluation episodes:

```
Greedy Evaluation Episode 1: Total Reward = 438
Greedy Evaluation Episode 2: Total Reward = 438
Greedy Evaluation Episode 3: Total Reward = 438
Greedy Evaluation Episode 4: Total Reward = 438
Greedy Evaluation Episode 5: Total Reward = 438
Greedy Evaluation Episode 6: Total Reward = 438
Greedy Evaluation Episode 7: Total Reward = 438
Greedy Evaluation Episode 8: Total Reward = 438
Greedy Evaluation Episode 9: Total Reward = 438
Greedy Evaluation Episode 10: Total Reward = 438
```

This confirms that SARSA successfully learned a near-optimal policy, slightly below the Q-learning performance (which achieved 452).

The SARSA agents learned a policy that consistently achieves total rewards around 438, slightly lower than the 452 achieved by Q-learning. The slower decay of epsilon and the on-policy nature of SARSA resulted in more cautious exploration and a more conservative policy.

While both algorithms converged to strong policies, Q-learning reached slightly higher rewards with faster convergence. This is expected since Q-learning's off-policy updates tend to favor more optimistic value estimates, whereas SARSA's updates follow the actual policy behavior.

Both approaches successfully solved the environment within 500 episodes, showing robustness under the shaped reward design and sparse delivery goals.

Double Q-Learning Implementation

To address overestimation bias in standard Q-learning, We implemented a Double Q-Learning algorithm for the multi-agent DroneDeliveryMultiAgentEnv. Each drone maintains two separate Q-tables (Q_A and Q_B). Updates alternate randomly between the two tables, using one table to select the best next action and the other to estimate its value

If updating Q_A:

$$Q_A(s,a) \leftarrow Q_A(s,a) + \alpha [r + \gamma Q_B(s', \operatorname{argmax}_{a'} Q_A(s',a')) - Q_A(s,a)]$$

Else:

$$Q_B(s,a) \leftarrow Q_B(s,a) + \alpha [r + \gamma Q_A(s', \operatorname{argmax}_{a'} Q_B(s',a')) - Q_B(s,a)]$$

This approach reduces the overoptimistic value estimates seen in regular Q-learning by decoupling action selection and evaluation.

Agents follow an epsilon-greedy policy with decaying epsilon, similar to the other algorithms.

Training Setup

The Double Q-learning agents were trained under the following configuration:

- Episodes: 1300
- Learning rate (α): 0.1
- Discount factor (γ): 0.99
- Initial epsilon: 1.0
- Minimum epsilon: 0.01
- Epsilon decay factor: 0.995
- Maximum timesteps per episode: 1000

We increased the training episodes compared to Q-learning and SARSA to give the agents more time to converge under the double update structure.

Results and Evaluation

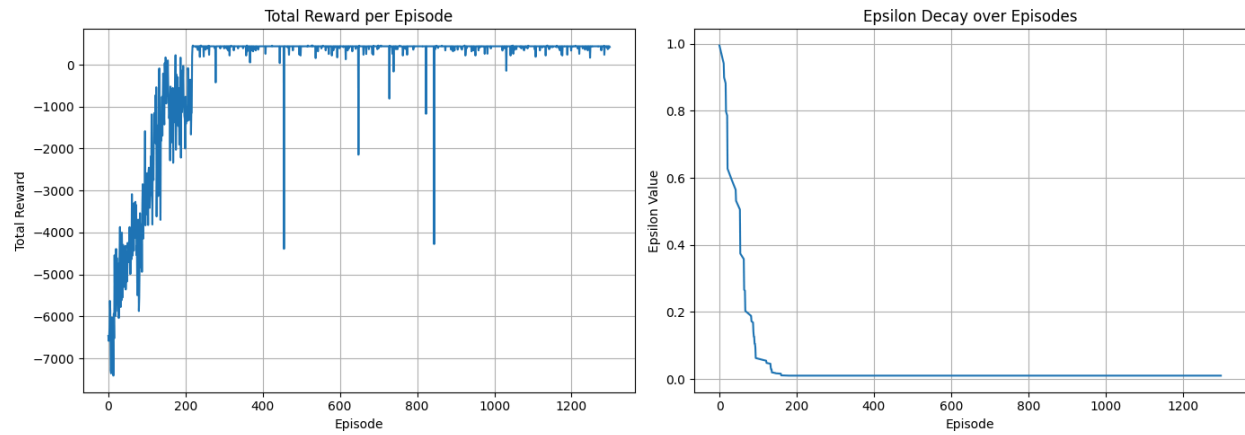
The training performance is shown in the following plots:

Total Reward per Episode:

The agents started with poor rewards around -7000 but steadily improved, reaching positive rewards near +438 by around episode 200. The rewards stabilized with occasional drops likely due to environment stochasticity or exploration.

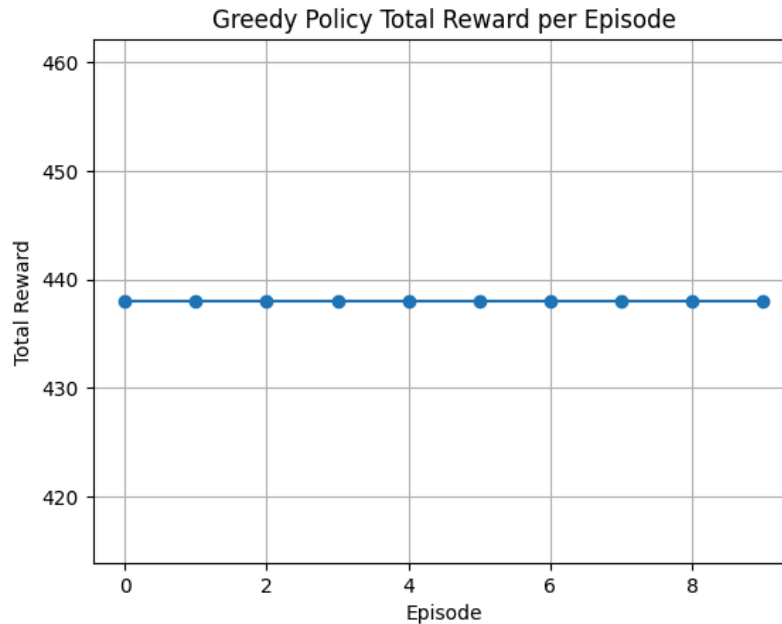
Epsilon Decay over Episodes:

Epsilon decayed rapidly, reaching the minimum of 0.01 around episode 150 and remaining constant thereafter, allowing full exploitation during the majority of training.



During greedy evaluation ($\epsilon=0$), the policy consistently achieved a total reward of 438 across 10 evaluation episodes:

```
Greedy Evaluation Episode 1: Total Reward = 438
Greedy Evaluation Episode 2: Total Reward = 438
Greedy Evaluation Episode 3: Total Reward = 438
Greedy Evaluation Episode 4: Total Reward = 438
Greedy Evaluation Episode 5: Total Reward = 438
Greedy Evaluation Episode 6: Total Reward = 438
Greedy Evaluation Episode 7: Total Reward = 438
Greedy Evaluation Episode 8: Total Reward = 438
Greedy Evaluation Episode 9: Total Reward = 438
Greedy Evaluation Episode 10: Total Reward = 438
```



Double Q-learning achieved stable and reliable performance, reaching a reward plateau around 438, similar to SARSA. Its behavior shows reduced overestimation but also slightly underestimates compared to Q-learning, which achieved a higher final reward of 452.

The policy learned by Double Q-learning is robust and consistent across evaluations, indicating effective convergence without significant variance. However, it required more episodes (1300) to stabilize compared to Q-learning (500), reflecting slower convergence under the more conservative update mechanism.

Double Q-learning is beneficial for environments where overestimation can lead to risky policies, offering a tradeoff between optimism and caution in policy learning.

Comparison of Tabular

The following plots summarize the learning performance across Q-learning, SARSA, and Double Q-learning in the DroneDeliveryMultiAgentEnv:

1. Total Reward Comparison (left plot):

All three algorithms started with large negative rewards (around -8000) but progressively improved towards positive rewards as training progressed.

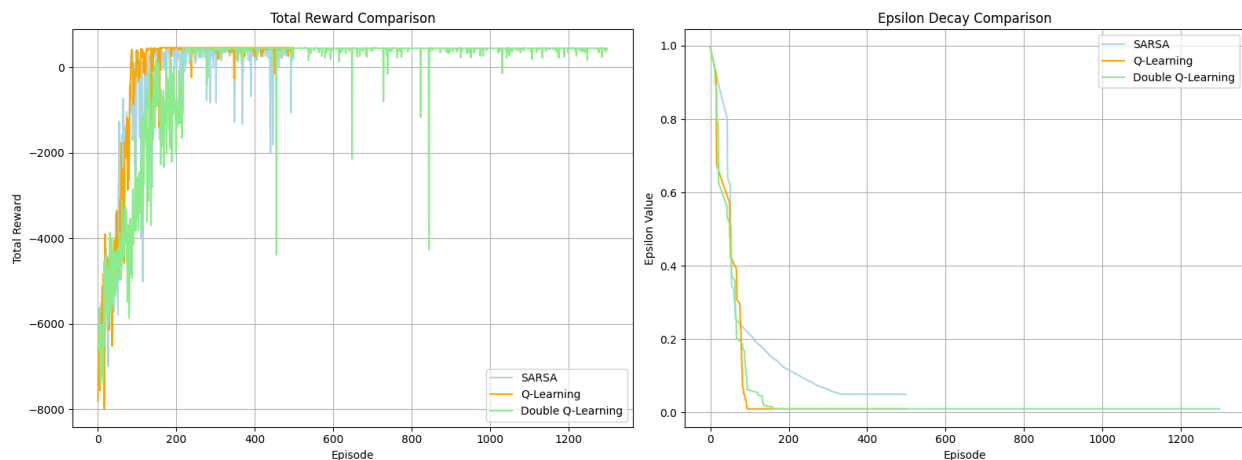
- Q-learning achieved the fastest convergence, surpassing zero around episode 90 and stabilizing at approximately +452 total reward.
- SARSA reached positive rewards slightly later (around episode 110) and stabilized near +438.

- Double Q-learning followed a similar trajectory to SARSA, converging around +438 but requiring more episodes (~200) to fully stabilize.
2. Occasional negative reward spikes in Double Q-learning indicate isolated failures (e.g., collisions or entering no-fly zones) despite an overall stable policy.
 3. Epsilon Decay Comparison (right plot):
 - Q-learning and Double Q-learning decayed epsilon more quickly, reaching minimum exploration ($\epsilon=0.01$) within ~150 episodes.
 - SARSA decayed more slowly, maintaining higher exploration for a longer period and reaching $\epsilon=0.05$ around episode 300.

Overall Analysis

- Q-learning outperformed SARSA and Double Q-learning in final reward, achieving ~452 compared to ~438.
- Double Q-learning and SARSA produced more conservative policies, converging to similar performance but slightly lower rewards than Q-learning.
- Q-learning exhibited faster convergence and greater optimism (due to maximization bias), while Double Q-learning mitigated overestimation at the cost of slower convergence.

All three algorithms successfully solved the environment by learning policies capable of delivering packages while avoiding penalties. However, the choice of algorithm can depend on tradeoffs between learning speed, policy optimism, and stability.



Applying Deep RL methods

Deep Q-Network

I implemented a Deep Q-Network (DQN) to apply function approximation to the multi-agent DroneDeliveryMultiAgentEnv. In this implementation, I used centralized training by feeding the global state (obtained from `get_centralized_state()`) into a single neural network to learn a joint Q-function for both agents.

The DQN architecture consisted of:

- Input layer: `state_size` (centralized state vector)
- Two hidden layers with 256 neurons and ReLU activations
- Output layer: `action_size` (joint action space)

The DQN was trained using experience replay (buffer of size 50,000) and a target network updated every 20 episodes to stabilize learning. The loss was computed using Mean Squared Error (MSE) between the predicted Q-values and the target Q-values.

The action selection followed an epsilon-greedy policy over the joint action space, with each index decoded into individual drone actions.

Training Setup

The DQN was trained with the following hyperparameters:

- Episodes: 3000
- Learning rate (lr): 0.001
- Discount factor (γ): 0.99
- Initial epsilon: 1.0
- Minimum epsilon: 0.05
- Epsilon decay factor: 0.995
- Target network update frequency: every 20 episodes
- Batch size: 64
- Max timesteps per episode: 300

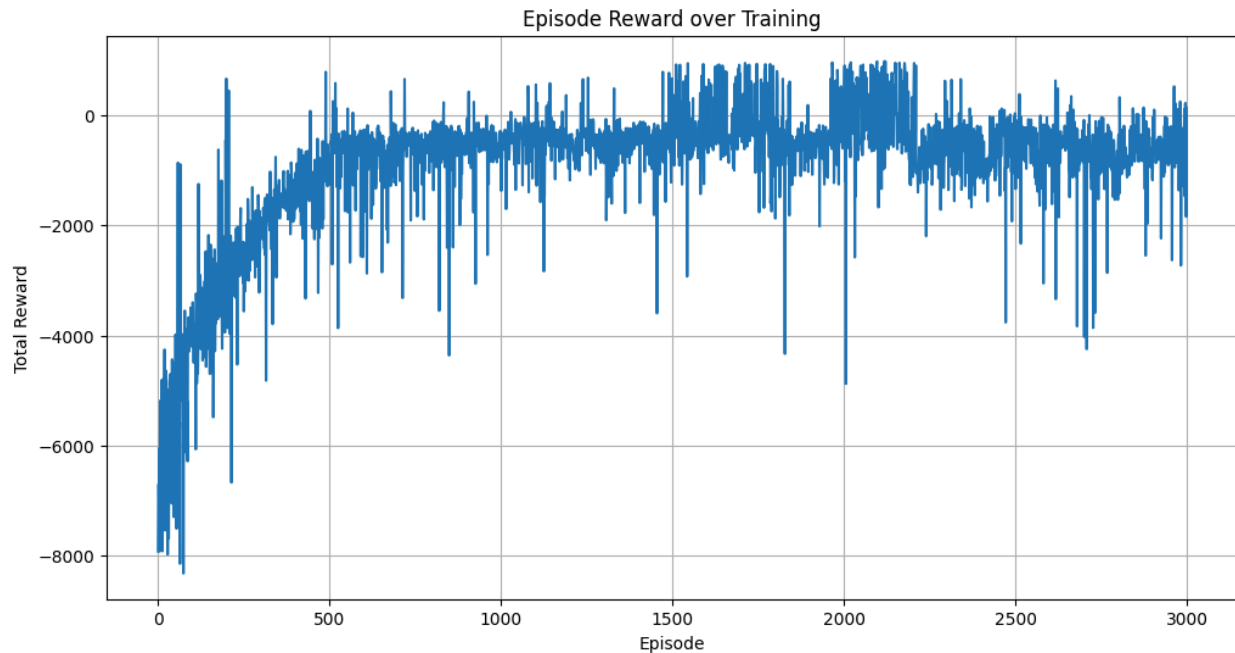
Training used centralized state inputs with decentralized execution during environment interaction.

Results and Evaluation

The training performance is summarized below:

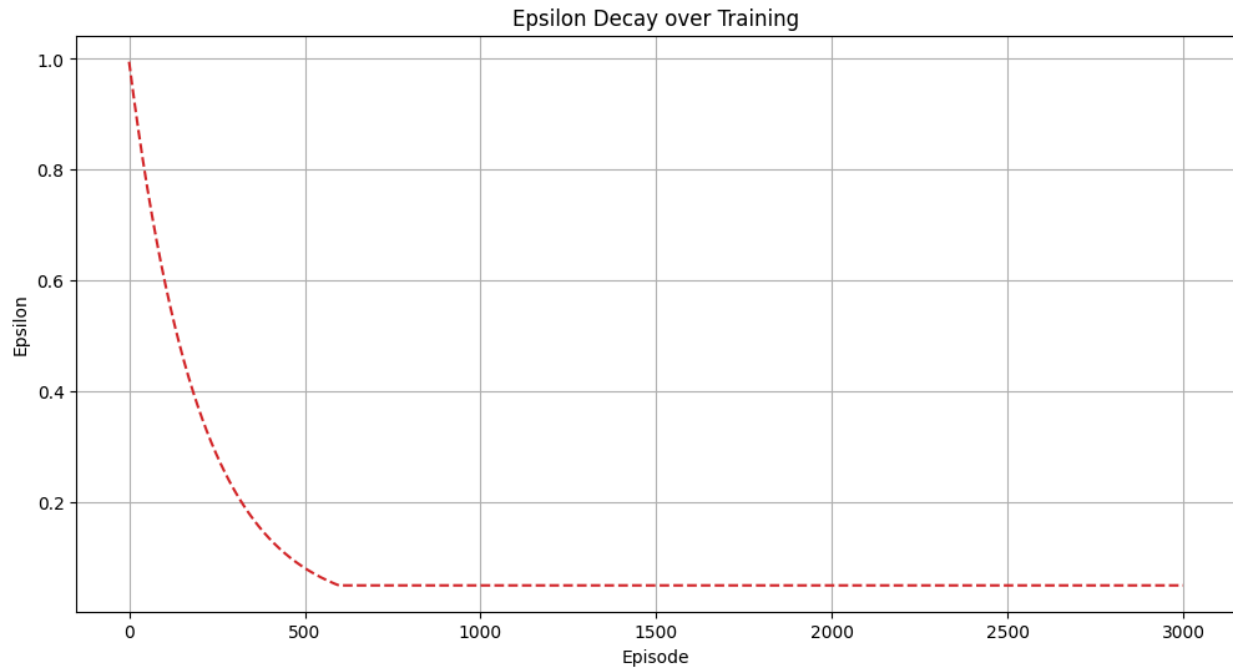
Episode Reward over Training:

Rewards improved significantly from an initial average around -8000 to fluctuating near -500 to 0 across episodes after 2000. The plot shows greater reward variance compared to tabular methods, with occasional negative reward spikes even after long training.

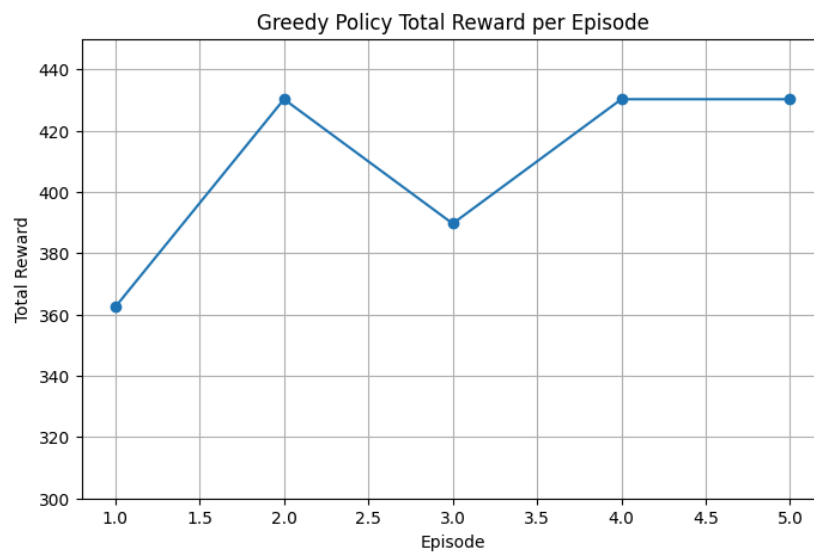


Epsilon Decay over Training:

Epsilon decayed smoothly from 1.0 to the minimum of 0.05 by around episode 600, promoting exploration early and transitioning to exploitation for the remaining episodes.



During greedy policy evaluation ($\epsilon=0$), the agent achieved the following results over 5 evaluation episodes:



The DQN achieved near-optimal performance (~ 430 total reward) but with higher variance in both training and evaluation compared to the tabular methods. This variance is attributed to the function approximation and stochastic optimization processes in deep reinforcement learning.

Compared to Q-learning (which achieved 452) and SARSA/Double Q-learning (438), DQN slightly underperformed in final reward but had the advantage of generalization across the centralized state space, enabling scalability to more complex environments.

QMIX implementation

In addition to tabular and deep Q-learning approaches, I implemented QMIX (Monotonic Value Function Factorization) to address multi-agent credit assignment while maintaining centralized training with decentralized execution (CTDE).

QMIX decomposes the global Q-value into individual agent Q-values via a monotonic mixing network:

$$Q_{\text{total}} = f(Q_1, Q_2, \dots, Q_n | s)$$

where f is a mixing function parameterized by a neural network conditioned on the global state. This ensures that an increase in an individual agent's Q-value never reduces the global Q-value, allowing decentralized greedy actions to optimize the centralized objective.

The implementation includes:

- Individual AgentQNetwork for each agent (input: local observation; output: per-agent Q-values)
- A MixerNetwork (input: agent Q-values and global state; output: Q_{total})
- A target network pair for stability
- A ReplayBuffer for experience replay

Training Setup

QMIX was trained with the following configuration:

- Episodes: 2000
- Learning rate (LR): 0.001
- Discount factor (γ): 0.99
- Initial epsilon: 1.0
- Minimum epsilon: 0.01
- Epsilon decay factor: 0.995
- Max steps per episode: 300
- Batch size: 64
- Replay buffer size: 10,000
- Target network updates: periodic soft copy

The agents acted independently based on their local Q-values, while the mixer combined them during training using the centralized state.

Results and Evaluation

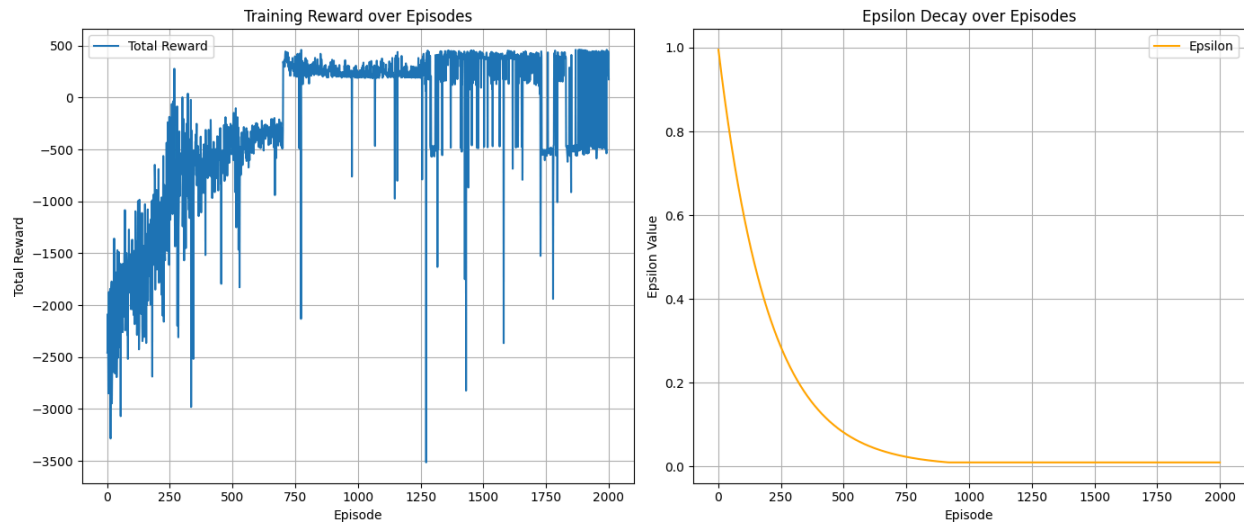
The training performance is summarized below:

1. Training Reward over Episodes:

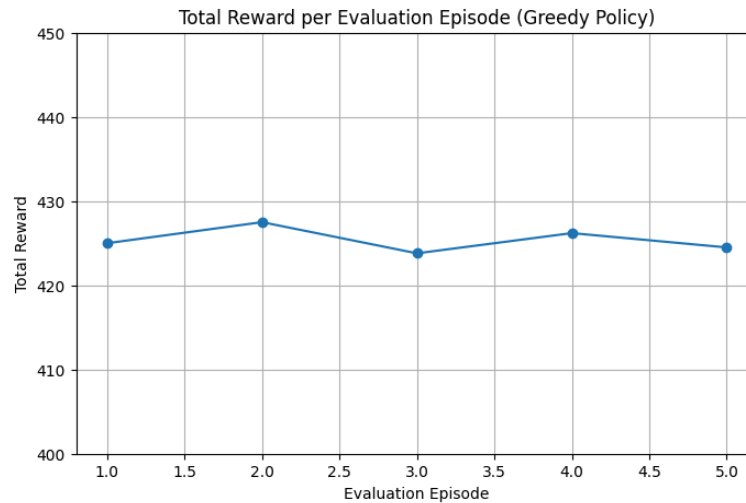
The total reward improves from -3000 at the start to a stable range between +400 and +500 by episode 1200. The rewards plateau with occasional dips likely due to stochasticity in environment events like collisions or no-fly zones.

2. Epsilon Decay over Episodes:

Epsilon decays smoothly from 1.0 to the minimum 0.01 by around episode 900, supporting a long exploratory phase before exploitation.



During greedy evaluation ($\epsilon=0$), QMIX achieved highly consistent rewards:



- Both DQN and QMIX are able to learn effective policies in the DroneDeliveryMultiAgentEnv.
- DQN learns a centralized Q-function over the joint action space but struggles slightly with scalability and coordination → leading to higher variance and slower convergence.
- QMIX leverages individual agent Q-networks with a mixing network, allowing better credit assignment and agent coordination → resulting in faster convergence and more stable rewards.
- Therefore, QMIX is more sample-efficient and robust in this multi-agent setting, while DQN is still effective but slower to converge.

Existing MARL Environment

Simple Spread (PettingZoo / MPE2 Environment Overview)

Environment name: simple_spread_v3

Library: PettingZoo → migrated to MPE2 package

Scenario type: Multi-Agent Cooperative

Description:

- Simple Spread is a cooperative multi-agent environment where N agents must collaboratively cover N landmarks (default $N=3$) while avoiding collisions.
- The goal is to minimize the total distance between each landmark and its closest agent.

Agents receive:

- Global reward: Negative sum of minimum distances from agents to landmarks.
- Local penalty: -1 reward per collision with other agents.
- The balance between global and local rewards is controlled by the `local_ratio` parameter.

Agent Observations:

- Each agent observes:
- Its own velocity (2D)
- Its own position (2D)
- Relative positions of all landmarks (N landmarks \times 2D)
- Relative positions of other agents ($N-1 \times$ 2D)
- Communication (usually zeros in this scenario)

→ Total dimension = 18 (for $N=3$)

Agent Actions:

Agents choose among 5 discrete actions:

- 0: No-op
- 1: Move left
- 2: Move right
- 3: Move down
- 4: Move up

Can also be continuous (if continuous_actions=True)

Key Arguments:

- N: Number of agents and landmarks (default: 3)
- local_ratio: Weighting of local vs. global rewards (0.0 → fully global, 1.0 → fully local)
- max_cycles: Episode length (default: 25)
- continuous_actions: Whether action space is continuous or discrete
- dynamic_rescaling: Enable/disable size scaling of agents and landmarks

Applying Deep RL methods

Deep Q-Network

We have implemented a Deep Q-Network (DQN) for the cooperative multi-agent environment `simple_spread_v3` from the `PettingZoo` library. In this implementation, we followed decentralized training, where each agent independently learned its own Q-function using only its local observation space.

The DQN architecture for each agent consisted of:

- **Input layer:** 18-dimensional local observation
- **Two hidden layers:** 256 and 128 neurons with ReLU activations
- **Output layer:** 5-dimensional output representing Q-values for each discrete action

Each agent maintained its own Q-network and target network. The Q-networks were trained using experience replay with a buffer size of 1000 transitions per agent, and target networks were synchronized every 10 episodes to stabilize learning. The loss function used was Mean Squared Error (MSE) between the predicted and target Q-values computed via the Bellman equation.

Action selection followed an ϵ -greedy strategy, allowing exploration during training. The agent selected a random action with probability ϵ or the highest Q-value action otherwise.

Double Deep Q-Network

To mitigate the overestimation bias in standard DQN, we implemented Double DQN for the `simple_spread_v3` environment. Like DQN, training was decentralized, with each agent learning a separate Q-function based on its local observations.

Each agent's Double DQN network had the same architecture:

- **Input layer:** 18-dimensional observation
- **Hidden layers:** 256 and 128 neurons with ReLU activation
- **Output layer:** 5-dimensional action Q-values

The key difference was in the target value computation. Instead of using the maximum Q-value from the target network, Double DQN decouples the selection and evaluation steps:

- The action is selected using the online network.

- The value of the selected action is evaluated using the target network.

Experience replay buffers (size 1000) and target networks were used per agent. Loss was computed via MSE.

The use of Double DQN significantly improved the stability of value estimates compared to standard DQN.

Dueling Double Deep Q-Network

We implemented a Dueling Double DQN agent architecture to further improve learning stability and performance in `simple_spread_v3`. Training was decentralized: each agent trained its own network independently from others.

The architecture was modified to separate value and advantage estimation:

- **Shared layers:** FC(18 \rightarrow 256 \rightarrow 128), ReLU
- **Value stream:** Outputs a scalar $V(s)$
- **Advantage stream:** Outputs a vector $A(s,a)$
- **Q-value combination:** $Q(s, a) = V(s) + A(s, a) - (1 / |A|) \times \sum A(s, a')$
 - $Q(s, a)$ is the estimated Q-value for state s and action a
 - $V(s)$ is the estimated value of the state
 - $A(s, a)$ is the advantage of taking action a in state s
 - $|A|$ is the number of possible actions
 - $\sum A(s, a')$ is the sum of advantages over all actions

This decomposition allows the network to better distinguish between valuable states and the advantage of specific actions in those states.

Training used standard DQN techniques with:

- Experience replay (buffer size 1000)
- Target network synced every 10 episodes
- MSE loss between predicted and target Q-values
- ϵ -greedy policy for action selection

The dueling architecture helped agents learn more efficiently in states where action differences were marginal, and it showed the best overall reward performance.

Training Setup & Hyperparameters:

Common Environment Setup:

- **Environment:** `simple_spread_v3.parallel_env(local_ratio=0.5)`
- **Number of agents:** 3
- **Observation dim:** 18
- **Action dim:** 5 (discrete)

Hyperparameter	Value	Description
Episodes	100000	Training - Number of episodes
Learning Rate	1e-3	Step size for the Adam optimizer
Discount Factor (γ)	0.9	Weights future rewards
Replay Buffer Size	1000	Max transitions stored per agent
Batch Size	32	Number of transitions sampled per update
Max Steps per Episode	25	Steps taken per episode
Target Network Sync	Every 10 episodes	Frequency to update target Q-network
Epsilon Start (ϵ)	1.0	Initial exploration rate
Epsilon Final (ϵ)	0.01	Minimum exploration rate
Epsilon Decay	Exponential	Gradually reduces exploration over episodes
Optimizer	Adam	Used for updating network parameters
Loss Function	Mean Squared Error	Used to minimize Q-value prediction error
Training Approach	Decentralized	Each agent learns independently

Each agent trained independently using its own experience buffer, policy network, and optimizer.

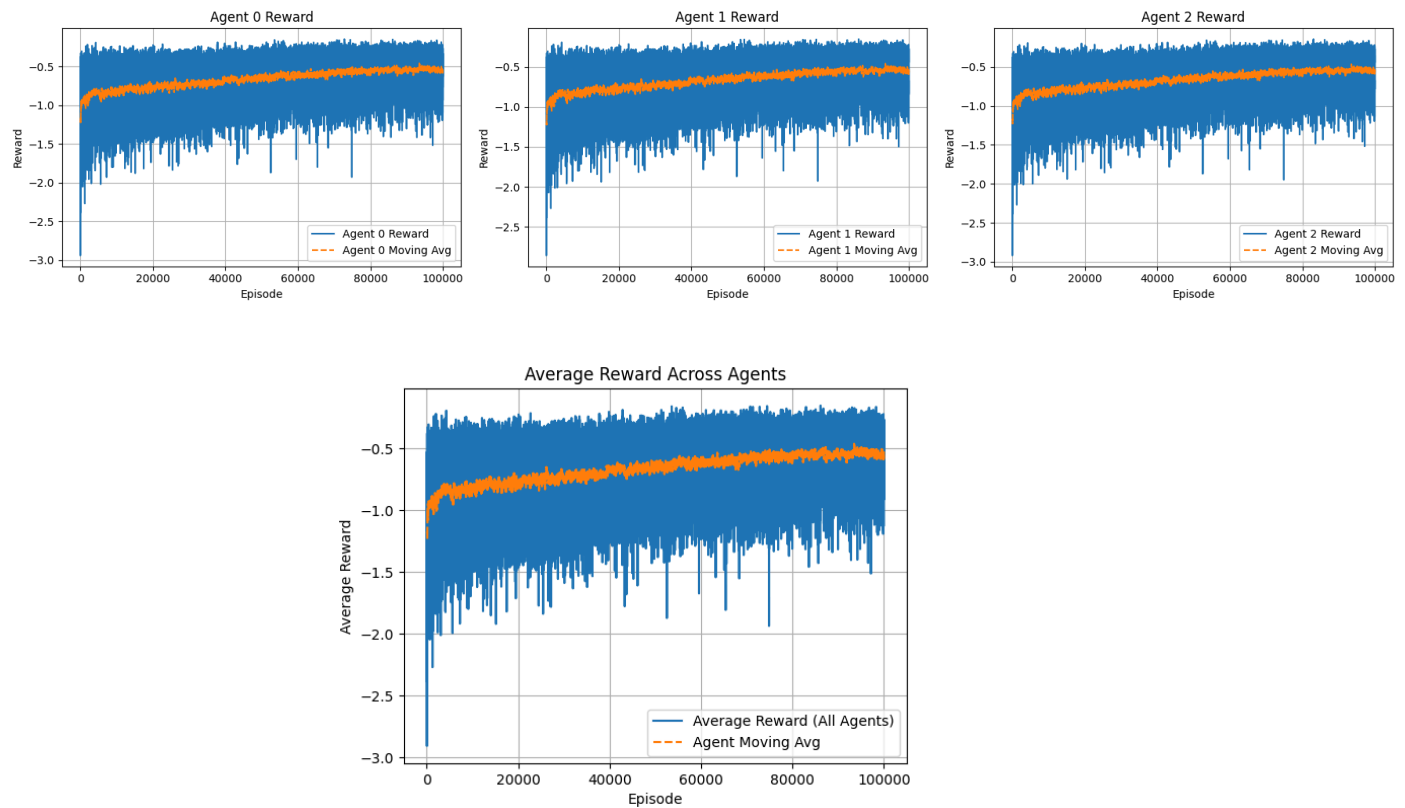
Aspect	DQN	Double DQN	Dueling DQN
Q-value Target Calculation	$r + \gamma \max_{a'} Q'(s', a')$	$r + \gamma Q'(s', \arg\max_{a'} Q(s', a'))$	Same as Double DQN
Q-Network Architecture	FC (18 → 256 → 128 → 5)	FC (18 → 256 → 128 → 5)	Shared FC (18 → 256 → 128), then: • Value head → 1 • Advantage head → 5
Overestimation Bias Control	-	Action selection & evaluation decoupled	Uses Double DQN logic + better separation of value/advantage
Value & Advantage Streams	-	-	Value (V) and Advantage (A) streams explicitly modeled

Results and Evaluation

DQN:

The training performance is summarized below:

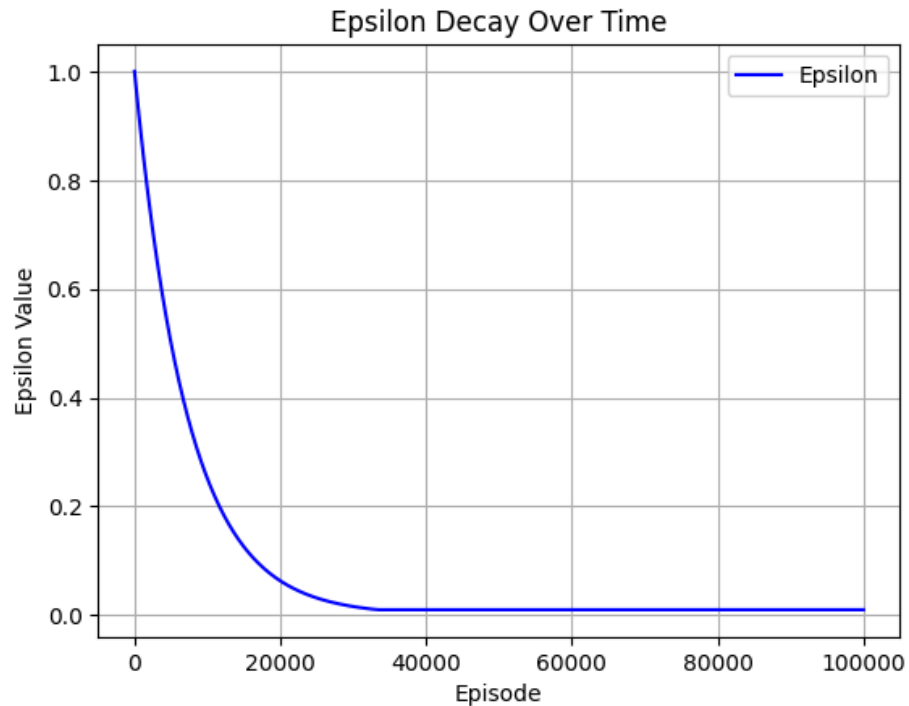
Episode Reward over Training (Per-Agent & Average):



Inference:

- **Trend:** All three agents exhibit a positive upward trend in episode rewards over time, with their moving averages steadily improving.
- **Noise:** Despite high reward variance initially, the moving average (orange line) smooths out and stabilizes over time.
- **Average Reward Curve:** Indicates a general improvement in cooperative behavior, with agents learning to better navigate toward landmarks.
- **Plateau:** The average reward across agents begins to plateau around -0.6, suggesting convergence.

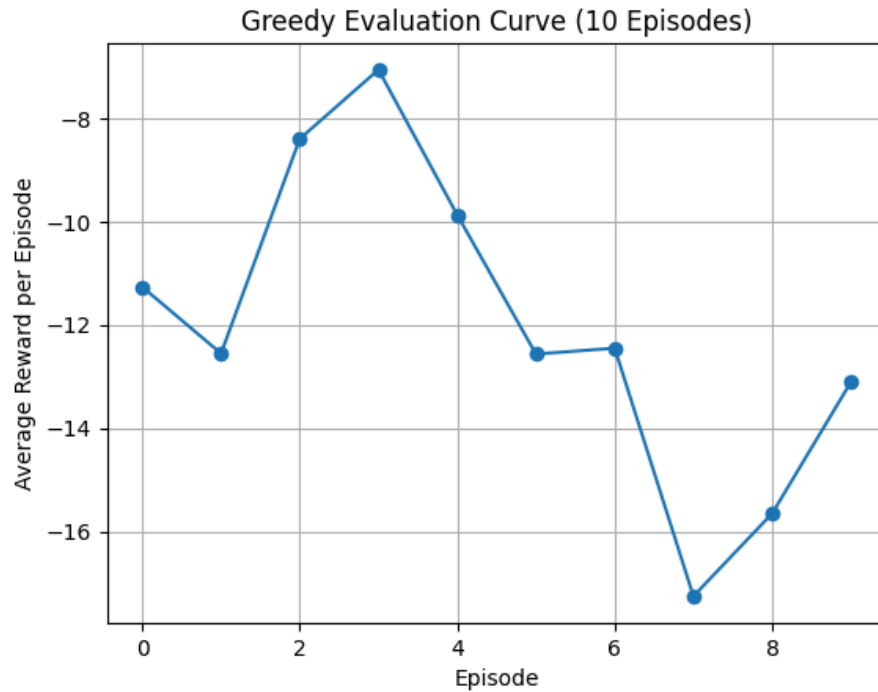
Epsilon Decay over Training:



Inference:

- **Observation:** Epsilon decayed exponentially from 1.0 to ~ 0.01 over the course of 100,000 episodes.
- **Effect:**
 - In early episodes (high ϵ), agents explored various strategies.
 - Over time, as ϵ dropped, agents exploited learned Q-values more reliably.
- **Result:** Gradual decay helped stabilize training by ensuring sufficient exploration early and reliable policy execution later.

Evaluation (Greedy Policy Performance):



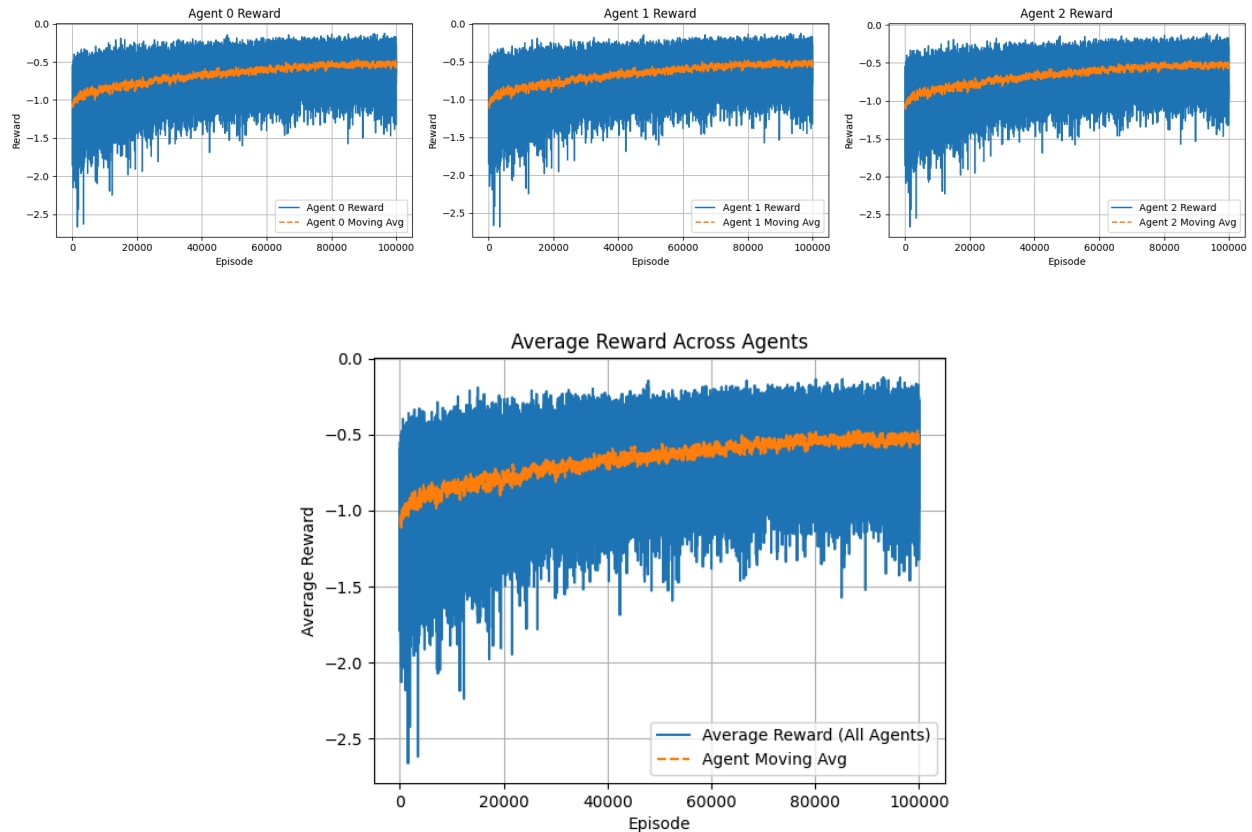
Inference:

- **Setting:** Evaluation with $\epsilon = 0$ (greedy policy), run for 10 episodes.
- **Results:**
 - The average reward fluctuates between -8 to -17, indicating varying episode quality.
 - Peak performance occurs in episode 3, suggesting the learned policy has potential but still exhibits inconsistency.
- **Interpretation:**
 - Performance variance may stem from stochastic dynamics or insufficiently generalized Q-values.
 - Policy is capable but not robustly stable yet — may benefit from further fine-tuning or advanced methods (e.g., Double DQN).

DQN-trained agents demonstrate learned cooperative strategies but still face stability challenges in deterministic evaluation runs.

Double DQN:

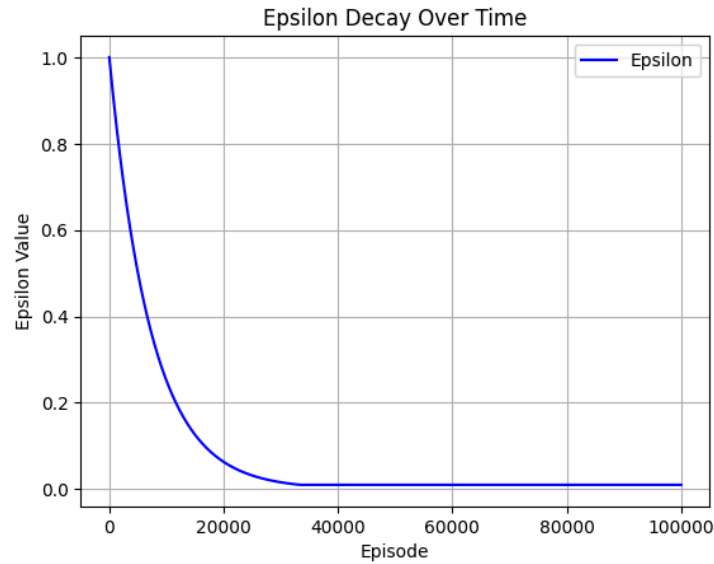
Episode Reward over Training (Per-Agent & Average):



Inference:

- **Overall Trend:** All three agents display a clear upward trend, indicating successful learning over time.
 - **Smoothness:** Compared to DQN, the reward signals appear more stable and converge slightly faster.
 - **Average Reward Curve:** The moving average curve shows consistent improvement, peaking close to -0.5, better than DQN.
- Reward Variance:** Reward variance still exists but is relatively controlled due to better Q-value estimation.

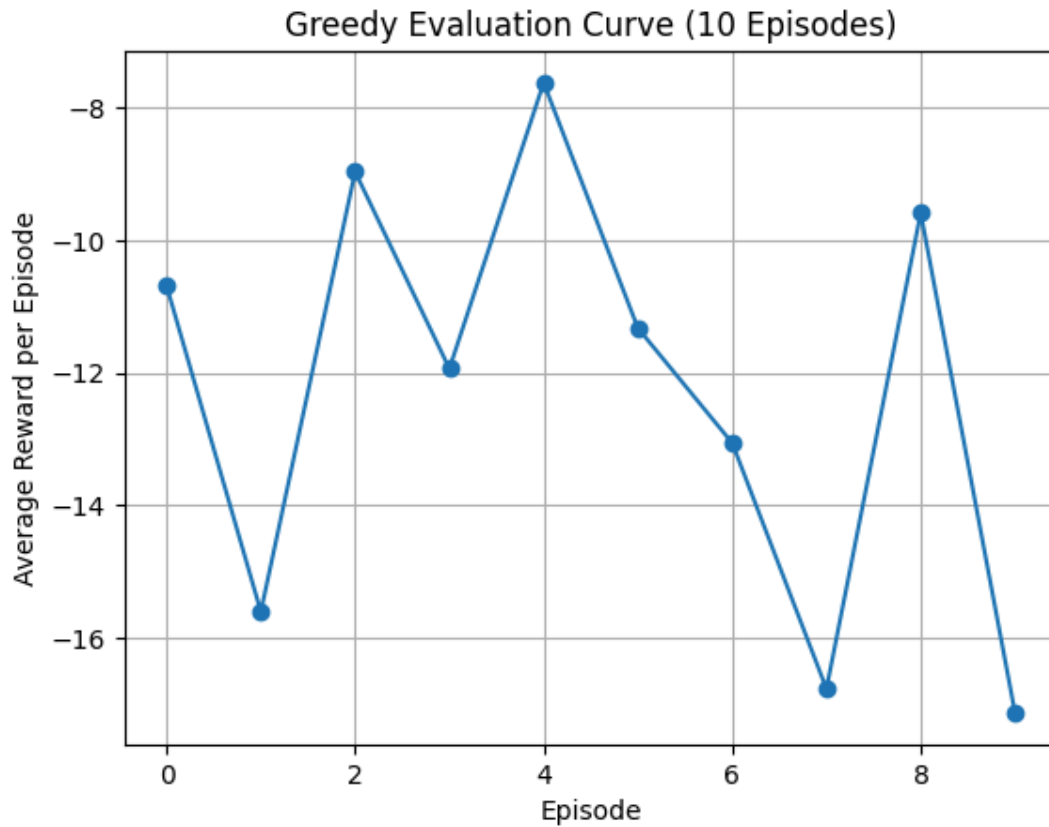
Epsilon Decay over Training:



Inference:

- **Decay Profile:** Epsilon decays exponentially from 1.0 to 0.01 similar to DQN.
- **Exploration vs. Exploitation:**
 - Early high ϵ encourages thorough exploration.
 - Gradual decay ensures the transition to exploitation without destabilizing learning.
- **Result:** Double DQN benefits more from this schedule due to its more robust value updates.

Evaluation (Greedy Policy Performance):

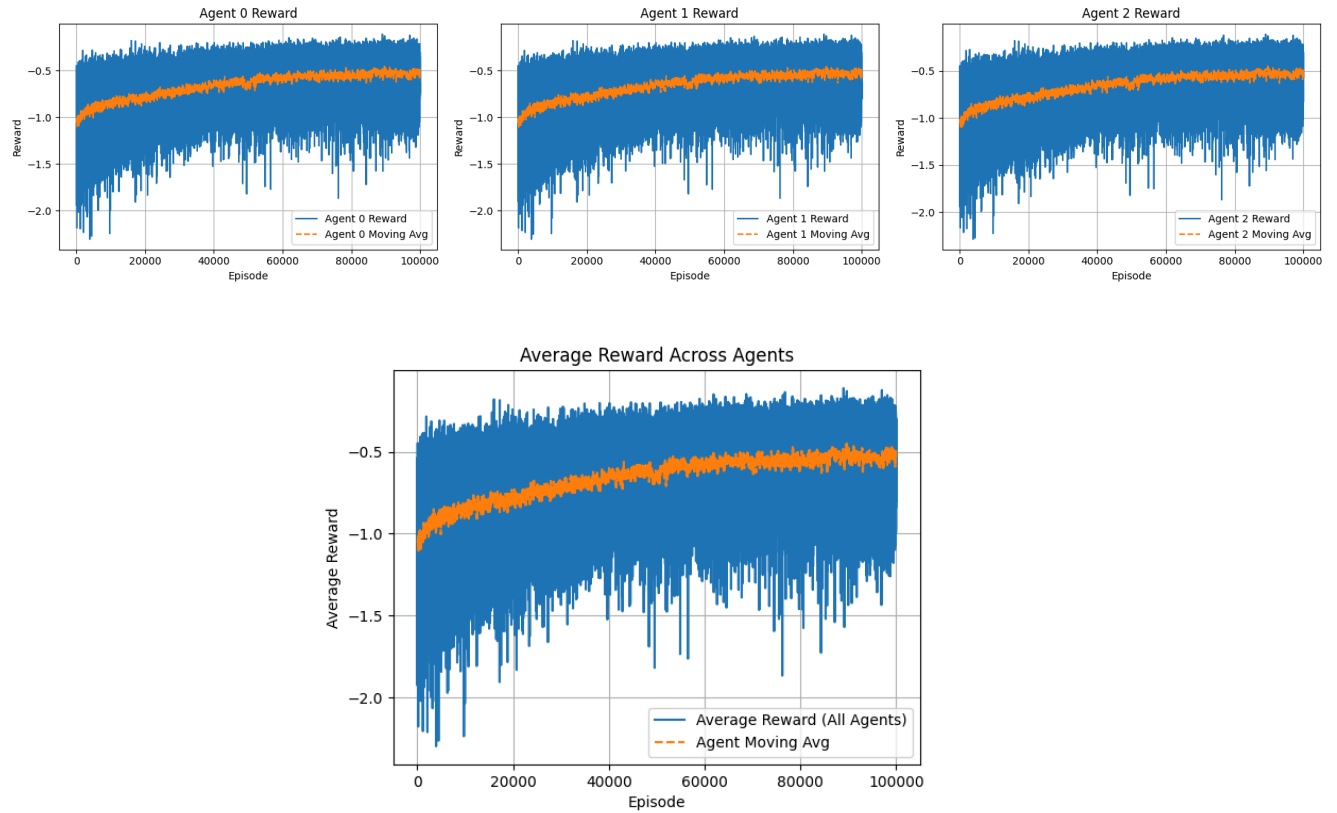


Inference:

- **Setup:** Agents evaluated with greedy policy ($\epsilon = 0$) for 10 episodes.
- **Results:**
 - Average rewards range from approximately -8 to -17, with a few spikes near -8, suggesting better peak performance than DQN.
 - Fluctuations still occur but are less severe compared to DQN.
- **Interpretation:**
 - The learned policy is more consistent and robust.
 - Agents have a better sense of cooperation and spatial awareness under greedy execution.

Dueling Double DQN:

Episode Reward over Training (Per-Agent & Average):

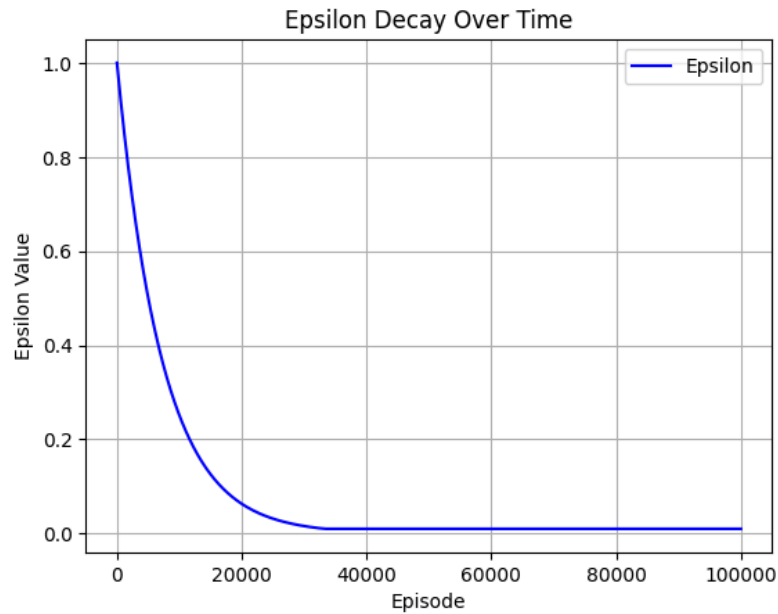


Inference:

- **Overall Trend:** All agents show a strong and consistently increasing reward trend.
- **Stability:** The moving averages are noticeably smoother and more stable compared to both DQN and Double DQN.
- **Average Reward:**
 - Peak average reward approaches -0.4 to -0.5, the best among all methods.
 - The variance remains low, indicating stable cooperative strategies.

Dueling Double DQN results in superior convergence and reward stability, validating the effectiveness of decomposing state value and action advantage.

Epsilon Decay over Training:

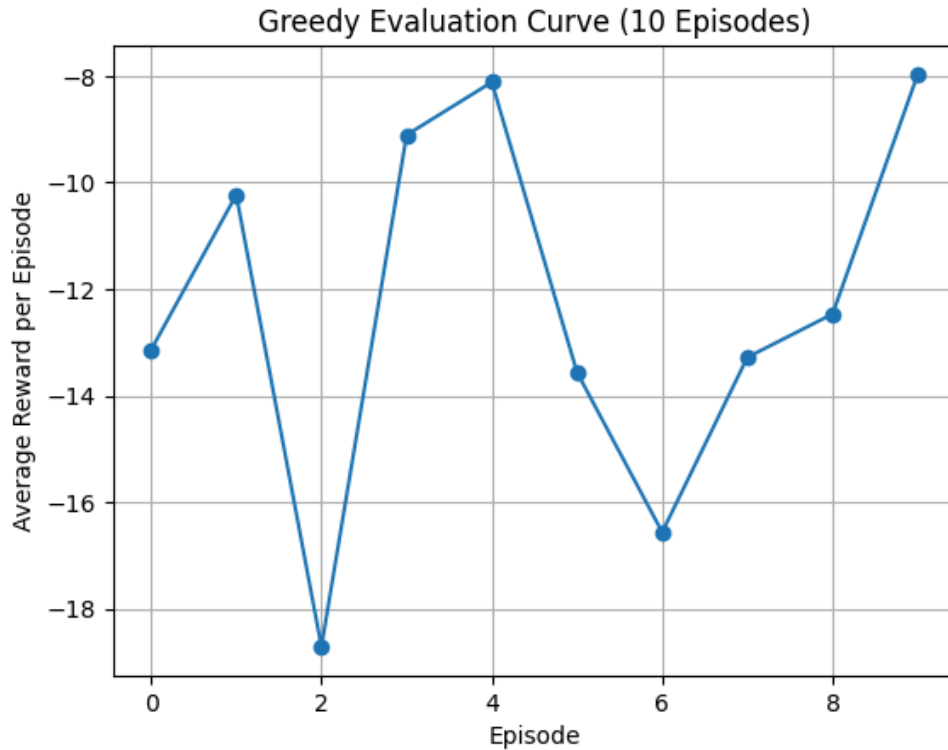


Inference:

- **Observation:** Exponential decay curve remains identical in shape to previous algorithms.
- **Impact:**
 - Early episodes focus on exploration, helping agents differentiate value and advantage early in training.
 - Final ϵ near 0 ensures almost purely greedy action selection toward the end.

Due to the more expressive Q-network architecture, the agent benefits more effectively from the same epsilon decay schedule than simpler counterparts.

Evaluation (Greedy Policy Performance):

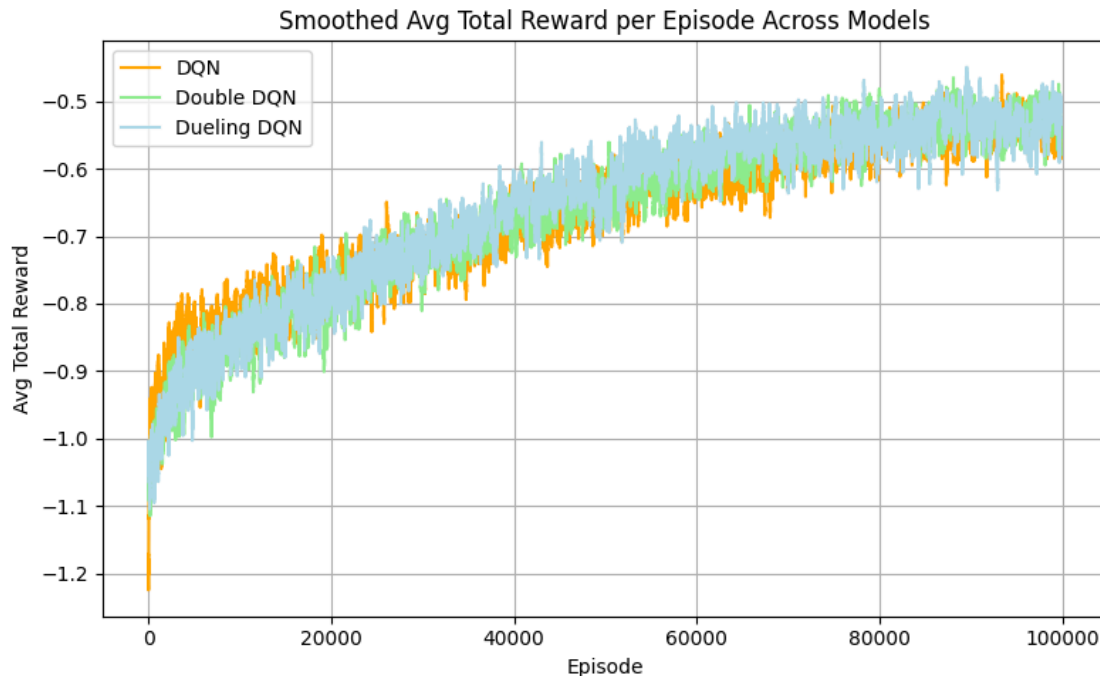


Inference:

- **Setup:** Agents evaluated using greedy policy ($\epsilon = 0$) for 10 episodes.
- **Results:**
 - Evaluation rewards consistently hover between -8 and -18, with 3 peaks nearing -8, the highest among all three algorithms.
 - There's still some fluctuation, but reward drops are narrower, indicating better policy robustness.
- **Interpretation:**
 - Agents show high-quality cooperation under deterministic evaluation.
 - The dueling structure helps select meaningful actions, even when action advantage values are similar.

Comparison of Deep RL Algorithms (DQN, Double DQN, Dueling DQN)

1. Smoothed Avg Total Reward (Training Performance)

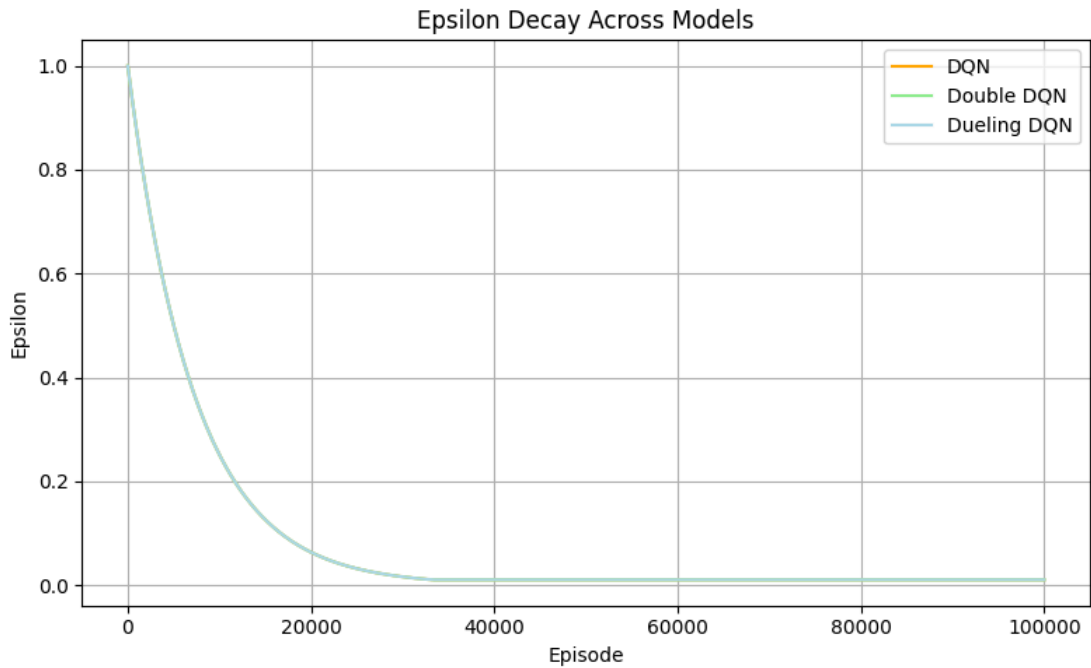


Insights from Plot:

- All models improve steadily, confirming learning over time.
- Dueling DQN consistently achieves the highest rewards, showing faster and more stable convergence.
- Double DQN outperforms DQN, particularly after the first 20,000 episodes, due to reduced overestimation bias.
- DQN lags slightly behind in final performance, with a lower reward ceiling and more variance.

Conclusion: Dueling DQN > Double DQN > DQN in terms of learning quality, reward magnitude, and convergence smoothness.

2. Epsilon Decay Across Models

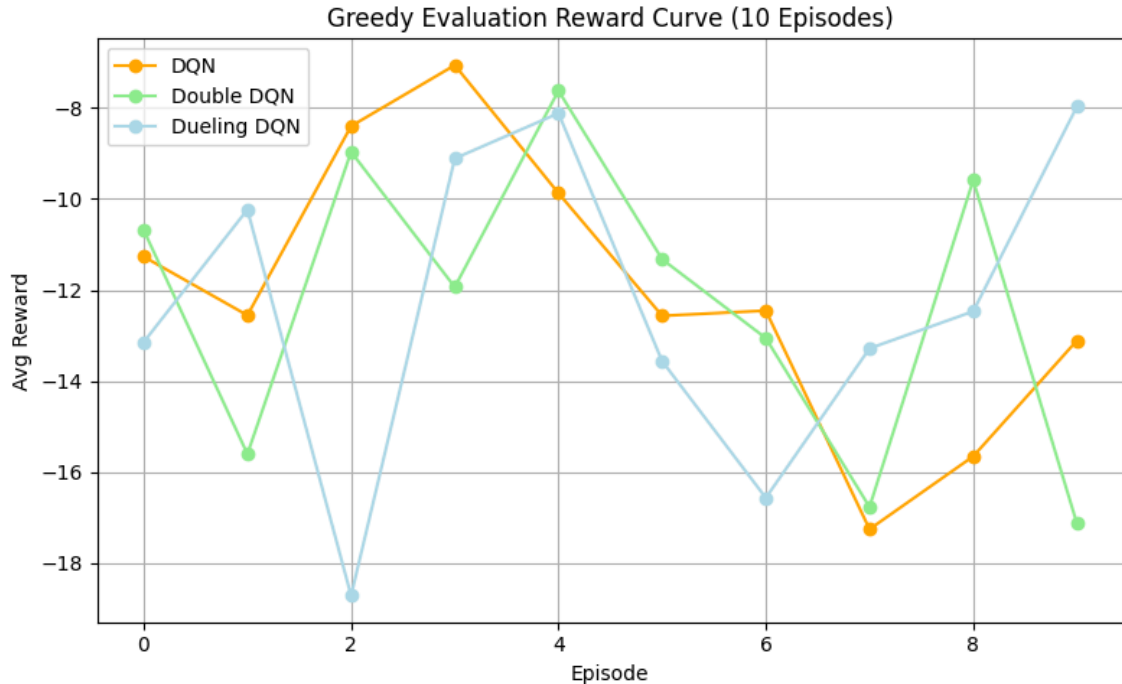


Insights from Plot:

- All models use the same exponential decay schedule from $\epsilon = 1.0$ to 0.01 over 100,000 episodes.
- Uniform decay ensures fair exploration-exploitation tradeoff for all models.

Conclusion: Decay behavior was identical and not a differentiating factor; performance differences stem purely from algorithmic improvements.

3. Greedy Evaluation Curve (10 Episodes)



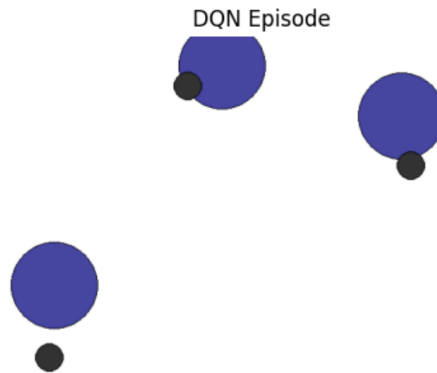
Insights from Plot:

- All three algorithms — DQN, Double DQN, and Dueling DQN — exhibit fluctuating episode rewards, with no algorithm consistently outperforming the others.
- Each model shows peaks and dips across different episodes, highlighting the inherent variance during short evaluation windows.

Conclusion: In this specific 10-episode evaluation window, all three algorithms show comparable performance with overlapping reward ranges. The episodic variability is high, likely due to stochasticity in environment initialization, agent start positions, or exploration residuals. Thus, no model is consistently superior based solely on this short evaluation, and longer evaluation or statistical averaging would be required for reliable ranking.

Qualitative inference for each algorithm based on the rendered final frame from a sample episode of each policy (DQN, Double DQN, Dueling DQN):

❖ **DQN Episode**



Visual Observation:

- Two agents are closely positioned to landmarks.
- One landmark is completely unattended.
- There's no clear separation or spatial allocation among agents.

Behavioral Inference:

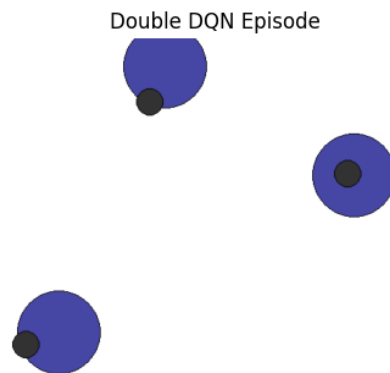
- Indicates partial learning of landmark coverage, but lacking full coordination.
- Likely that agents are competing or overlapping in their learned policies, possibly influenced by Q-value overestimation.
- The unassigned landmark suggests that agents haven't developed a robust cooperative strategy for global coverage.

Strategic Insight:

- DQN agents learn individual value approximations, but lack the expressive structure to handle subtle cooperative tradeoffs.
- The absence of state-value separation may hinder nuanced decision-making — especially in multi-agent cooperative tasks.

Summary: DQN shows signs of emerging cooperation, but the lack of precise role distribution leads to suboptimal global behavior.

❖ Double DQN Episode



Visual Observation:

- Each agent is positioned near a different landmark.
- Agents appear evenly spaced, minimizing overlap and ensuring all landmarks are covered.

Behavioral Inference:

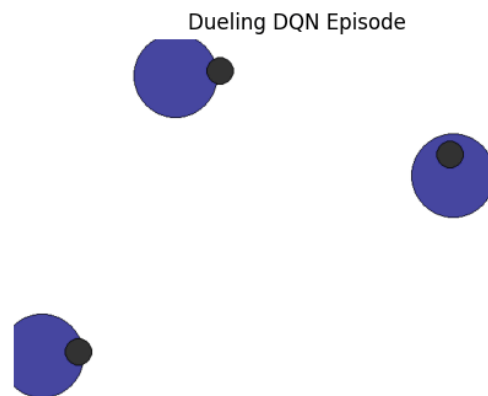
- Agents have learned to spread efficiently, suggesting reduced action-value bias and better policy convergence.
- Policies seem more deterministic and less noisy compared to DQN — an effect of stable Q-value targets.

Strategic Insight:

- The use of decoupled Q-learning targets helps agents distinguish subtle differences in action choices, especially in states with similar values.
- This leads to more stable behavior and improved task decomposition (i.e., landmark-to-agent assignment).

Summary: Double DQN agents display solid coordination and spatial awareness, with successful task allocation across all agents.

❖ Dueling DQN Episode



Visual Observation:

- Agents are tightly aligned with all three landmarks.
- The positioning is precise, minimalistic, and non-overlapping.
- Each agent has clearly “claimed” a landmark.

Behavioral Inference:

- Demonstrates mature cooperative behavior — agents not only reach the targets but do so with optimized spatial strategy.
- Policy is likely both stable and generalized, showing low-variance decisions regardless of environment randomness.

Strategic Insight:

- The dueling architecture allows agents to separate the value of being in a state from the value of individual actions, helping them ignore noisy or non-impactful decisions.
- This decomposition gives rise to confident, discriminative action selection and high policy clarity.

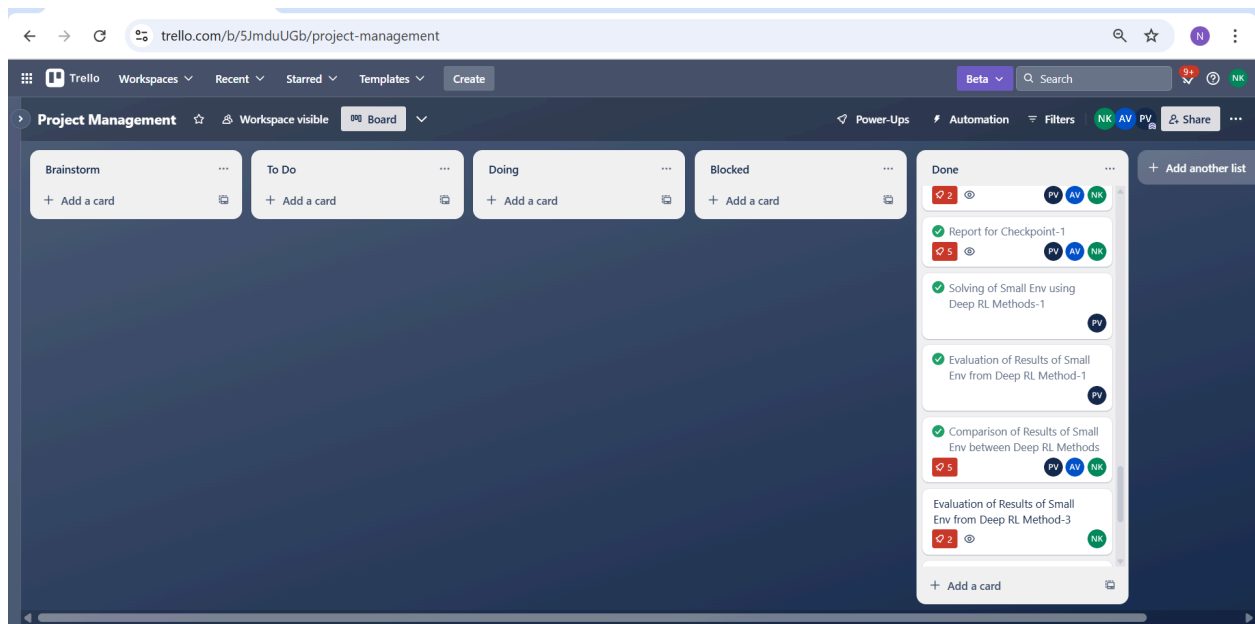
Summary: Dueling DQN exhibits the strongest form of emergent coordination — agents consistently learn roles and execute them with precision.

References

- https://pettingzoo.farama.org/environments/mpe/simple_spread/
- QMIX : <https://arxiv.org/pdf/1803.11485>
- JupyterLab: <https://github.com/jupyterlab/jupyterlab-desktop>
- VSCode: <https://code.visualstudio.com/docs/datascience/jupyter-notebooks>
- pandas: <https://pandas.pydata.org/docs/>
- numpy: <https://numpy.org/doc/stable/>
- matplotlib: <https://matplotlib.org/stable/index.html>
- Github : <https://github.com/>
- GeeksforGeeks : <https://www.geeksforgeeks.org/>
- Stackoverflow : <https://stackoverflow.com/>
- Towards Data science : <https://towardsdatascience.com/>
- Hugging Face : <https://huggingface.co/learn/deep-rl-course/en/unit0/introduction>
- RL_hand book : spring25_RL_Handbook.docx
- https://www.gymnasium.dev/environments/classic_control/cart_pole/
- https://gymnasium.farama.org/environments/box2d/lunar_lander/
- GYM_Library : <https://www.gymnasium.dev>
- RL_Book : incompleteideas.net/book/RLbook2020.pdf
- Lecture slides

Project management tool link


<https://trello.com/b/5JmduUGb/project-management>





Git Repository


https://github.com/AishwaryaVirigineni/RL_Project.git


Commits



 main


 All users



 All time


 Commits on May 3, 2025



Final Files
 Aishwarya Virigineni authored and Aishwarya Virigineni committed 2 minutes ago


b2e17fa  


Add files via upload
 AishwaryaVirigineni authored 14 minutes ago



Verified 98dda86  


Add files via upload
 AishwaryaVirigineni authored 14 minutes ago



Verified 99d4c92  

 Commits on Mar 7, 2025

Add files via upload
 AishwaryaVirigineni authored on Mar 7

Verified 8872797  

Create README.md
 prajesh-1003 authored on Mar 7

Verified 396a837  

Contribution of team mates

The work in this project was distributed equally among all members. Members were fully involved in coding, tuning hyperparameters, experiment executions, plot generations, results analysis, and final report writing. The decisions were made collectively to have equal contributions of all participants.

Aishwarya Virigineni	Nithya Kaandru	Prajesh Gupta Vizzapu
33.33%	33.33%	33.33%