PHENIKAA UNIVERSITY

FACULTY OF VEHICLE AND ENERGY ENGINEERING

AUTOMOTIVE ENGINEERING CAPSTONE DESIGN

ADVANCED REINFORCEMENT LEARNING COURSE

Multi-agent Delivery

STUDENTS:

Hạ Huy Thiện 22014474

Advisor: Vũ Hoàng Diệu

Date :27/05/2025

AUTOMOTIVE ENGINEERING CAPSTONE DESIGN COURSE REPORT

Team
Hạ Huy Thiện - ID : 22014474

Advisor : Vũ Hoàng Diệu ***Sponsor***
Faculty name : EEE **Contact name [If available]**

Abstract

This capstone project presents the design and development of a multi-agent delivery system using Deep Reinforcement Learning (DRL), aimed at simulating efficient autonomous package delivery in complex environments. With the rapid advancement of intelligent transportation and logistics systems, coordinating multiple delivery agents (robots) to operate collaboratively in dynamic environments is a key challenge that this project addresses.The system models a grid-based environment where each autonomous agent is tasked with picking up and delivering packages while avoiding collisions, minimizing delivery time, and optimizing route efficiency. We implemented a centralized training–decentralized execution (CTDE) approach using Deep Q-Networks (DQN) for each agent. A custom simulation environment was developed using Pygame to visualize agent movements, delivery progress, and real-time learning behavior.Preliminary results show that the agents learn to navigate efficiently, avoid obstacles, and complete deliveries with increasing success rates over training episodes. The system demonstrates scalability with up to 5 agents and maintains coordination even as delivery points and obstacles vary randomly. Remaining tasks include integrating more realistic urban traffic elements, optimizing reward functions, and extending the framework for real-time robotic deployment.This report details the project objectives, system architecture, DRL methodology, simulation outcomes, and future enhancements for smart logistics applications.

Contents

List of Figures

List of Tables

Chapter 1. INTRODUCTION

1.1.Problem situation

The project aims to develop a multi-agent delivery simulation system operating in a 2D grid environment or a simple urban map, where autonomous robots (agents) can learn to plan optimal routes, avoid collisions, and efficiently complete delivery tasks by applying Reinforcement Learning (RL) algorithms.

Instead of programming fixed-rule behaviors, each agent will be trained to learn from experience through interaction with the environment. The goal is to optimize delivery time, minimize conflicts, and use resources efficiently. Each agent must adapt its strategy not only based on the environmental state but also considering the behaviors of other agents, making this a Multi-Agent Reinforcement Learning (MARL) problem.

The simulation system will be built using the Pygame library to visualize the training process, navigation, and coordination among robots in an environment with multiple orders, obstacles, and dynamic, randomly changing conditions.

1.2.Problem definition

The primary goal of this project is to design and develop a scalable and visually interpretable multi-agent delivery simulation platform. This platform will simulate a dynamic 2D environment populated with multiple autonomous delivery agents, orders, and obstacles. Reinforcement Learning (RL) techniques will be employed to enable agents to learn effective strategies for pathfinding, collision avoidance, and timely delivery execution.

The system is designed to address the complexities of multi-agent coordination under conditions that involve dynamic order generation, obstacle interference, and stochastic environmental changes. Each agent is expected to optimize its delivery behavior over time by learning from both environmental feedback and the observable behaviors of other agents.

Moreover, the project explores the use of Multi-Agent Reinforcement Learning (MARL) algorithms to enhance cooperative decision-making among agents. By simulating shared environments with multiple goals and limited resources, the system aims to provide insights into collaborative learning strategies. Ultimately, the simulation platform can serve as a foundation for research in intelligent logistics, autonomous robotics, and real-world delivery systems.

Chapter 2. LITERATURE REVIEW

2.1. Project Background

Reinforcement Learning (RL) is a type of machine learning where an agent learns to make optimal decisions by interacting with an environment. At each step, the agent observes the current state, takes an action, receives a reward, and transitions to a new state. The goal is to learn a policy that maximizes cumulative rewards over time. Key components of RL include the agent, environment, state, action, reward, and policy. Unlike supervised learning, RL does not rely on labeled data but learns through trial and error. It is widely applied in robotics, game playing, autonomous vehicles, and multi-agent coordination, where learning from sequential decisions is crucial.

2.2 DQN

The Deep Q-Network (DQN) algorithm is a reinforcement learning method that combines Q-learning with deep neural networks to enable agents to learn optimal actions in high-dimensional or complex environments. Instead of using a Q-table, which becomes impractical when the state space is large, DQN uses a neural network to approximate the Q-value function Q(s,a), which estimates the expected future rewards for taking action aaa in state s.

In the Multi-Agent Delivery problem, each agent (delivery robot) uses a DQN as its "brain" to learn how to make decisions at each step. The DQN takes as input the agent's current state, including its position, destination, and information about the surrounding environment (such as obstacles and other agents), and outputs an optimal action—moving up, down, left, right, or staying still. The agent follows an ε-greedy policy to select actions, executes the chosen action, receives a reward from the environment, and stores the experience in memory for training. The DQN helps the agent estimate the Q-value Q(s,a) and gradually improves its decision-making ability over time. Depending on the design, each agent may use a separate DQN to learn its own policy, or they may share a common Q-network to learn a coordinated policy across agents.

2.2 Double DQN

Double Deep Q-Network (Double DQN) is an improved version of DQN that reduces overestimation of Q-values by separating action selection and evaluation. The online network selects the best action, while the target network evaluates its value, resulting in more stable learning.In the Multi-Agent Delivery problem, each robot uses a Double DQN to make decisions based on its current state (position, goal, obstacles, other agents). The network outputs the optimal action (move up, down, left, right, or stay).Agents follow an ε-greedy policy, store experiences, and train using replay memory. Double DQN helps agents learn more accurately and enables better coordination if they share a common network.

2.3.Multi – Agent

Multi-Agent Reinforcement Learning (MARL) is an extension of traditional Reinforcement Learning (RL) where multiple agents learn and interact within a shared environment. Each agent independently observes its local state, selects actions, and receives rewards based on its behavior and the collective dynamics of the system. Unlike single-agent RL, MARL presents unique challenges such as non-stationarity, coordination, and competition due to the presence of other learning agents. Agents must adapt their policies not only to the environment but also to the evolving strategies of others. MARL techniques are commonly used in applications like autonomous driving, robot collaboration, traffic control, and smart logistics, where multiple decision-makers must learn to cooperate or compete to achieve individual or shared goals.

Chapter 3. SYSTEM DESIGN

3.1. Description of the environment (grid, obstacles, delivery points, agents)

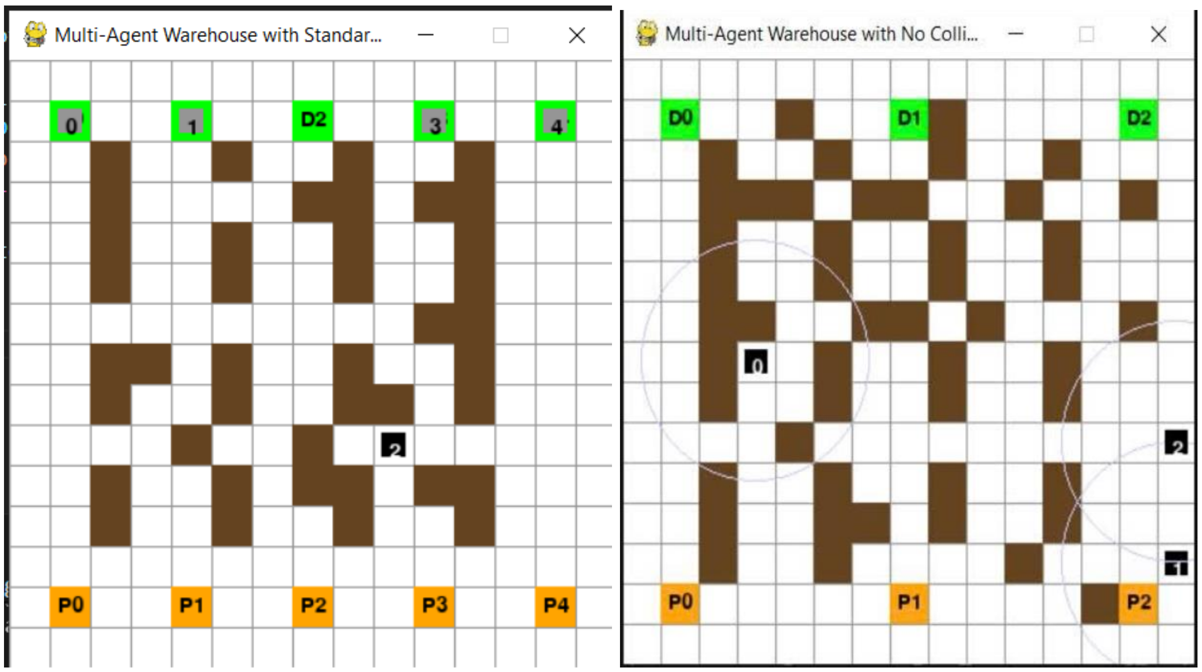

Multi-Agent Warehouse with Standar...
0
1
D2
3
4
2
P0
P1
P2
P3
P4
Multi-Agent Warehouse with No Colli...
D0
D1
D2
0
2
1
P0
P1
P2

Figure 1: Map lever 1 and lever 2 with 3,5 agent

The simulation environment is a 2D grid designed for the multi-agent delivery problem. It consists of a square space divided into cells where agents can move in four directions: up, down, left, and right. Brown cells represent obstacles (walls) that agents cannot pass through, creating navigation challenges. Pickup points, shown as orange cells labeled P1, P2, P3, etc., are where agents begin their tasks by collecting items. Delivery points, marked as green cells with labels D1, D2, D3, etc., indicate the destinations for the deliveries. Agents are displayed as black cells or cells with numeric identifiers such as "1", "2", and "3". Each agent has a visible perception area shown as a faint circle, which allows it to sense its surroundings within a limited radius. Agents move one step at a time and learn to optimize their routes from pickup to delivery points while avoiding obstacles, preventing collisions, and minimizing delivery time.

3.2. Agent modeling

STATE: The state of the agent includes the agent's current position normalized according to the environment grid size, represented as coordinates (x, y), along with the normalized positions of the pickup point and the delivery point. Additionally, the state indicates whether the agent has already picked up the package through a boolean variable has_package (True or False). The distance between the agent and the current target (either the pickup or delivery point, depending on whether the package has been picked up) is also calculated and included in the state. To allow the agent to observe its nearby surroundings, a local GRID_SIZE X CELL_SIZE grid map representing obstacles and paths around the agent is incorporated as a matrix. Finally, if there are multiple agents in the environment, the state also contains information about other agents within communication range to facilitate effective coordination in the shared task.

ACTIONS:

The agent has five possible actions:

0: Stay still (no movement).

1: Move up.

2: Move down.

3: Move left.

4: Move right.

REWARD : The reward system is designed as follows: The agent receives +20 points upon successfully reaching the pickup point and +50 points upon successful delivery. Additionally, the reward is proportional to the distance reduced when moving closer to the current target (pickup or delivery point). Other agents within communication range near the delivery point also receive a secondary reward up to +5 points, decreasing with distance to that point. Each movement step incurs a penalty of -0.01 points to encourage faster task completion. If the agent attempts to move but fails (due to collision or prolonged standing still), a penalty of -0.1 points is applied. Finally, a bonus reward is given if all agents complete their tasks before the step limit, based on the remaining steps.

Chapter 4: Algorithm & Learning Strategy

4.1 DQN.

Deep Q-Network (DQN) algorithm is used to train agents navigating a 15x15 warehouse environment with the goal of efficiently picking up and delivering packages. Each agent uses a deep neural network consisting of three fully connected layers with ReLU activations to approximate the Q-value function Q(s,a)Q(s, a)Q(s,a), which predicts Q-values for five possible actions (stay, up, down, left, right) based on a state that includes the agent's coordinates, pickup/delivery locations, package status, and a 5x5 local map.

DQN leverages **experience replay** with a buffer size of 10,000 to store experiences (s,a,r,s′,d), allowing random sampling to break temporal correlations. A **target network** is maintained to compute target Q-values, which is updated every 50 steps to stabilize learning. The target Q-value is computed as follows:

$$Q_{\text{target}}(s,a) = r + \gamma \max_{a'} Q_{\text{target}}(s',a')$$

where rrr is the reward, γ=0.9 is the discount factor, and Qtarget(s′,a′) is the Q-value from the target network for the next state s′.

The loss function is defined as the mean squared error:

$$\text{Loss} = \mathbb{E}\left[\left(Q_{\text{target}}(s,a) - Q(s,a)\right)^2\right]$$

Quá trình tối ưu sử dụng thuật toán Adam với tốc độ học là 0.001. Chiến lược **epsilon-greedy** được áp dụng, với ε\varepsilonε giảm dần từ 0.9 xuống 0.05 để cân bằng giữa khám phá và khai thác. Phần thưởng được thiết kế nhằm khuyến khích nhặt hàng (+20), giao hàng (+50), phạt các hành động sai (-0.1) và giảm chi phí thời gian mỗi bước (-0.01).

4.2 DDQN

In the proposed program, the **Double Deep Q-Network (DDQN)** algorithm is applied to train agents navigating a 15x15 warehouse environment, enabling them to efficiently pick up and deliver packages. Each agent employs a deep neural network consisting of two input streams—one for the 5x5 local map and another for non-spatial state features. These streams are followed by fully connected layers with ReLU activations to approximate the Q-value function Q(s,a), which predicts Q-values for five discrete actions (stay, up, down, left, right). The state input includes the agent's current position, pickup and delivery point indicators, package carrying status, local 5x5 grid observations, and communication data from nearby agents.

DDQN incorporates an **experience replay** buffer of size 10,000, which stores transitions (s, a, r, s', d), allowing mini-batch sampling to decorrelate training data and improve stability. A **target network** is used to generate more stable Q-value targets and is updated every 50 steps by copying weights from the main network. Unlike standard DQN, which tends to overestimate Q-values by using the same network for both action selection and evaluation, DDQN alleviates this bias by separating these steps. Specifically, it selects the action using the main network and evaluates it using the target network:

$$Q_{\text{target}}(s,a) = r + \gamma Q_{\text{target}}\left(s', \arg\max_{a'} Q(s',a')\right)$$

where r is the reward, γ=0.95 is the discount factor, and Qtarget(s′,arg max a′Q(s′,a′))represents the target network's evaluation of the action selected by the main network in the next state s′.

The loss function used is the mean squared error between the predicted Q-value and the target:

$$\text{Loss} = \mathbb{E}\left[\left(Q_{\text{target}}(s,a) - Q(s,a)\right)^2\right]$$

The network is trained using the Adam optimizer with a learning rate of 0.001. An **epsilon-greedy** policy is adopted for exploration, with ε\varepsilonε decaying linearly from 0.9 to 0.05 to encourage gradual exploitation over time. The reward structure is carefully designed: agents receive +20 for successful pickups, +50 for successful deliveries, -0.1 for invalid actions, and -0.01 per time step to discourage idling. Additionally, **proximity-based** bonus rewards are awarded to encourage collaborative behavior among agents.

Chapter 5. Experiments

5.1 Training Parameters

The training process is governed by a set of carefully chosen hyperparameters that directly influence the performance and stability of the learning agents. The environment is structured as a 15×15 grid (`GRID_SIZE = 15`), where each cell measures 25 pixels (`CELL_SIZE = 25`), resulting in a visual display size of 375×375 pixels. This grid defines the spatial resolution of the simulation and constrains agent movement.

A total of five agents (`NUM_AGENTS = 5`) operate in this environment, learning concurrently across 1000 episodes (`EPISODES = 1000`). Each episode allows up to 1000 interaction steps (`MAX_STEPS = 1000`), providing sufficient exploration opportunities per episode.

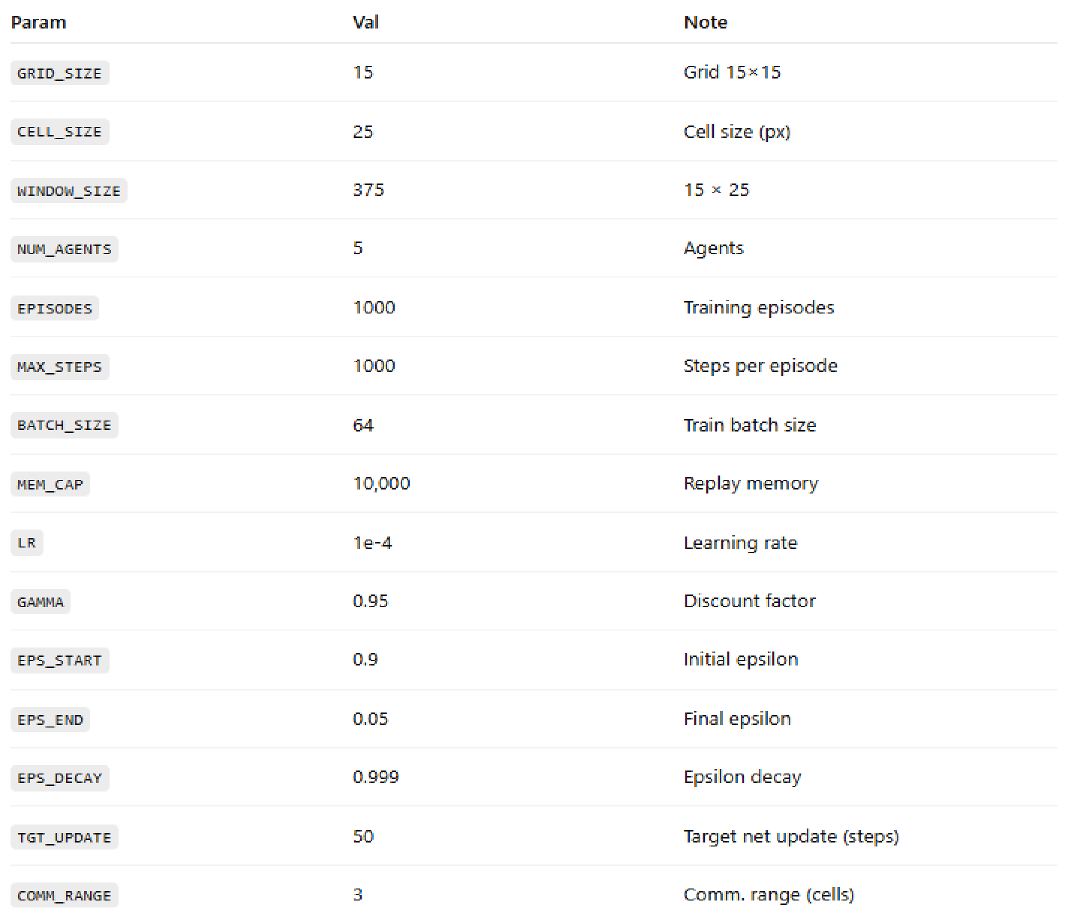
To ensure effective learning, agents are trained using mini-batches of 64 samples (`BATCH_SIZE = 64`) drawn from a replay memory of capacity 10,000 (`MEMORY_CAPACITY = 10000`). This replay buffer helps in stabilizing learning by breaking the temporal correlations between consecutive experiences.

The model employs a relatively small learning rate (`LEARNING_RATE = 1e-4`) to allow gradual and stable updates to the neural network weights. A discount factor γ of 0.95 (`GAMMA = 0.95`) is used to balance the importance of immediate versus future rewards, encouraging long-term strategy development.

Exploration is handled via an epsilon-greedy policy, beginning with a high exploration rate (`EPSILON_START = 0.9`) and gradually decaying to a minimal level (`EPSILON_END = 0.05`) with a decay rate of 0.999 (`EPSILON_DECAY = 0.999`). This ensures a shift from exploration to exploitation over time, which is essential for converging on optimal policies.

The target network — a stabilized version of the Q-network — is updated every 50 steps (`TARGET_UPDATE_FREQ = 50`) to improve learning consistency. Furthermore, inter-agent communication is allowed within a 3-cell radius (`COMMUNICATION_RANGE = 3`), fostering cooperation and collective behavior among agents.

Table 1 : Parameters



Param	Val	Note
`GRID_SIZE`	15	Grid 15×15
`CELL_SIZE`	25	Cell size (px)
`WINDOW_SIZE`	375	15 × 25
`NUM_AGENTS`	5	Agents
`EPISODES`	1000	Training episodes
`MAX_STEPS`	1000	Steps per episode
`BATCH_SIZE`	64	Train batch size
`MEM_CAP`	10,000	Replay memory
`LR`	1e-4	Learning rate
`GAMMA`	0.95	Discount factor
`EPS_START`	0.9	Initial epsilon
`EPS_END`	0.05	Final epsilon
`EPS_DECAY`	0.999	Epsilon decay
`TGT_UPDATE`	50	Target net update (steps)
`COMM_RANGE`	3	Comm. range (cells)

5.2 Results Visualization

5.2.1 Results DQN

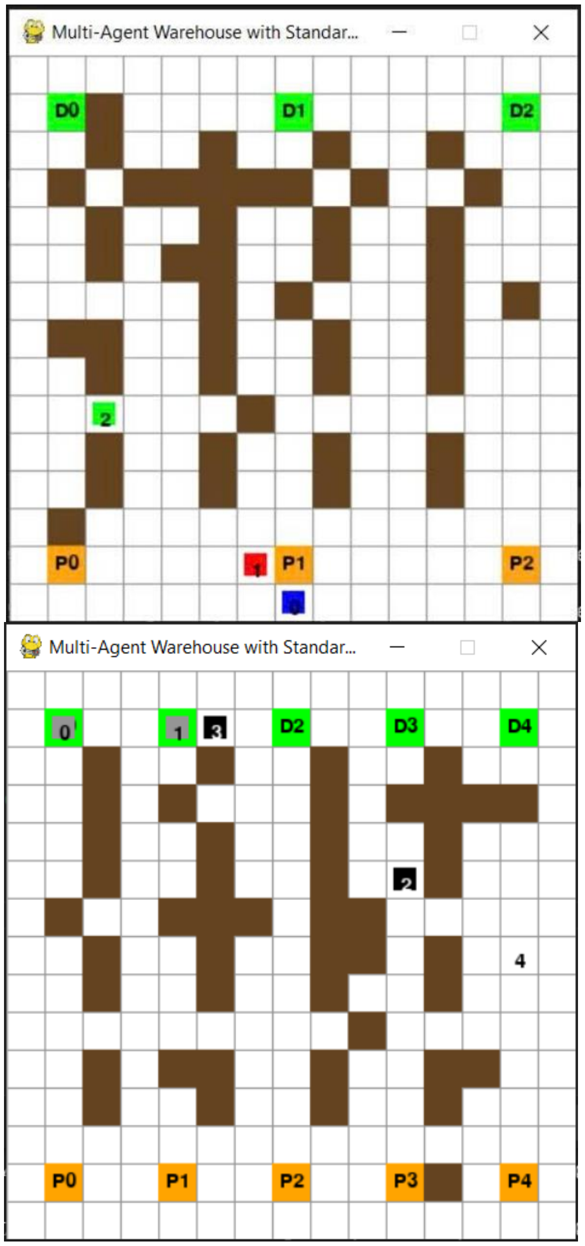
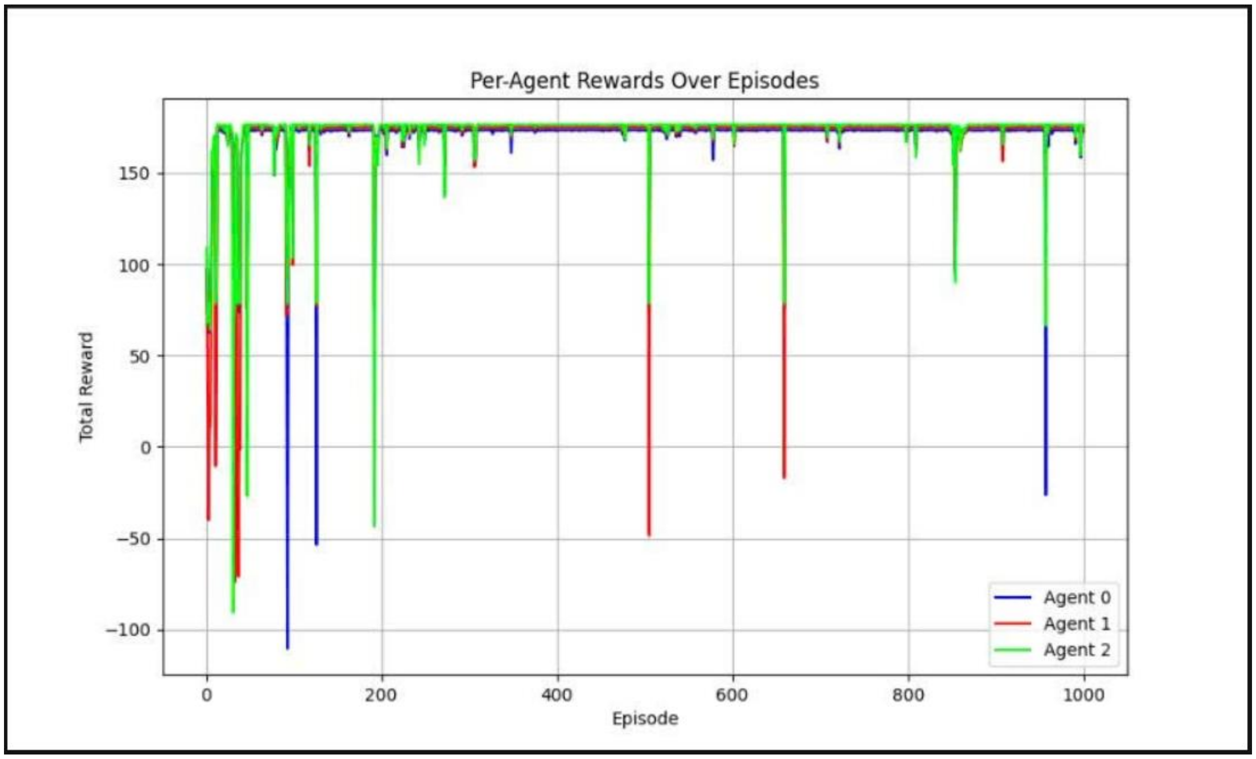

Multi-Agent Warehouse with Standar...
D0
D1
D2
2
P0
1
P1
P2
Multi-Agent Warehouse with Standar...
0
1
3
D2
D3
D4
2
4
P0
P1
P2
P3
P4

Figure : 2 The map includes three agents and five agents(DQN)

Figure 2: The simulation shows a delivery environment where brown tiles represent walls or obstacles, yellow tiles indicate pickup points, green tiles mark delivery destinations, and agents are deployed (three agents in the first map, and five agents in the second map).



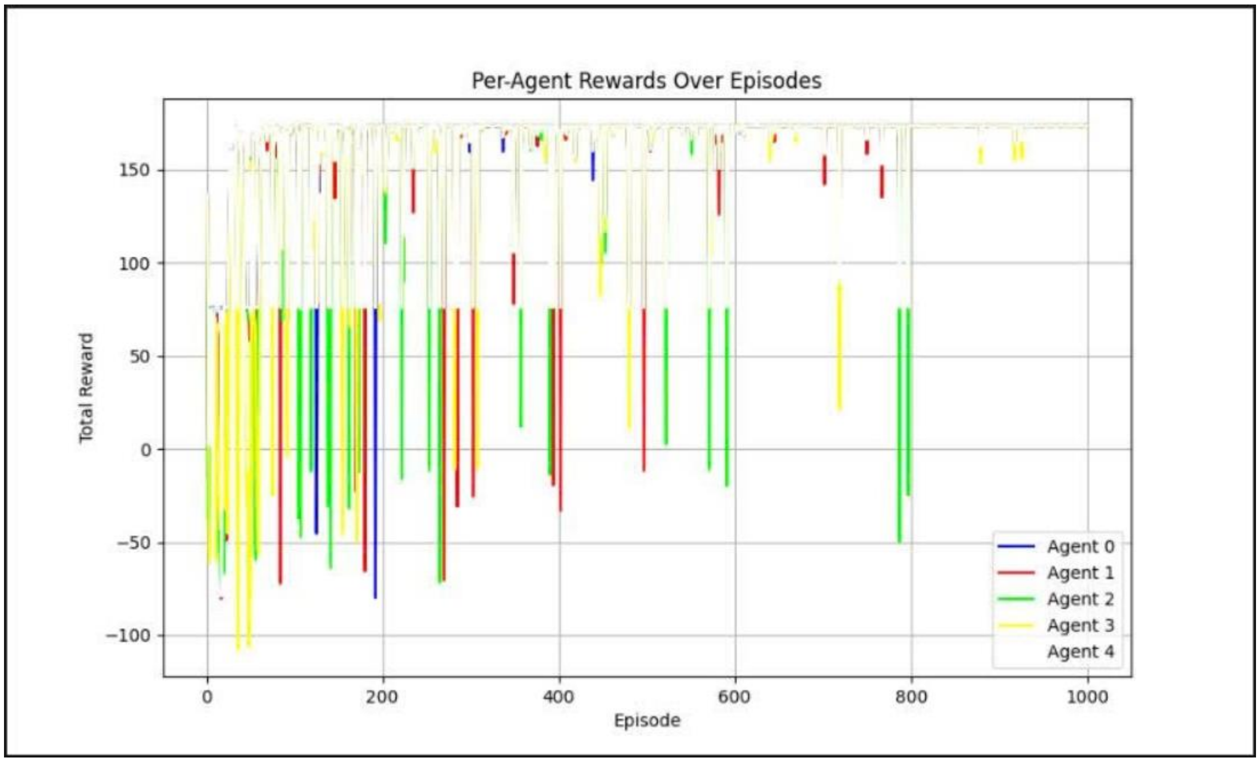


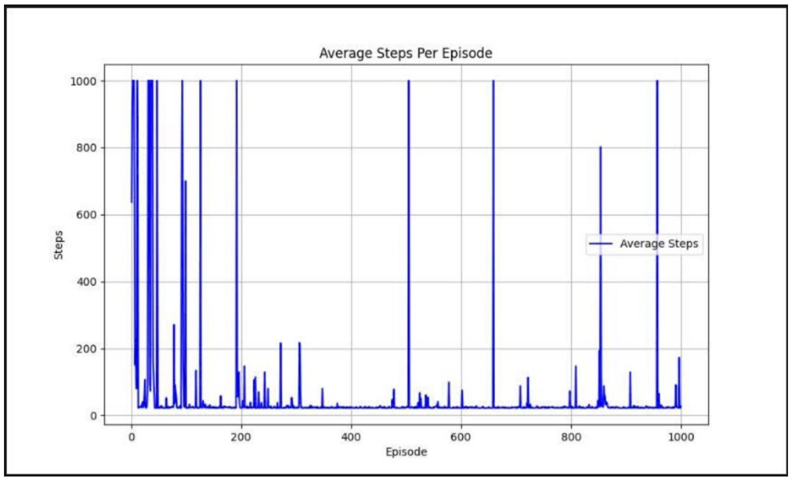
Figure : 3 Agent Reward over Episodes (DQN)

Figure 3 illustrates the total reward received by each agent over 1000 episodes in two different scenarios: one with three agents and the other with five agents.

In the first chart (three agents), Agent 2 shows a stable and rapid learning curve, achieving high and consistent rewards early, around episode 50. Meanwhile, Agent 0 and Agent 1 exhibit significant fluctuations in the early episodes and gradually improve over time, although occasional drops still occur, indicating unstable learning behavior.

In the second chart (five agents), the reward trajectories are more chaotic, especially during the first 200 episodes. This reflects the increased complexity and interaction among agents in a more crowded environment. Agent 3 and Agent 4 generally perform better and reach higher rewards faster than the others. However, some agents, particularly Agent 0 and Agent 2, experience frequent negative rewards or remain inconsistent throughout the training.

Overall, as the number of agents increases, the learning environment becomes more competitive and unstable, which leads to a longer convergence time and uneven learning progress among agents. This highlights the challenges of scaling reinforcement learning in multi-agent systems.



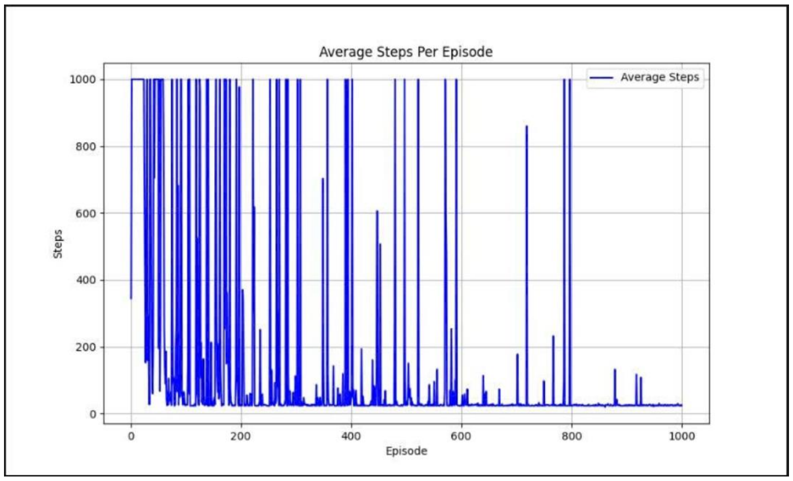
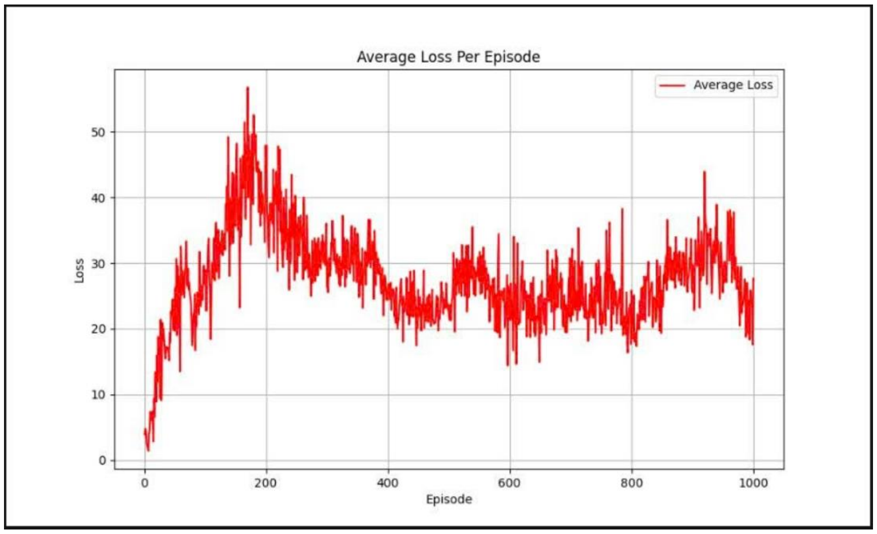


Figure : 4 Average Step Per Esisode (DQN)

The top chart (3 agents) shows that the agents learn relatively quickly, as the average number of steps drops significantly after around 100 episodes and stabilizes at a low level. This indicates an efficient learning process.

In contrast, the bottom chart (5 agents) displays greater fluctuations and takes much longer to stabilize (around 700 episodes), suggesting that coordination among a larger number of agents is more complex and slows down learning.

Conclusion: As the number of agents increases, the learning process becomes more challenging and unstable. However, with sufficient training episodes, the system still manages to achieve effective performance.



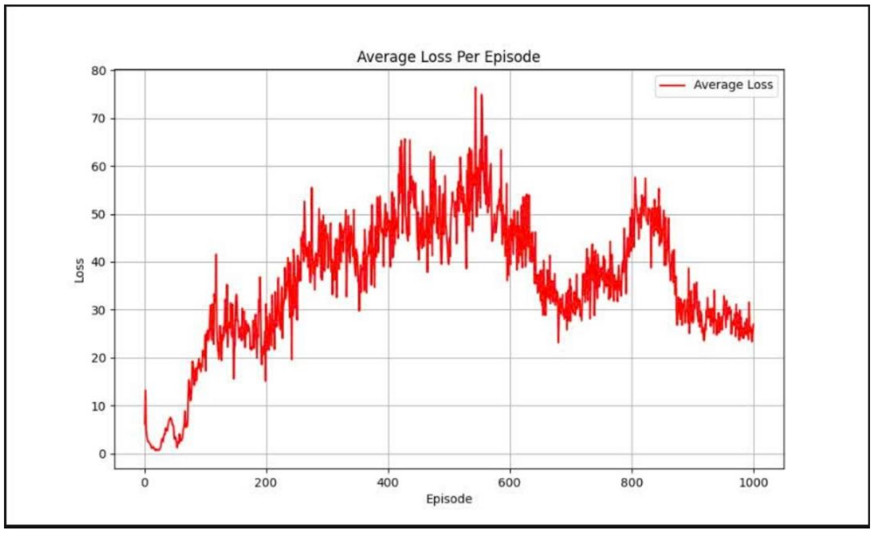


Figure : 5 Average loss Per Esisode(DQN)

The top chart (3 agents) shows a rapid increase in loss at the beginning, followed by moderate fluctuations and a slight decrease toward the end, indicating a relatively stable learning process.

The bottom chart (5 agents) shows higher and more volatile loss, especially between episodes 200–600. However, the loss decreases toward the end, suggesting effective learning is still achieved.

Conclusion: With more agents, training becomes more complex and unstable, but the model still shows signs of convergence over time.

For the case of 3 agents, the DQN algorithm demonstrates effective learning. The average number of steps decreases rapidly after about 100 episodes and remains low, while the loss value fluctuates slightly and shows a downward trend. This reflects a stable and efficient training process.

In contrast, with 5 agents, the number of steps fluctuates significantly and takes about 700 episodes to stabilize. The loss value also varies more, especially during the middle phase of training. This indicates that as the number of agents increases, the state space becomes more complex, making the learning process more difficult and slower.

In summary, the DQN algorithm performs well with a moderate number of agents. When the number of agents increases, the learning process becomes less stable but still achieves effectiveness after sufficient training episodes.

5.2.2 Results DDQN

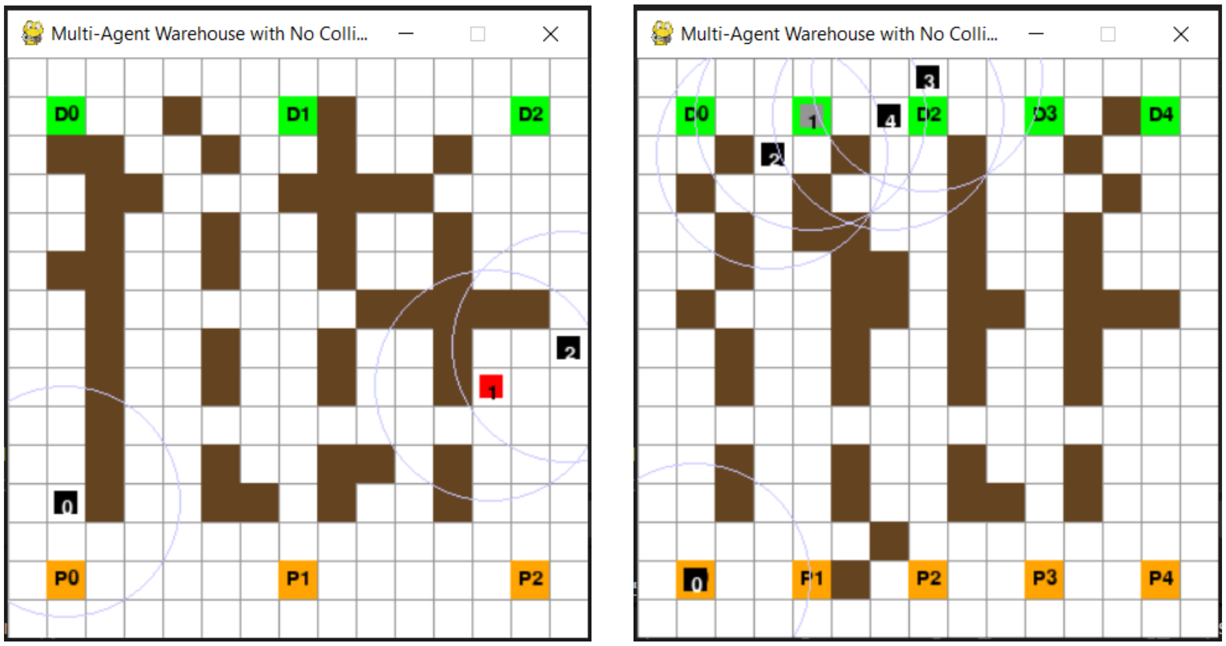
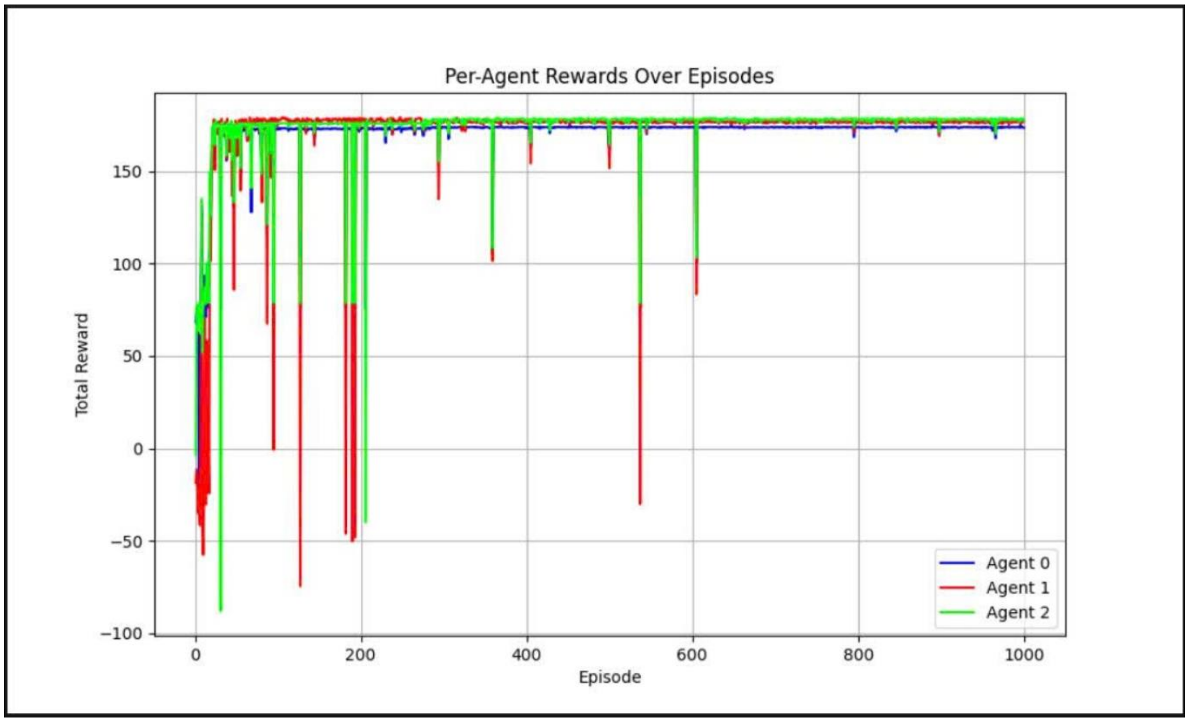

Multi-Agent Warehouse with No Colli...
D0
D1
D2
2
1
0
P0
P1
P2
Multi-Agent Warehouse with No Colli...
3
D0
1
4
D2
D3
D4
2
0
P1
P2
P3
P4

Figure 6: The map includes three agents and five agents(DDQN)

Figure 5 illustrates the environment map with obstacles (brown), pickup points (yellow), and delivery points (green). As the number of agents increases from 3 to 5, the movement space becomes more constrained and complex due to the density of obstacles and the distribution of pickup and delivery points, increasing the difficulty of learning and navigation for the agents.



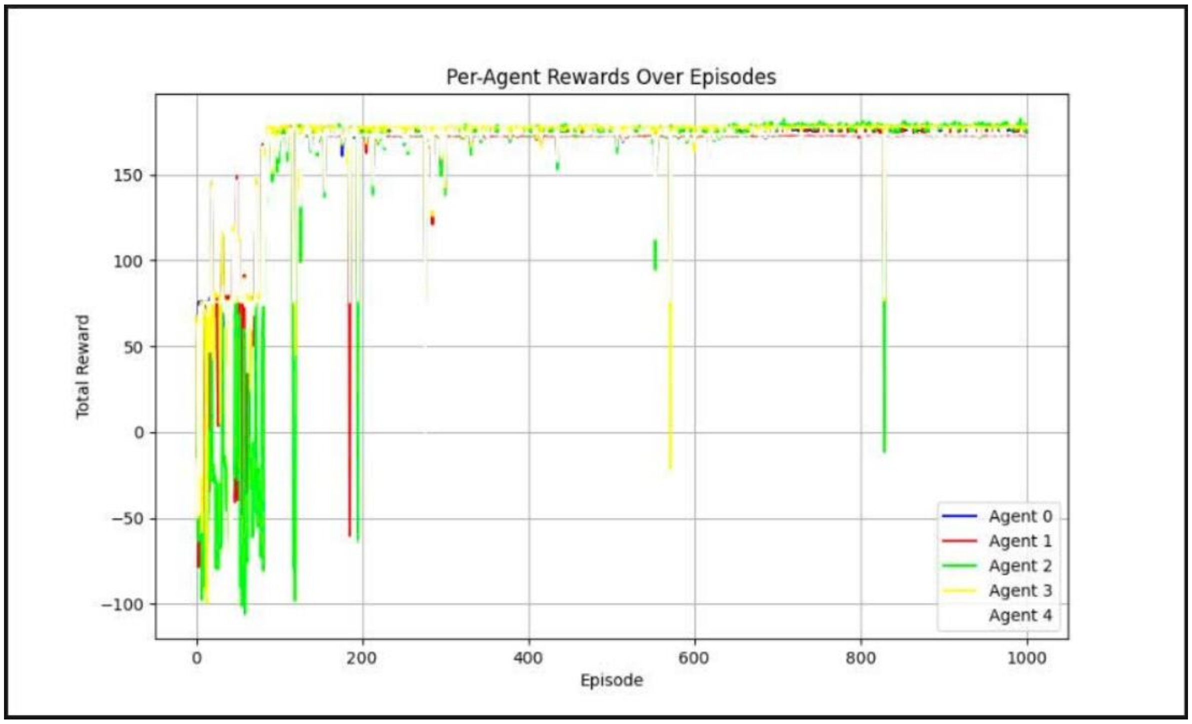
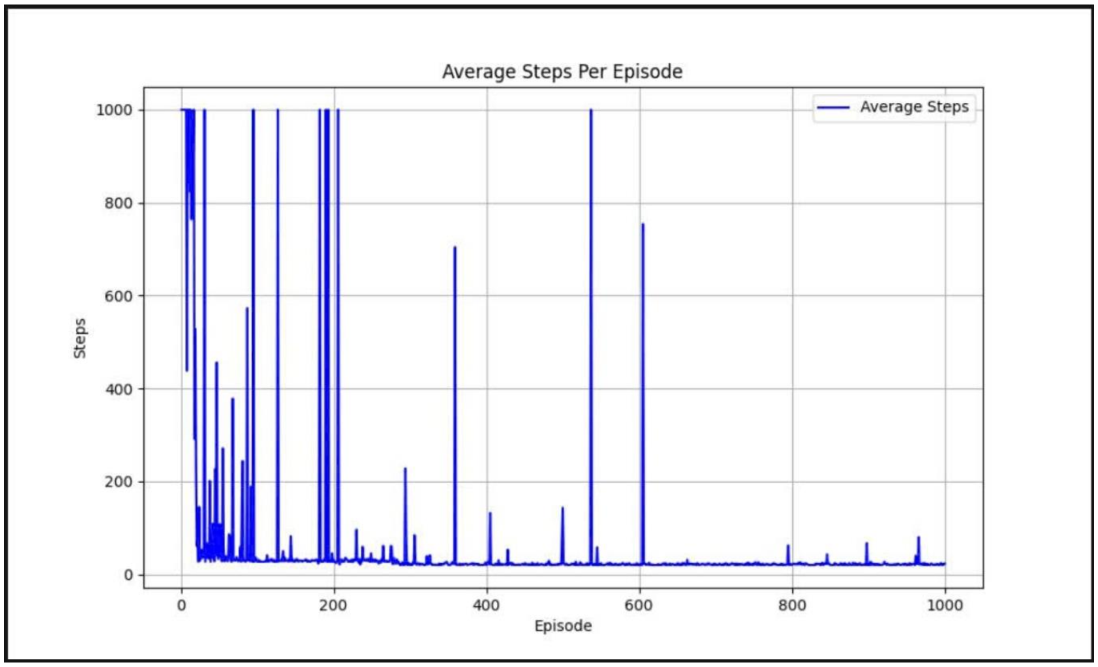


Figure 7: Agent Reward over Episodes (DDQN)

Figure 6 shows the per-agent rewards over training episodes. With 3 agents (top graph), the reward curves converge quickly after around 100 episodes, with minimal fluctuations and stable high values. In contrast, with 5 agents (bottom graph), the rewards exhibit greater fluctuations in the early stages and require about 700 episodes to stabilize. This indicates that as the number of agents increases, the learning process becomes more challenging and takes longer to achieve high performance.



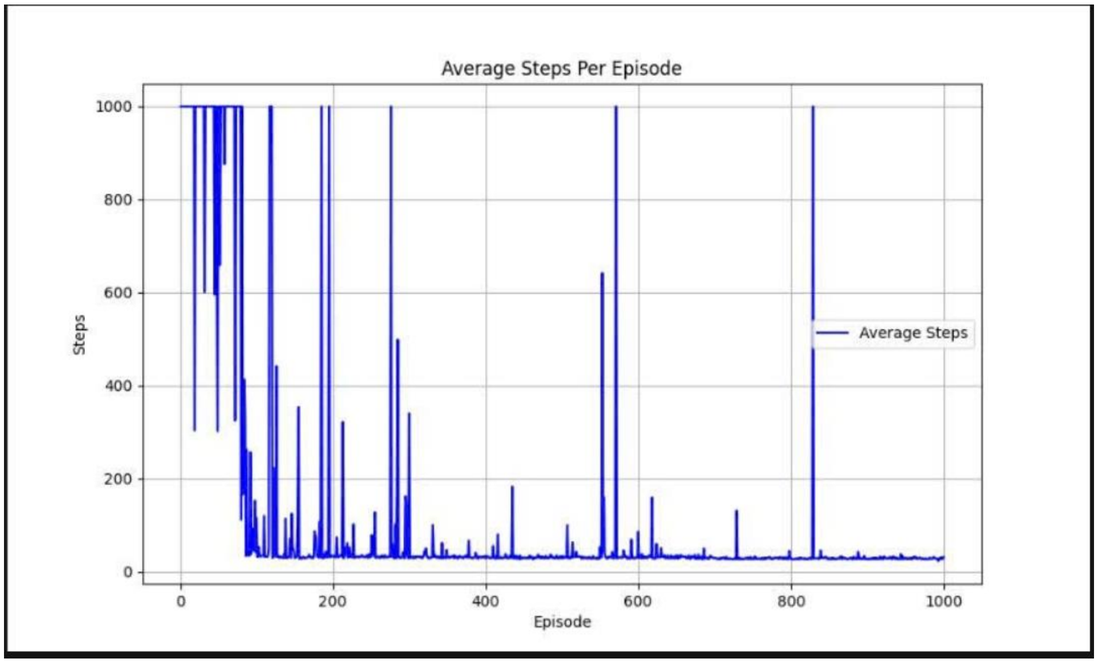
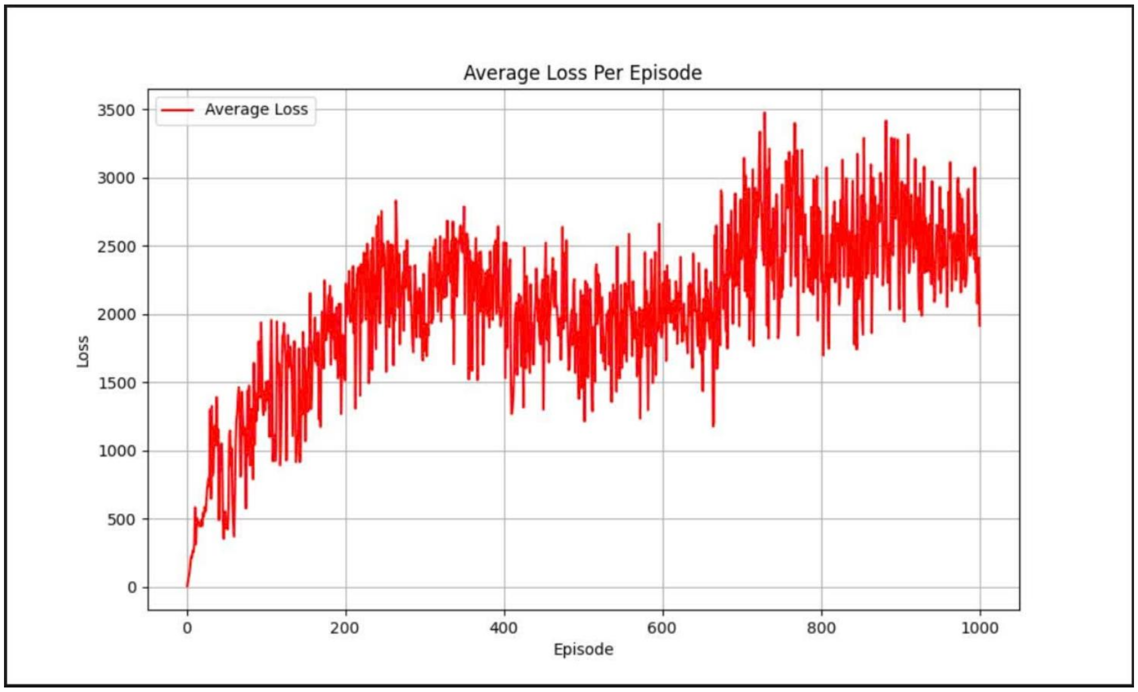


Figure : 8 Average Step Per Esisode(DDQN)

Figure 7 illustrates the average steps per episode during training. With 3 agents (top chart), the number of steps drops quickly and stabilizes below 100 steps after around 200 episodes, indicating that the agents learn to optimize their paths efficiently. In contrast, with 5 agents (bottom chart), convergence takes longer, with greater fluctuation and a slower decline, showing increased difficulty as the number of agents increases.



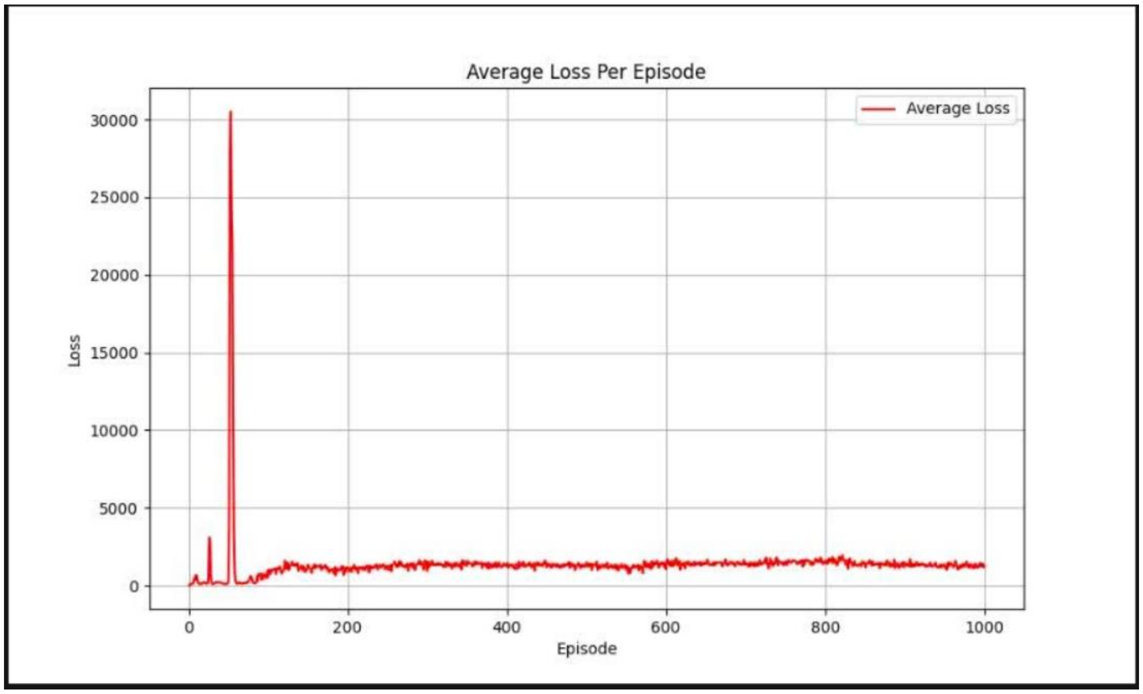


Figure : 9 Average Step Per Esisode(DDQN)

The upper chart (3 agents) shows increasing and highly fluctuating loss, indicating an unstable learning process and lack of clear convergence. In contrast, the lower chart (5 agents) has only a few high loss spikes at the beginning, then quickly stabilizes with minimal fluctuations.

This suggests that the model with 5 agents learns more effectively and converges better than the model with 3 agents, thanks to the ability to gather more information from multiple agents during training.

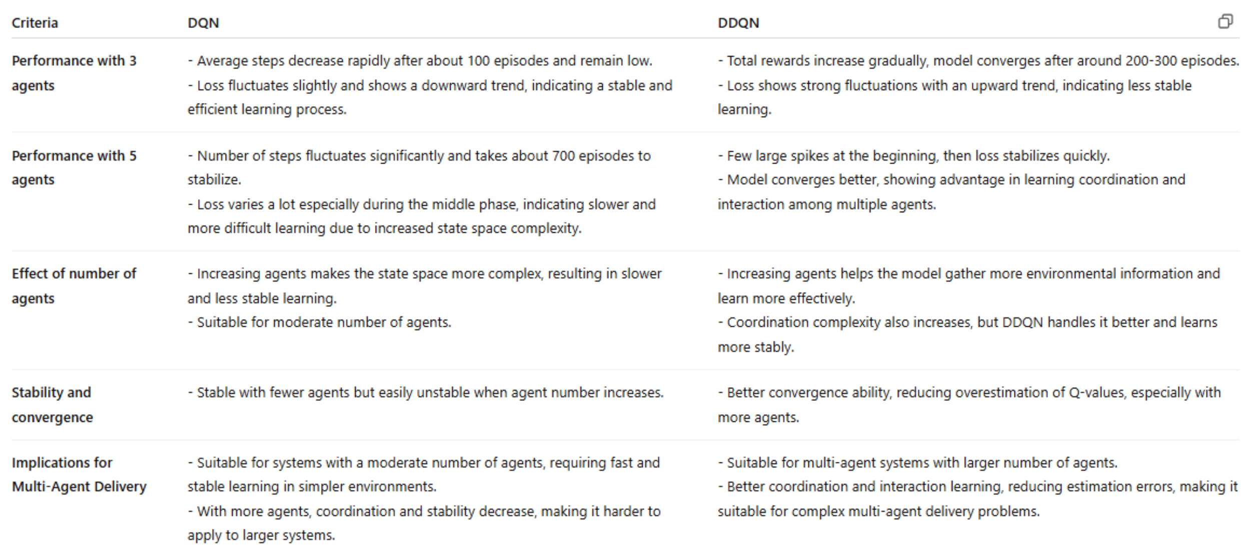
The Double Deep Q-Network (DDQN) algorithm demonstrates effectiveness in multi-agent environments. The model converges after approximately 200–300 episodes, as shown by the increasing total rewards and a significant decrease in the average number of steps per episode.

Regarding the loss function, the model with 3 agents shows strong fluctuations and an upward trend in loss, indicating an unstable learning process. In contrast, the model with 5 agents experiences only a few large spikes at the beginning, then stabilizes quickly, suggesting better convergence.

This indicates that increasing the number of agents helps the model learn more effectively by collecting more information from the environment. However, it also increases the complexity of coordination and interaction among agents.

5.2.3 Comparison between DQN and DDQN in Multi-Agent Delivery Environmen

Table 2 Comparison between DQN and DDQN in Multi-Agent Delivery Environmen



Criteria	DQN	DDQN
Performance with 3 agents	- Average steps decrease rapidly after about 100 episodes and remain low. - Loss fluctuates slightly and shows a downward trend, indicating a stable and efficient learning process.	- Total rewards increase gradually, model converges after around 200-300 episodes. - Loss shows strong fluctuations with an upward trend, indicating less stable learning.
Performance with 5 agents	- Number of steps fluctuates significantly and takes about 700 episodes to stabilize. - Loss varies a lot especially during the middle phase, indicating slower and more difficult learning due to increased state space complexity.	- Few large spikes at the beginning, then loss stabilizes quickly. - Model converges better, showing advantage in learning coordination and interaction among multiple agents.
Effect of number of agents	- Increasing agents makes the state space more complex, resulting in slower and less stable learning. - Suitable for moderate number of agents.	- Increasing agents helps the model gather more environmental information and learn more effectively. - Coordination complexity also increases, but DDQN handles it better and learns more stably.
Stability and convergence	- Stable with fewer agents but easily unstable when agent number increases.	- Better convergence ability, reducing overestimation of Q-values, especially with more agents.
Implications for Multi-Agent Delivery	- Suitable for systems with a moderate number of agents, requiring fast and stable learning in simpler environments. - With more agents, coordination and stability decrease, making it harder to apply to larger systems.	- Suitable for multi-agent systems with larger number of agents. - Better coordination and interaction learning, reducing estimation errors, making it suitable for complex multi-agent delivery problems.

DQN offers a simple and efficient learning process when applied to environments with a moderate number of agents (e.g., 3 agents), where it quickly reduces the number of steps and maintains stability. However, as the number of agents increases (e.g., 5 agents), the learning process becomes slower and less stable due to the expanded and more complex state space. In contrast, DDQN addresses the limitations of DQN by mitigating Q-value overestimation, resulting in better convergence and improved performance, especially in environments with more agents. This makes DDQN more suitable for complex multi-agent delivery tasks, where stable learning and effective coordination among agents are essential. Therefore, for projects involving a larger number of agents, DDQN is the preferred choice to ensure robust and efficient training.

Chapter 6: Conclution

Through the implementation and comparison of DQN and DDQN algorithms in a multi-agent delivery environment, I have gained valuable insights into the practical applications of reinforcement learning in complex, dynamic systems. Initially, I observed that the Deep Q-Network (DQN) performs efficiently in environments with a limited number of agents, offering fast convergence and stable learning. However, as the agent count increased, DQN exhibited instability and slower convergence due to the exponential growth in the state and action space. This led to the realization that traditional DQN has limitations in handling complex coordination and interaction among multiple agents.

On the other hand, Double DQN (DDQN) demonstrated its strength in mitigating Q-value overestimation—a known weakness of DQN—leading to more stable learning, even as the number of agents increased. The experimental results showed that DDQN not only maintained consistency in performance but also converged more reliably under high agent density, making it a more robust solution for large-scale multi-agent problems.

Through this project, I not only deepened my understanding of value-based reinforcement learning methods but also developed essential skills in designing experiments, analyzing learning curves, and interpreting performance metrics such as loss, rewards, and average steps. Furthermore, I learned how the complexity of an environment—such as agent interaction and state space size—directly impacts learning stability and convergence.

This project has strengthened my ability to choose and implement appropriate reinforcement learning algorithms based on specific problem characteristics. It also highlighted the importance of algorithmic improvements such as DDQN when scaling reinforcement learning to more challenging, real-world scenarios. Overall, this experience has significantly enriched my knowledge of multi-agent systems and has prepared me for further exploration and application of reinforcement learning in both academic and practical domains.

Tài liệu tham khảo:

[1] https://ieeexplore.ieee.org/abstract/document/4445757

[2] https://link.springer.com/book/10.1007/978-3-642-14435-6

[3] **Multi-Agent Reinforcement Learning: A Review of Challenges and Applications**

LINK YOUTUBE:

https://youtu.be/3pXNIr2ddjQ