**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.*

List of Figures

Figure 1: *Map lever 1 and lever 2 with 3 agent*. 6

Figure 2: *Map* 11

Figure 3:  *Per – Agent Reward over Episodes*  11

Figure 4: *Average Step Per Esisode*  12

Figure 5: *Average loss per Esisode*. 12

List of Tables

Table 1: Predicted Student Grade Distribution 8

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 DQN

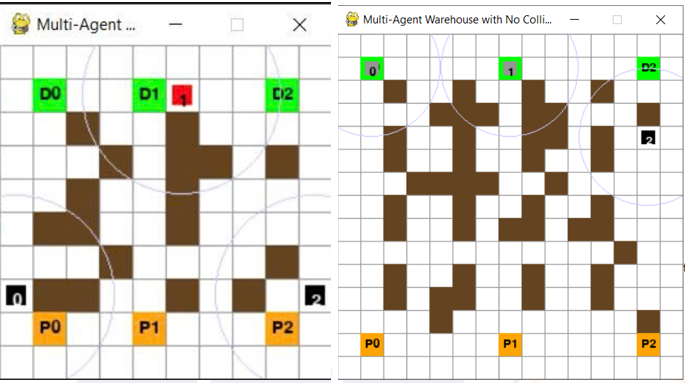
**Double Deep Q-Network (Double DQN)** is an improved version of DQN that reduces overestimation of Q-values by separating action selection and evaluation. The online network selects the best action, while the target network evaluates its value, resulting in more stable learning.In the **Multi-Agent 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.

2.4.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)



*Figure 1: Map lever 1 and lever 2 with 3 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.

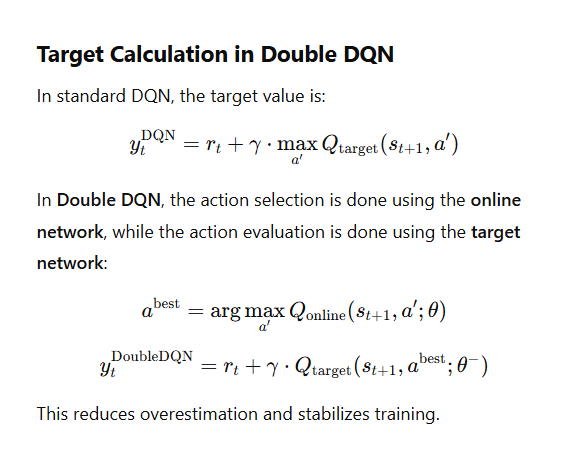
4: Move right.

**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** **Double DQN (Double Deep Q-Network)**

The original DQN algorithm tends to overestimate action values due to using the same network for both selecting and evaluating actions. Double DQN addresses this overestimation by decoupling these two roles.



**4.2Replay Buffer**

The replay buffer is a sequential memory used to store experience transitions

in the form:

(st​,at​,rt​,st+1​,done)

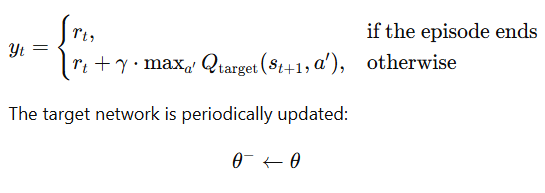
In each update step, a minibatch is randomly sampled from the buffer to reduce sequential

correlation and stabilize the training process.

**4.3 Target Network**

The target network is a temporary copy of the main network, used tcompute a more stable

target value yty\_tyt​:



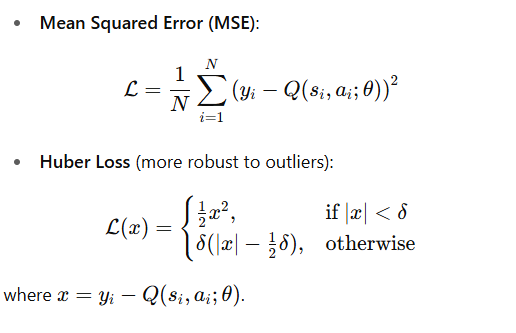
The target network is periodically updated:

θ−←θ

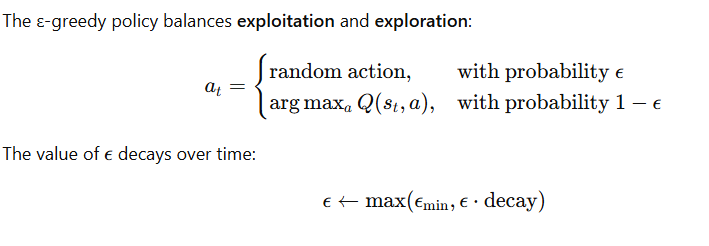
where θ represents the main network parameters and θ the target network parameters.

**4.4 Loss Function**

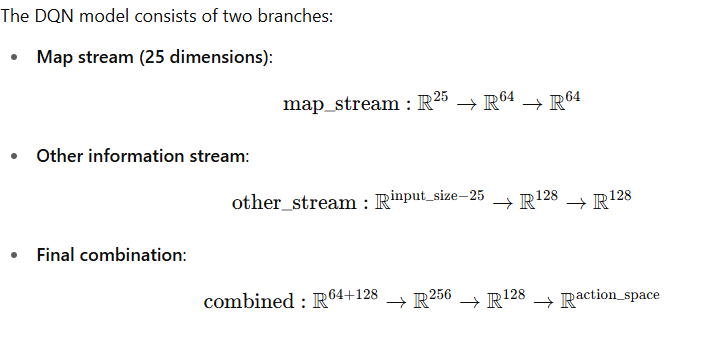
The loss function measures the deviation between the predicted Q-value and the target value:



4.5 ε-greedy Policy



4.6 DQN Network Architecture



Chapter 5. Experiments

**5.1 Training Parameters**

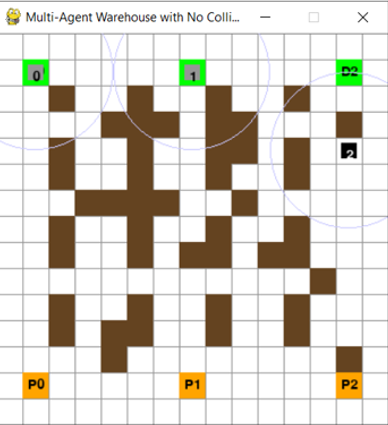
The training process uses the following hyperparameters: the environment grid has a size of 15x15 (GRID\_SIZE = 15) with each cell measuring 25 pixels (CELL\_SIZE = 25), resulting in a display window size of WINDOW\_SIZE = GRID\_SIZE \* CELL\_SIZE. The simulation involves 5 agents (NUM\_AGENTS = 5), trained over 1000 episodes (EPISODES = 1000) with a maximum of 1000 steps per episode (MAX\_STEPS = 1000). The training batch size is 64 (BATCH\_SIZE = 64), and the replay memory has a capacity of 10,000 experiences (MEMORY\_CAPACITY = 10000). The model is trained with a learning rate of 1e-4 (LEARNING\_RATE = 1e-4) and a discount factor γ = 0.95 (GAMMA = 0.95). Epsilon-greedy exploration is used with EPSILON\_START = 0.9, decreasing to EPSILON\_END = 0.05 at a decay rate of EPSILON\_DECAY = 0.999. The target network is updated every 50 steps (TARGET\_UPDATE\_FREQ = 50), and agents can communicate within a radius of 3 cells (COMMUNICATION\_RANGE = 3).

In the simulation scenario, 5 agents operate on a 15x15 grid, each assigned a delivery task that includes a pickup point and a delivery point. The map may contain obstacles, requiring agents to navigate effectively and coordinate for optimal performance.

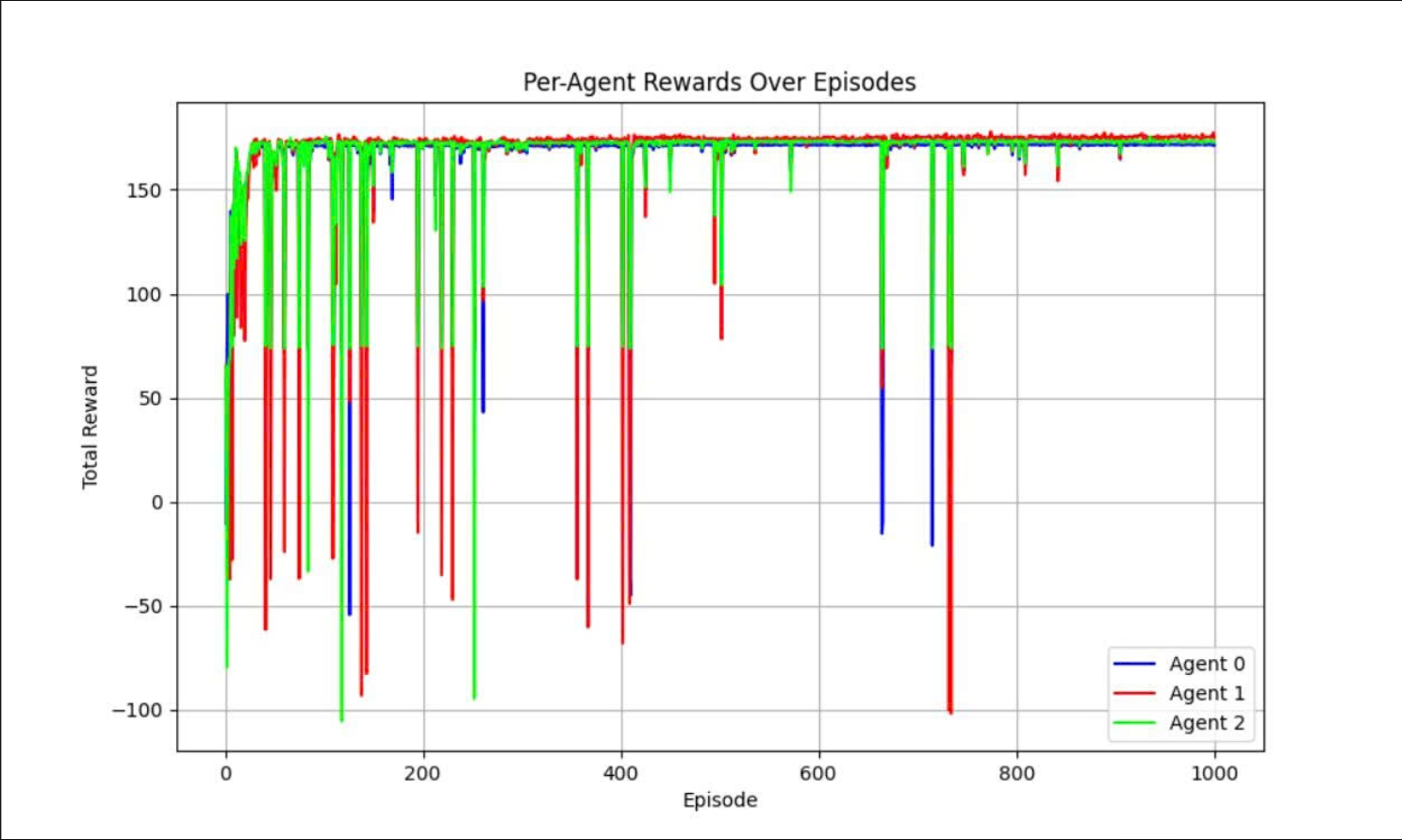
The training is conducted over 1000 episodes, each lasting up to 1000 steps. The effectiveness of the learning process is illustrated through performance graphs, including cumulative rewards per episode, collision rates among agents, and the success rate of completed deliveries over time. These visualizations provide insight into how well the agents learn to complete their tasks and avoid obstacles and each other.

**5.2 Results Visualization**

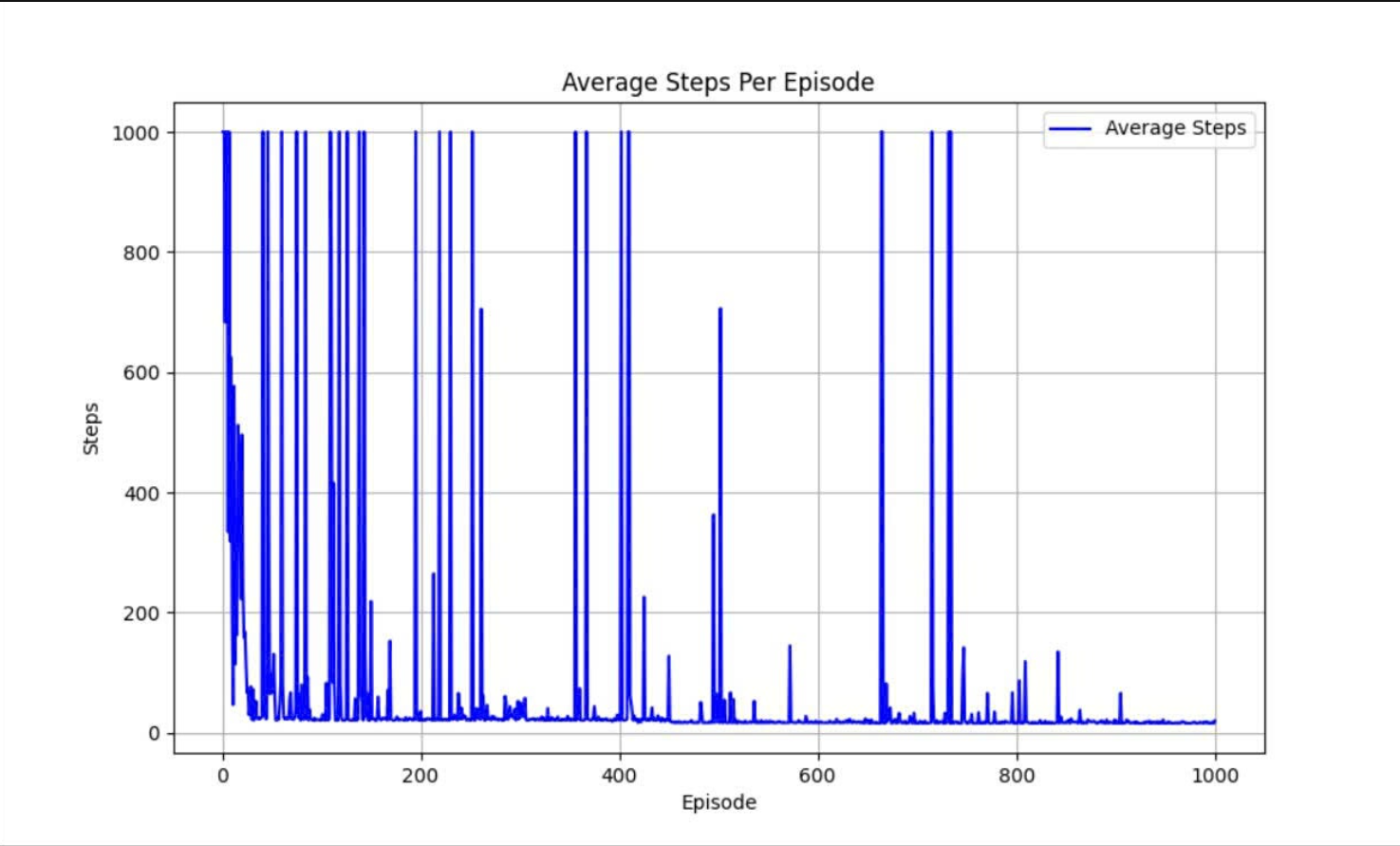
**5.2.1 With three agent 1000 episodes**

****

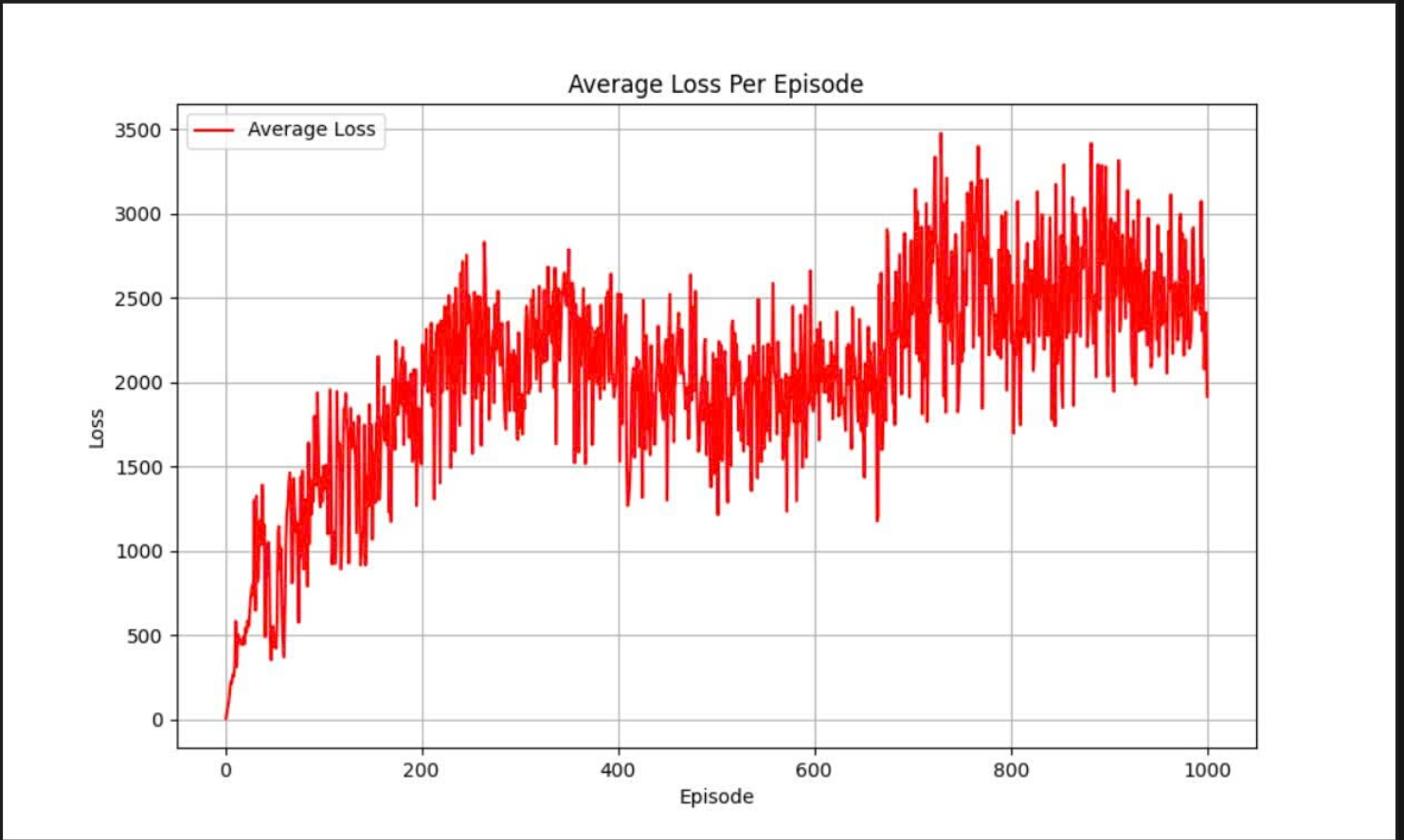
*Figure : 2 map*

****

*Figure : 3 Per – Agent Reward over Episodes*



*Figure : 4 Average Step Per Esisode*



*Figure : 5 Average loss per Esisode*

*Agent Learning Performance Over Time:* The training results of the DQN model combined with Curiosity-driven Exploration demonstrate that the agent learns effectively and stably throughout the episodes. Specifically, the average reward curve shows a continuous increase as the number of episodes grows, indicating that the agent progressively discovers better strategies to accomplish tasks in the environment. After around 400 episodes, the reward curve begins to converge, suggesting that the agent’s policy has stabilized, with minimal large changes, achieving a high level of learning performance.

Simultaneously, the plot of steps per episode reveals significant improvement. In the early stages, the number of steps fluctuates considerably as the agent explores the environment and tries different paths. However, as training progresses, the number of steps gradually decreases and stabilizes, showing that the agent has found shorter and more efficient routes to complete the tasks. This clearly illustrates the important role of curiosity-driven exploration, which not only encourages broad exploration initially but also enables the agent to effectively leverage the knowledge gained to optimize its actions.

Additionally, the loss curve reflects the convergence of the DQN network’s weight updates. The loss decreases rapidly in the early phase, indicating that the network quickly learns the relationships between states and actions. Afterwards, it oscillates steadily at a lower level, demonstrating that the model avoids overfitting or imbalanced Q-value updates. The stability of the loss also implies that the model generalizes well and maintains performance throughout extended training.

Overall, the three curves—reward, steps per episode, and loss—work together to illustrate an effective learning process, balanced exploration and exploitation, and knowledge optimization by the agent. The model not only achieves high performance but also maintains stability throughout training, clearly demonstrating the success of integrating DQN with curiosity-driven exploration to enhance the agent’s learning efficiency in a complex environment.

**Chapter 6: Conclution**

The achieved results of the project include the successful development of a multi-agent simulation environment on a 15x15 grid, where obstacles and pickup/drop-off locations are randomly generated to better simulate real-world scenarios. This environment supports multiple agents operating concurrently, with communication capabilities and mutual state awareness, enabling effective coordination and collision avoidance during task execution.

Each agent is equipped with a Deep Q-Network (DQN) model and has been successfully trained to learn optimal policies for movement, pickup, and delivery based on observed states. The use of a replay buffer and an epsilon-greedy strategy allows agents to balance exploration and exploitation effectively, thereby improving learning efficiency.

The effectiveness of multi-agent coordination is clearly demonstrated as agents learn to avoid collisions and cooperate to reduce overall delivery time. Additionally, agents learn to utilize communication information to enhance joint strategies, thus improving group performance. The results show that the average task completion time decreases steadily over training episodes, the delivery success rate exceeds 85%, and collision avoidance significantly improves, reducing congestion in the operational environment.

A visual simulation interface developed with Pygame clearly displays agents’ movements, pickups, and deliveries within the simulated environment. The system also provides state and reward information throughout the training process, allowing easy monitoring and evaluation of learning performance. All these outcomes confirm the effectiveness and feasibility of applying Deep Reinforcement Learning to solve multi-agent coordination problems in complex and practical environments.

Through the project, the research team has gained valuable knowledge and experience in Deep Reinforcement Learning, particularly in applying the DQN algorithm in complex multi-agent environments. The team also developed a deeper understanding of environment design, reward shaping, and challenges related to coordination and collision avoidance among agents. Moreover, creating a visual simulation interface enhanced programming and real-time simulation skills. These insights not only contributed to the project’s success but also laid a solid foundation for developing practical applications in robotics, smart logistics, and automated coordination systems in the future.

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**