Multi-Agent System for Cooperative Navigation

Magdalena Warylak
111531
Instituto Superior Técnico

Jakub Krupiński
111530
Instituto Superior Técnico

ABSTRACT

Project presents a multiagent system for cooperative navigation, addressing coordination challenges and communication dynamics among autonomous agents. Leveraging the Simple Spread framework, the system orchestrates interactions to optimize task performance while minimizing collisions. Through simulations with agents and landmarks, agents learn individual policies and coordination strategies to cover landmarks efficiently. Evaluation metrics assess both local collision avoidance and global task performance.

1 Introduction

In recent years, multiagent systems have garnered significant attention due to their ability to model and solve complex problems by coordinating the actions of multiple autonomous agents. This project aims at developing a multiagent system for cooperative navigation, with the objective of addressing challenges related to coordination problems, communication and cooperation between agents.

The motivation behind this project stems from the growing need for efficient coordination and collaboration among autonomous entities in various domains. With the rise of artificial intelligence and autonomous systems, there is a pressing demand for multiagent system solutions that can adapt to dynamic environments and handle uncertainty. In many real-world scenarios, such as robotic swarm deployment, autonomous vehicle routing, and distributed sensor networks, effective coordination among agents is essential for achieving optimal outcomes.

The problem we aim to address revolves around coordinating a group of autonomous agents to perform collaborative tasks efficiently. For this purpose, we use the Spread environment from the PettingZoo framework. Specifically, the Spread environment addresses the problem of multi-agent, which need to learn both individual policies and coordination strategies to achieve optimal performance. Agents interact with each other and the environment, making decisions that impact on all agents' rewards.

Our main objective is to simulate the problem using a multiagent system based on Spread environment and test it in terms of policy given to the agents. We would like to study the behaviour of agents, based on the policies given to them and inferred using reinforcement learning.

2 Related Work

In the realm of Multi-Agent Reinforcement Learning (MARL), addressing cooperative problems has garnered considerable attention. One notable contribution is the paper titled "On Solving Cooperative MARL Problems with a Few Good Experiences." as shown in [1]. This work explores the challenges of cooperative MARL and proposes a solution based on experience replay, where agents store and reuse successful experiences to improve learning efficiency and coordination.

Another relevant study in cooperative MARL is [2]. This comprehensive review discusses various theoretical frameworks and algorithms for cooperative MARL, shedding light on the complexities and potential solutions in this domain.

3 Approach

To simulate the problem, we use an environment that has N=3 independent agents and N=3 landmarks. Agents must learn to cover all the static landmarks while avoiding collisions. More specifically, all agents are globally rewarded based on how far the closest agent is to each landmark (sum of the minimum distances). Locally, the agents are penalized if they collide with other agents (-1 for each collision).

> Agent observations: [self_velocity, self_position, landmark_relative_positions, other_agent_relative_positions, communication]
> Agent possible actions: [no_action, move_left, move_right, move_down, move_up]

Each agent to coordinate needs to know information about other agents' positions (to avoid collisions) as well as positions of landmarks (to know which area to cover). With that information agents can learn a policy or try one already defined to maximize the defined rewards.

Each agent can have one of three policies:

- Go straight to the nearest landmark
- Go straight to the nearest landmark, but to avoid collisions each agent can choose only one point
- Policy based on reinforcement learning using DQN neural network

4 Empirical evaluation

Defined metrics for evaluating agents' behaviour will be sum of local and global rewards. Local rewards will indicate how often an agent collides with other agents. Global rewards show how well the task is performed by all the agents. Ideally the agents will maximize the rewards while not being penalized for collisions. The local reward will play the largest role when using reinforcement learning. In the case of the first two policies, the reward obtained will also consist of a global and a local rewards. For setting targets, however, agents do not take into account possible collisions - they choose the target to which they are closest.

5 Game-theoretical formalism

The presented problem of coordinating three agents in one environment can be described as a Stochastic Game or Markov Game. It involves multiple agents interacting in a shared environment where their individual actions and states influence the overall outcome, and they must learn strategies to maximize their rewards while considering the actions of other agents. The following is a detailed overview of the environment using Markov's game description:

Finite set of n = 3 agents:

- N = {agent_0, agent_1, agent_2}

Finite set of actions. Each agent i can choose an action a_i from the following set:

- A ={no_action, move_left, move_right, move_down, move_up}

Finite set of states of dimension (54,), which consists of:

- self-velocity and position for each agent
- relative positions of landmarks to each agent.
- relative positions of other agents to each agent.
- communication signals among agents

The reward function R: $S \times A \times S \rightarrow \mathbb{R}$ consists of two components:

- global reward - based on the sum of minimum distances from each landmark to the closest agent:

$$R_{global} = -\sum_{k=1}^{3} min_{i=0,1,2} ||landmark_k - agent_i||$$

- local penalty - for collisions between agents:

$$R_{local} = -1 \cdot num\_of\_collisons$$

- total reward for agent i is a weighted sum:

$$R_i(\mathrm{s}, \mathrm{a}, \mathrm{s}') = (1 - ratio) \cdot R_{global} + ratio \cdot R_{local}$$

  where ratio is weight applied to local and global reward to decide their importance on total rewards

Each agent i follows a policy $\pi_i : S \rightarrow A$ to select actions based on the current state. The policy for agent i can be represented as π_i (s_i) ∈ A. The agents can use one of the following policies (which are described in detail in the next section):

- Simple Policy
- Complex Policy
- Reinforcement Learning Policy

To coordinate, agents exchange information about:

- positions of other agents (to avoid collisions).
- positions of landmarks (to determine coverage areas).

Each agent aims to maximize its cumulative reward over time by choosing actions according to its policy. The main goal is to cover all landmarks effectively while minimizing collisions.

6 Policies

In our project, we decided to use three different policies for agents to achieve their goals. In this way, we want to compare whether the use of reinforcement learning algorithms will positively affect the reward results obtained. The first two policy are based only on observations that agents make - we did not use any learning algorithms in them. The third approach uses reinforcement learning based on neural networks. Each policy is described in more detail in the subsections below.

In all of our solutions we use parallel environment. It allows for simultaneous actions and interactions of multiple agents in a shared space, which is closer to real-world scenarios and essential for testing algorithms that require real-time decision making.

6.1 Simple policy

In a simple policy, we decided to use the approach that first comes to mind - let each agent choose the closest goal and pursue it. The policy was defined based on observations made for each agent separately. Based on information about its own location and the location of the landmarks, the agent makes distance calculations and selects the landmark that is closest to it. Then, based on the distance vector, it decides in which direction to move according to the logic:

```
if |target_pos.x| > |target_pos.y|:
    if target_pos.x > EPSILON:
        move RIGHT
    else:
        move LEFT

else:
    if target_pos.y > EPSILON
        move UP
    else:
        move DOWN
```

According to the environment, an agent can only move in four directions, so we first check on which axis the agent is further away from the target and whether it should choose horizontal or vertical movement first. Then, based on the distance vector, we determine the exact movement on a given axis. If an agent in both directions approaches the target at a distance less than the EPSILON variable (=0.1) it stops moving. The only factor that can then cause it to move is a collision with another approaching agent. The agent's observations are updated with each step made, so at any time the agent can get closer to another landmark and change its target.

Despite its simplicity, such an inflicted policy has some disadvantages that we can point out without research - agents do not exchange information about their targets, so collisions can occur, which are penalized.

6.2 Complex policy

Complex policy is based on a similar premise to simple, but already takes into account communication between agents. after making observations, the positions of the agents and landmarks are converted to a global coordinate system, and then the distances are calculated for each agent we determine the length of the path he must take to each landmark. In this way we obtain a distance matrix based on which to each landmark is assigned the agent that is closest to it. This modification allows us to simulate communication between agents - since each knows the position of the others it gives priority to the agent that is closer to the selected marker. The landmark selection is performed every step to avoid possible collisions. In the next steps, agents perform actions to approach their chosen landmarks. Action selection is based on the same principles used in simple policy.

As with simple policy, we expect some drawbacks to the defined solution. For some cases, agents' paths may cross, which will cause collisions, and the chosen solution does not present any solution for agents to avoid them.

6.3 DQN

In the process of learning our agents, we applied reinforcement learning and used a deep q-learning algorithm based on a DQN neural network.

6.3.1 Introduction to DQL

Deep Q-Learning (DQL) is an advanced form of Q-learning. Traditional Q-learning involves learning a Q-value function that maps state-action pairs to rewards, essentially guiding an agent to take actions that maximize its cumulative future rewards in a given environment. In Q-learning, the Q-value function is updated using the Bellman equation. Deep Q-Learning enhances this concept by using deep neural networks to approximate the Q-value function. This allows DQL to handle high-dimensional state spaces, such as those encountered in image-based inputs or complex environments where tabular methods are infeasible.

A Deep Q-Network (DQN) is the specific implementation of the DQL algorithm that utilizes a neural network to approximate the Q-value function. The neural network, referred to as the Q-network, takes the state as input and outputs Q-values for each possible action. The main components of DQN include:

- **Q-Network**: A neural network that estimates the Q-values for all actions given a state.
- **Experience Replay**: A memory buffer that stores the agent's experiences. During training, batches of experiences are sampled randomly from this buffer to break the correlation between consecutive experiences and stabilize training.
- **Target Network**: A separate network with the same architecture as the Q-network, used to compute target Q-values. The target network's weights are updated periodically with the Q-network's weights to stabilize training.

6.3.2 DQN Model Architecture

Our DQN model consists of a neural network designed to process the agent's state and output Q-values for each possible action. The architecture of it is shown on Figure 1. and includes:

- An input layer that receives the state vector, encompassing the agent's velocity, position, and relative positions of landmarks and other agents.
- Two hidden layers with 256 and 128 neurons respectively, both using ReLU activation functions.
- An output layer that produces Q-values for the five possible actions: no action, move left, move right, move down, and move up.

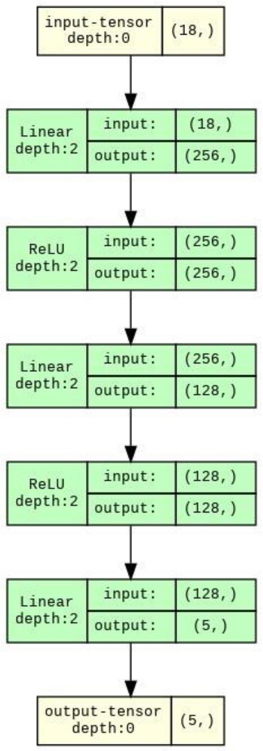


Figure 1. DQN Architecture

We used MSELoss as the loss function, which measures the mean squared error (squared L2 norm) between each element in the input x and target y. As an optimizer we used Adam, which is a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments.

To stabilize training, we implement an experience replay memory, which stores transitions (state, action, next state, reward, done) encountered by the agents. During training, batches of experiences with a size of 32 are randomly sampled from this buffer to update the Q-network's parameters, facilitating efficient learning from past interactions.

6.3.3 Training

In our project, we use a separate neural network to learn each agent. This procedure has ensure that each agent can learn a unique and specialized policy tailored to its own experiences and observations within the environment. This approach helps to avoid policy overlap and interference, promoting diverse strategies among agents, which ultimately leads to more effective coordination and minimizes the global reward by efficiently covering landmarks and avoiding collisions.

The training process of the model starts with the initialization phase, where hyperparameters such as the learning rate, discount factor, network synchronization rate, replay memory size, and mini-batch size are defined. During this phase, Deep Q-Network (DQN) models, including both policy and target networks, are initialized for each agent. Additionally, optimizers and loss functions are set up for each agent, and experience replay memories are prepared.

Following initialization, the training process enters the episode loop. At the beginning of each episode, the environment is reset and the initial state is obtained. During each step within the episode, each agent selects an action based on an ε-greedy policy. The chosen actions are executed in the environment, leading to new states and rewards. These transitions are stored in the replay memory. Once a sufficient number of experiences have been collected, a mini-batch is sampled from the replay memory to optimize the DQN. The exploration rate (ε) is gradually decayed over time. Periodically, the policy network is synchronized with the target network. The model is also saved periodically and at the end of the training.

Optimization is performed on each mini-batch by computing the target Q-values using the Bellman equation. The policy network is then updated by minimizing the loss between the current Q-values and the target Q-values. This process involves using backpropagation and an optimizer to adjust the weights of the policy network.

The prepared policy models were achieved through a learning process in which each agent performed 25 steps in 100,000 learning episodes. in each episode, the local_ratio was set to 0.5 so that the local and global reward values would have the same meaning. For each episode, the average value of the rewards obtained was calculated. The obtained results for agents, along with the trend lines, are shown successively in Figures 2, 3, 4.

Due to the scale of the training (a large number of episodes) and the parallel execution of actions, the results for all agents look similar, but on closer inspection like the execution of 10 episodes, one can easily see the difference in results. The reward values for such a test are shown in Table 1.

Table 1. Average Rewards for agents during 10-episodes learning

Episode	Average Rewards		
	Agent 0	Agent 1	Agent 2
0	-0.86	-0.83	-0.86
1	-1.29	-1.29	-1.32
2	-1.26	-1.26	-1.26
3	-0.92	-0.92	-0.90
4	-0.96	-1.02	-1.02
5	-1.04	-1.07	-1.04
6	-1.54	-1.54	-1.54
7	-1.4	-1.46	-1.4
8	-1.41	-1.41	-1.41
9	-1.44	-1.42	-1.45

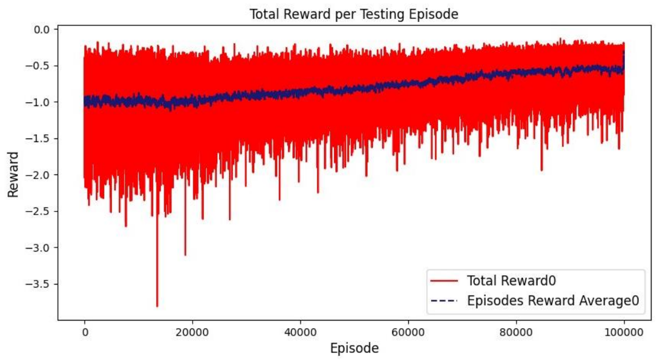


Figure 2. Average rewards for agent_0 obtained in the learning process

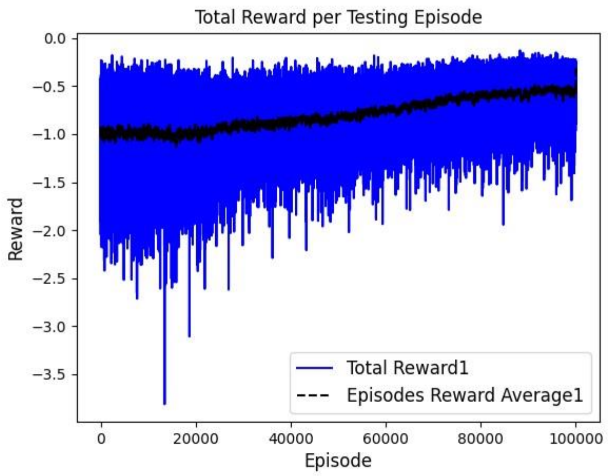


Figure 3. Average rewards for agent_1 obtained in the learning process

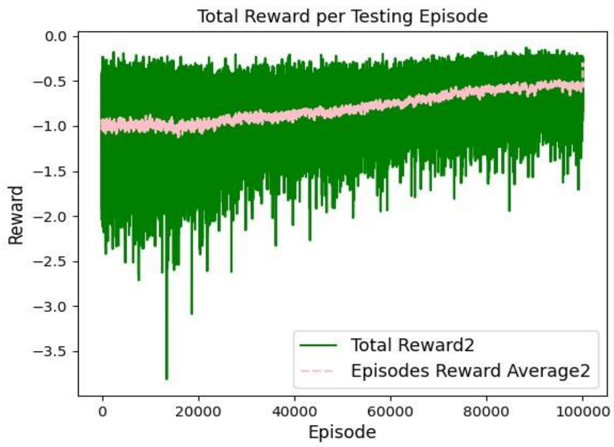


Figure 4. Average rewards for agent_2 obtained in the learning process

The results in the form of policy models were recorded at the end so that they could be used in tests.

7 Experiments

We decided to run three different tests to determine which approach achieves the best results. The focus of our tests was to see whether it was better for agents to follow the same policy at the same time or whether it might be worthwhile for each to follow something different. In addition, we wanted to emerge which of the prepared policies allows for better results in the long run. We conducted preliminary tests for 25 steps and using a local_ratio value of 0.5.

7.1 Comparison of the same policies

In the first test we conducted, we compared the behavior of agents according to the policies they follow, with the proviso that each agent must base the choice of his action on the same policy as the others. For the three selected seeds 10, 69 and 73 we collected results relating to the final rewards and average rewards for each agent individually as well as the average reward for all agents. All agents were guided first only by the policy based on reinforcement learning, then simple policy and finally complex policy. The values of the final rewards obtained are shown in Table 2 and the average rewards from the entire run in Table 3.

Table 2. Final rewards obtained for selected seeds

Seed	Policy	Final Rewards			
		Agent 0	Agent 1	Agent 2	Average
10	RL	-0.1	-0.1	-0.1	-0.1
69	RL	-0.22	-0.22	-0.22	-0.22
73	RL	-0.26	-0.26	-0.26	-0.26
10	SP	-0.4	-0.4	-0.4	-0.4
69	SP	-0.83	-0.83	-0.83	-0.83
73	SP	-0.3	-0.3	-0.3	-0.3
10	CP	-0.27	-0.27	-0.27	-0.27
69	CP	-0.29	-0.29	-0.29	-0.29
73	CP	-0.32	-0.32	-0.32	-0.32

Table 3. Average rewards obtained for selected seeds in whole run

Seed	Policy	Average Rewards			
		Agent 0	Agent 1	Agent 2	Average
10	RL	-0.56	-0.56	-0.54	-0.55
69	RL	-0.32	-0.32	-0.32	-0.32
73	RL	-0.67	-0.67	-0.67	-0.67
10	SP	-1.22	-1.18	-1.1	-1.17
69	SP	-0.75	-0.89	-0.89	-0.84
73	SP	-0.49	-0.49	-0.49	-0.49
10	CP	-0.77	-0.81	-0.79	-0.79
69	CP	-0.36	-0.36	-0.36	-0.36
73	CP	-0.48	-0.48	-0.48	-0.48

Based on the results, we can draw preliminary conclusions that reinforcement learning is the best solution. However, for such a small research sample, this is too large a generalization, so we decided to run the program for 50 random seeds (layouts of landmarks). Based on the collected values of average final rewards, we prepared the graph shown in Figure 5.

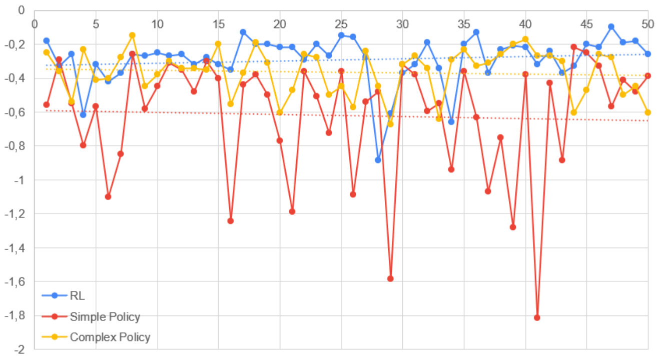
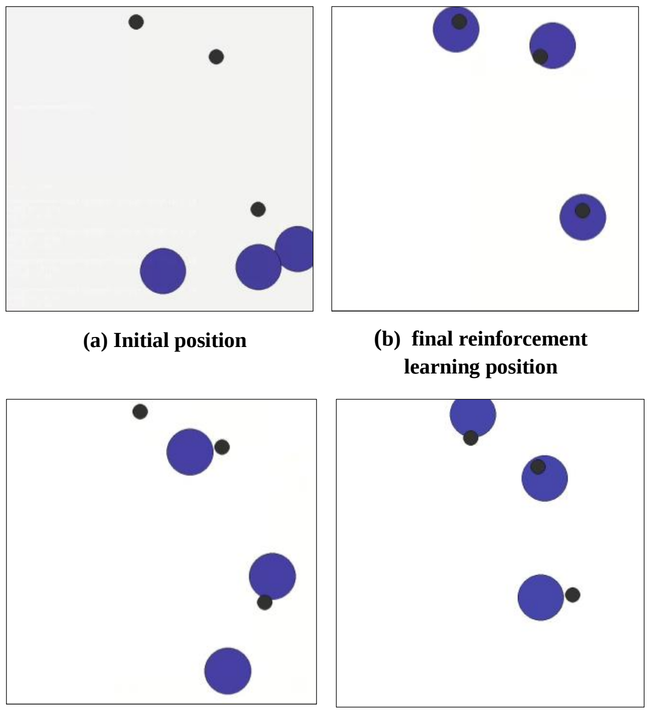


Figure 5. Comparison of average reward values

Based on the results, it can be seen that the rewards obtained with reinforcement learning are the highest for most cases. Agents get the lowest value of rewards if they move according to simple policy.

To further illustrate the difference in agent behavior below in Figure 6 we show the initial position of the agents and the final position using each policy for seed = 10. Based on this figure, it is clear that agents using reinforcement learning reach the closest to the landmarks



(a) Initial position **(b) final reinforcement learning position**

(c) final simple policy position **(d) final complex policy position**

Figure 6. Comparison of initial and final positions for different policies

7.2 Comparison of different policies

In the second test, we examined whether agents moving each with a different policy would get better results. We conducted the tests on the two layouts of landmarks that were used in the previous tests - 10 and 73. We tested all available combinations in which agents can perform actions, with the assumption that two agents cannot use the same policy. The results obtained are presented in Table 4.

Table 4. Average rewards obtained for mix policies

Seed	Policy for Agent			Average Rewards for Agent			
	0	1	2	0	1	2	Avg.
10	RL	SP	CP	-0.32	-0.32	-0.32	-0.32
10	RL	CP	SP	-0.97	-0.97	-0.47	-0.80
10	SP	RL	CP	-1.01	-1.01	-0.51	-0.84
10	CP	RL	SP	-0.85	-0.85	-0.85	-0.85
10	SP	CP	RL	-0.78	-0.78	-0.78	-0.78
10	CP	SP	RL	-0.72	-0.72	-0.72	-0.72
73	RL	SP	CP	-0.48	-0.48	-0.48	-0.48
73	RL	CP	SP	-0.48	-0.48	-0.48	-0.48
73	SP	RL	CP	-0.72	-0.72	-0.72	-0.72
73	CP	RL	SP	-0.65	-0.65	-0.65	-0.65
73	SP	CP	RL	-1.08	-1.08	-1.08	-1.08
73	CP	SP	RL	-1.33	-1.33	-1.33	-1.33

In most cases for mixed policies, the reward values obtained are lower than when only one policy is used. The results indicate that when agents use mixed policies (each agent follows a different policy), the overall reward values are generally lower compared to scenarios where all agents follow a the same policy. This suggests that mixed policy approaches may introduce inefficiencies or conflicts in agent behavior.

7.3 Impact of *local_ratio*

As the last one, we conducted a study of how the results obtained are affected by the factor that introduces weighting between local and global awards. We conducted the test on one layout (seed = 10) only for the case when agents move according to the same policy. The results obtained are shown in Table 5.

Table 5. Impact of local-ratio on rewards

local ratio	RL		Simple policy		Complex policy	
	Final	Avg	Final	Avg	Final	Avg
0	-0.2	-1.08	-0.8	-2.03	-0.54	-1.49
0.1	-0.18	-0.97	-0.72	-1.86	-0.49	-1.35
0.25	-0.15	-0.82	-0.6	-1.61	-0.41	-1.14
0.5	-0.1	-0.56	-0.4	-1.18	-0.27	-0.8
0.75	-0.05	-0.3	-0.24	-0.75	-0.14	-0.46
0.9	-0.02	-0.14	-0.18	-0.48	-0.10	-0.22
1	-0.0	-0.08	-0.15	-0.30	-0.06	-0.14

As the local-ratio increases from 0 to 1, the reward values (both final and average) improve across all policies (RL, Simple, and Complex). This indicates that placing more emphasis on local rewards tends to result in better overall performance. The results suggest that a balanced or skewed focus towards local rewards can significantly enhance agent performance. Purely global rewards (local-ratio of 0) lead to the worst performance, likely due to the lack of immediate feedback that guides local decision-making.

8. Conclusion

The solution we created meets the objectives - the prepared implementation allows simultaneous coordinated actions of multiple agents. The choice of action for each agent can be based on one of three proposed policies, one of which is based on advanced reinforcement learning.

Our tests revealed several key insights into the effectiveness of different policies and strategies for multi-agent coordination. From these, we can conclude that reinforcement learning policies, based on DQN networks, consistently outperform simple and complex policies, achieving higher rewards in both preliminary and extended tests. This indicates that RL is the most effective policy for agent coordination and task completion in our environment.

Regarding the implementation of various policies, it is clear that, agents following the same policy simultaneously perform better than those using mixed policies. The lower rewards in mixed policy scenarios could be attributed to coordination challenges. When agents follow different policies, their actions might not be well-coordinated, resulting in less effective collective behavior. This highlights the importance of synchronized strategies in multi-agent systems for achieving higher rewards.

In our environment, where we can define the weights of rewards and punishments for agents in different ways, emphasizing local rewards (higher local_ratio) significantly improves agents performance across all policies. Local feedback is crucial for effective decision-making and task execution

These findings underscore the importance of using RL policies, maintaining uniform strategies among agents, and focusing on local rewards to optimize performance in multi-agent systems.

REFERENCES

[1] Rajiv Ranjan Kumar and Pradeep Varakantham, 2020. Solving Cooperative MARL Problems with a Few Good Experiences. DOI: https://doi.org/10.48550/arXiv.2001.07993.

[2] Kaiqing Zhang, Zhuoran Yang and Tamer Başar. 2021. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. DOI: https://doi.org/10.48550/arXiv.1911.10635.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. DOI:https://doi.org/10.48550/arXiv.1312.560