AY2021 Master Thesis

Analysis of Coordination Structures of Partially Observing Cooperative Agents by Multi-Agent Deep Q-Learning

Ken Smith

A Thesis Submitted to the Department of Computer Science and Communications Engineering, the Graduate School of Fundamental Science and Engineering of Waseda University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Student ID	5119FG08-9
Date of Submission	January 25th, 2021
Advisor	Prof. Toshiharu Sugawara
Research Guidance	Research on Intelligent Software

Contents

Abstract

We compare the coordination structures of agents using different types of inputs for their deep Q-networks (DQNs) by having agents play a distributed task execution game. The efficiency and performance of many multi-agent systems can be significantly affected by the coordination structures formed by agents. One important factor that may affect these structures is the information provided to an agent's DQN. In this study, we analyze the differences in coordination structures in an environment involving walls to obstruct visibility and movement. Additionally, we introduce a new DQN input, which performs better than past inputs in a dynamic setting. Experimental results show that agents with their absolute locations in their DQN input indicate a granular level of labor division in some settings, and that the consistency of the starting locations of agents significantly affects the coordination structures and performances of agents.

Preface

This paper is based on "Analysis of Coordination Structures of Partially Observing Cooperative Agents by Multi-Agent Deep Q-Learning" from the PRIMA 2020 conference with additional experimental results and discussion. This work is partly supported by JSPS KAKENHI Grant number 17KT0044.

1 Introduction

Although cooperation and coordination are crucial in many multi-agent systems for improving overall efficiency, designing the appropriate coordination and cooperation regime is challenging because many factors must be considered, such as the problem structure, environmental characteristics, and agent abilities. Additionally, a change in any of these parameters can drastically change the optimal regime for a given environment, making the task of finding the appropriate regime a challenge.

Recently, owing to the development of deep reinforcement learning (DRL), researchers have successfully acquired coordinated behaviors [4,5,12]. Foerster et al. [4] gave multiple agents with deep distributed recurrent Q-networks the task of solving riddles requiring communication and coordination to find the correct answer. The agents were successful in finding the answer to these riddles, so effective coordination and communication was present. Gupta et al. [5] were able to extend three single-agent DRL methods to a cooperative multi-agent context. Lowe et al. [12] introduced a centralized critic in a multi-agent deep reinforcement learning (MADRL) system to allow these agents to outperform agents without a centralized critic.

These studies provide useful insight into how MADRL can improve performance by enabling agents to coordinate and cooperate with one another under some circumstances. However, the focus of these studies was improving the performance of agents and how network architecture and parameter settings affected this performance, and the researchers did not discuss the reason from the viewpoint of what kinds of coordination structures emerged in multi-agent environments. Therefore, we cannot predict the features of emerging coordination regimes, such as robustness, tolerance, and adaptability to changes in the environment and agents (such as the failure or upgradation of agents).

Because it is important to understand the features of the coordination regime, the coordinated/cooperative behaviors generated by DRL, and how these features are affected by the neural network and input structure, a few studies have attempted to identify the coordinated behaviors learned with DRL [9, 13]. For example, Leibo et al. [9] discovered that agents may learn selfish or cooperative behaviors depending on the amount of resources available in the environment. Miyashita & Sugawara [13] focused on the structures and information included in the input to networks and attempted to obtain a coordination regime in a task execution game within the multi-agent system context; subsequently, they analyzed coordination structures that emerged from agents by observing different aspects of the environment. They discovered that by allowing agents to observe their absolute locations within the environment, the agents can allocate their workload and improve the overall performance. However, the studies were conducted in a rudimentary environment. Consequently, there is not enough information to assess how agents with these observational methods would perform in more complex environments involving obstructions in movement and line of sight, which is critical for real world application; hence, further studies should be performed.

Our goal is to analyze the change in the coordinated behavior of agents generated by DRL based on input structures using a variant of a task execution game [13] in a more complex environment that is close to a real world setting to better clarify the relationship between the input structure and the generated coordination regime. To this end, we compare the performance and coordination regimes of input structures that include the absolute location of agents with input structures that only include an agent's local environment. Given the widespread usage of GPS technology, we believe the implementation of input structures involving an agent's absolute location has a strong potential to be feasible.

Our experiments indicate that allowing agents to observe their absolute locations within their environments resulted in a high degree of divisional cooperation and improved performance. Furthermore, we discovered that under a less consistent environment where the starting positions of agents were randomized over a designated area, the agent's observational method must be reconfigured to maintain an improved performance level. However, the information provided to the agent remained nearly identical. Further investigations were made to identify the cause of the difference in performance. Although no conclusive evidence was acquired, we managed to rule out several possibilities leading us to believe that the low performance originated from the structure of the input given to the agents.

2 Related Studies

A significant amount of research has been conducted on DRL [6–8, 10, 14, 15, 20, 21]. For example, Mnih et al. [14] succeeded in training a DQN with convolutional layers to play a variety of Atari games using the raw pixels shown by these games as input. Hessel et al. [7] used Atari games as a benchmark to compare the performance of several DQN variants including their proposed variant called "Rainbow". The researchers were able to demonstrate that Rainbow delivered superior performance over the other DQN variants. Wang et al. [21] proposed a new neural network architecture called the "dueling network" which provided better performance compared to state-of-the-art models when learning to play Atari games.

Some studies have included coordination, cooperation, or communication between agents in a multi-agent setting [2,4,5,9,12]. For example, Gupta, Egorov, and Kochenderfer [5] compared the performances of several DRL algorithms in a cooperative multi-agent setting, where agents must coordinate with each other to succeed. The study indicated that one method outperformed the other methods, and that recurrent neural network architectures delivered better performances overall compared with feedforward networks. Lowe et al. [12] applied lenient learning to the deep Q-networks (DQNs) of agents in a multi-agent setting, which associates state–action pairs with decaying temperature values to avoid updating agent policies with outdated pairs. They discovered that by applying leniency to the learning process, cooperation was facilitated in a fully cooperative environment. Foerster, Assael, Freitas, and Whiteson [4] analyzed how multiple agents, each using their own recurrent DQN, attempted to solve riddles that required communication between agents to succeed. They discovered that agents successfully developed a communication protocol to solve these riddles, and that the communication protocols can be modeled in the form of a decision tree. In general, these studies involved agent coordination but focused primarily on the performance of agents rather than the coordination regimes. Although the performance improvement of multi-agent systems is important, more research should be performed regarding the coordination structures of agents to acquire new information.

A few studies have focused on the analysis of coordination and cooperation regimes formed in a multi-agent setting. To illustrate, Diallo & Sugawara [2] analyzed strategic group formations developed using a combination of centralized and decentralized deep Q-learning in a nonstationary and adversarial multi-agent environment. The study used variants of the DQN and indicated that combining a double DQN and a DuelDQN generated strategies that delivered improved performances. Leibo et al. [9] investigated cooperative and selfish behaviors formed from multiple independently learning agents with DQNs playing two different games in a two-dimensional grid world environment. The researchers analyzed the agents' strategies in environments with varying degrees of resource scarcity and introduced the concept of sequential social dilemmas. Studies that focused on the coordination regimes of multi-agent systems, such as the aforementioned, are rare. Therefore, further research is required to investigate the coordination and cooperation structures of agents.

3 Background

3.1 Reinforcement Learning

Reinforcement learning is based on the Markov Decision Process (MDP). MDP is often shown as a tuple $< S, A, r, P, \gamma >$ where there are a set of states S and a set of actions A. We have a probability function P which determines to probability at time t for a state-action pair to lead to a state transition such that $P(S_{t+1}|S_t, A_t)$. Reward r is provided for every state-action pair and the resulting state transition such that $r(S_{t+1}, S_t, A_t)$. The goal in reinforcement learning is for the learning agent to learn the optimal policy $\pi : S \rightarrow A$ which maximizes $\sum_{t=0}^{\infty} \gamma^t r_t$, the cumulative discounted reward at time t where $0 \leq \gamma < 1$ is the discount factor.

The discount factor determines the level of influence that future rewards have on present policy of the agent when determining an action; $\gamma = 0$ would train an agent to learn a policy maximizing current rewards while a γ value close to 1 would train the agent to maximize long term rewards.

3.2 Deep Q-Learning

Q-learning [22] is a form of reinforcement learning that attempts to find the optimal policy through the use of Formula 1:

$$Q'(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{1}$$

Formula 1 utilizes the concept of the Q-value, which is the expected reward, at time t, for taking action a_t at state s_t, denoted as $Q(s_t, a_t)$. A Q-value for each state-action pair is stored in a table and updated over several iterations to eventually converge to a value which allows agents at state s_t to take action a_t to maximize discounted cumulative rewards. $Q'(s_t, a_t)$ denotes the updated Q-value which will replace $Q(s_t, a_t)$. $0 < \alpha \leq 1$ is the learning rate, which is the value that determines the magnitude of the changes made to the Q-value after each iteration. r_t is the observed reward after taking action a_t at state s_t, and γ is the discount factor. $\max_a Q(s_{t+1}, a)$ is the maximum possible reward that is expected to be obtained in state s_{t+1}.

Deep Q-learning [3, 14] replaces the Q-table with a multi-layered neural network which is trained to approximate the optimal action for each state. With environments of increasing complexity, Q-learning will require a larger Q-table, and thus more memory. Deep Q-learning has the advantage of not necessarily needing more memory with an increased complexity of the environment.

3.3 Multi-Agent Patrolling Problems

Finding efficient methods for multiple agents to autonomously patrol an area can be useful in a multitude of contexts such as search and rescue operations, cleaning tasks, and security patrols. However, coordinating multiple agents to effectively conduct a patrolling task is complex and challenging. Many researchers have attempted to address the issue in the past [17] and continue to do so.

Sugiyama et al. [18] is one such study where researchers attempted to address the Continuous Cooperative Patrolling Problem (CCPP). In CCPP, multiple agents patrol an environment where different areas have different visitation requirements. Agents are not given instructions on how to cooperate or patrol the environment; thus, agents must learn on their own to coordinate and cooperate. Similar to CCPP, Miyashita & Sugawara [13] used a distributed task execution game where tasks appear at random locations within the environment, and without any clear instructions on how to cooperate or patrol, multiple agents learn to complete these task in some manner.

As Portugal & Rocha [17] noted, finding an effective solution to these patrol problems offer the "potential to replace or assist human operators in tedious or dangerous real-life scenarios" [17] and "the possibility to relieve human beings, enabling them to be occupied in nobler tasks" [17]. As such, researching autonomous solutions to these patrolling problems is critical.

4 Problem Formulation

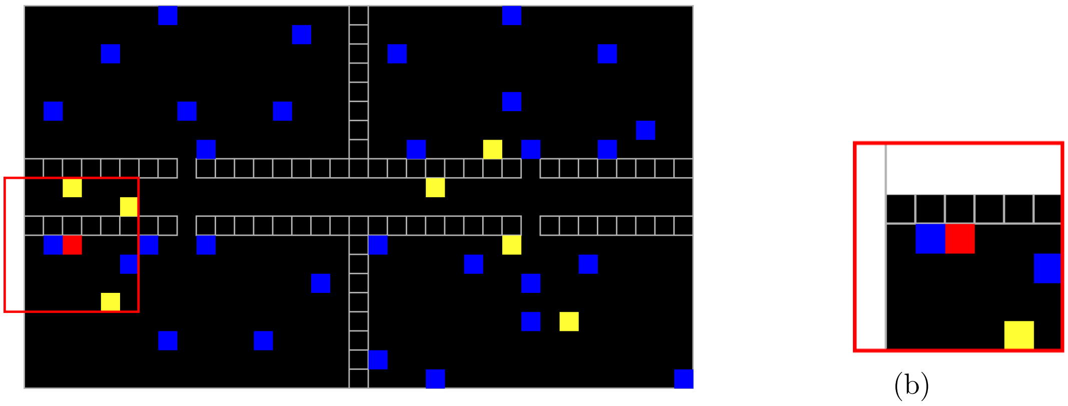
4.1 Problem and Environment

To analyze the coordinated behavior of multiple agents, we introduce a multi-agent problem called the *distributed task execution game.* The problem environment S can be represented as a two-dimensional grid comprising 35×20 cells. An example of this environment is shown in Fig. 1a. In this grid, three different types of objects can occupy the cells: agents, tasks, and walls. Each instance of an object occupies one cell. Walls are arranged in the environment to create a 35×2 corridor spanning the environment horizontally. Fig. 1a shows the wall locations of the environment marked as gray boxes. From the corridor, agents can access four different rooms, each the size of 17×8 cells. Each room has 1 entryway the size of 1 cell. Initially, a certain number of tasks were scattered in environment S. The corridor and cells between the corridor and each room did not have any task throughout the experiment.

Next, we introduce a time unit called a step. Let A be a set of agents; agent $i \in A$ takes one action from $M = \{\text{Up, Down, Left, Right}\}$ in every step. If i moves on a cell that is occupied by a task, then it is assumed to be executed by i, and a new task is spawned at a randomly selected empty cell in room $s \in S$; therefore, the number of tasks in the environment is unchanged.

Once a task is executed, the agent that executed the task receives a positive reward. Each agent learns, using a DQN, to determine actions that maximize the reward earned by considering other agents' actions cooperatively. If i attempts to move to a cell containing a wall or outside of the environment, i remains in the same position for that step. We consider this event to be a wall collision. Likewise, if i attempts to move to a cell containing another agent, i does not change its position for that step; we consider this event to be an agent collision. After agents perform their actions and new tasks are spawned, the next step begins. At the end of step 300, the episode ends; the DQNs of all agents are updated independently with the rewards and observations of each agent. Subsequently, the environment is initialized for the next episode.

As shown in Fig. 2, certain areas in the environment are marked in blue or red. The areas marked in blue signify areas where tasks can spawn, whereas those marked in red signify areas where agents can spawn. The areas marked in red in Fig. 2a are numbered according to the ID of each agent, indicating that the same agent spawns in the exact same location whenever the environment is initialized.


(a) Actual Environment (b) Local View

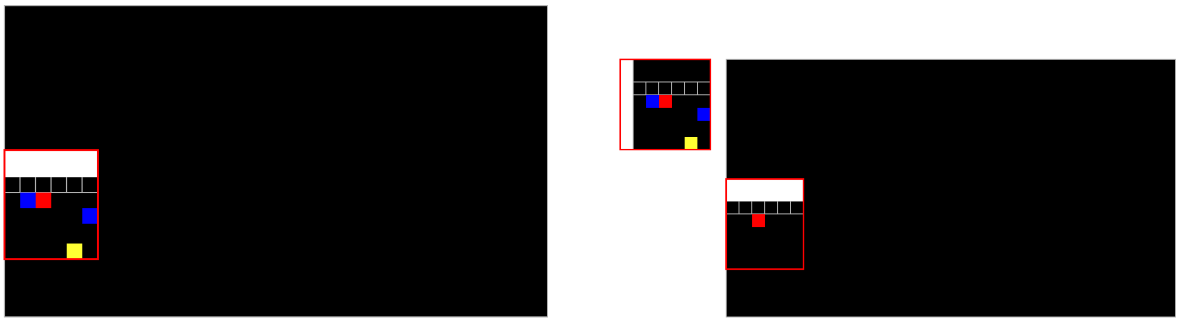

(c) Relative View (d) Merged View

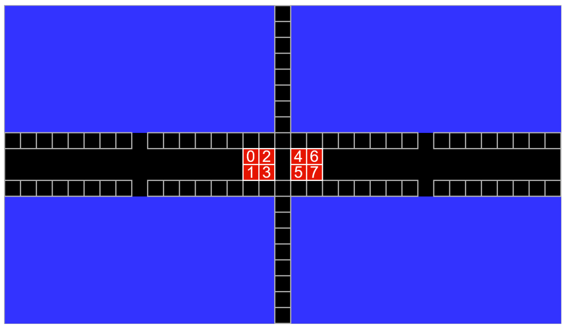
Figure 1: Comparison between actual environment and view observed by agent for each view method

Agents can only communicate with each other by observing each other's positions. They may use different types of views of the environment as inputs for their DQNs; therefore, they may view the environment differently. Regardless of the agent's view type, their visible area will be limited for tasks, walls, and other agents, which we refer to as the *agent's window* of observation. In this study, the agent's window of observation is a square of length N cells centered on the agent. For reference, Fig. 1a shows an example of the environment with the agent's window of observation when $N = 7$. The agent performing the observation is represented by the solid red square, and the agent's window of observation is represented by the red square outline surrounding the agent.

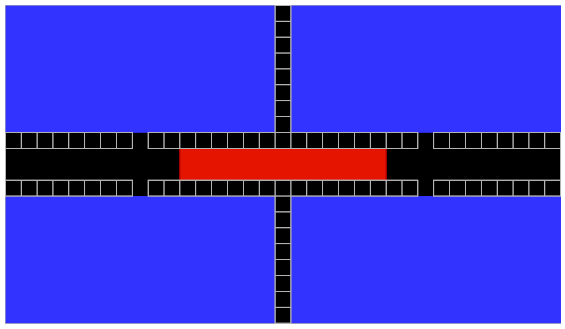
4.2 View Obstruction

We assumed that the agents could not view past walls in our environment and implemented a view obstruction feature for our agents. The view obstruction method employs a form of ray casting to determine the visibility region for each agent at every step.

To calculate the visibility region for an agent, we obtained the center point of the



(a) Experiment 1 Initial Environment



(b) Experiment 2 Initial Environment

Figure 2: Experimental Environment

agent, selected a wall visible to the agent, and constructed two straight lines from the agent's center point to each end of the selected wall. These two lines serve as the agent's cone of vision for the selected wall. Any unit of space behind the wall and within the agent's cone of vision is marked as "not visible". The same procedure is repeated for every wall within the agent's window of observation. Because our environment is a grid, our visibility region calculations resulted in many cases of partially visible cells. We defined a cell to be visible to an agent if more than 50% of the area of the grid space is within the agent's visibility region.

An illustration of this process for 1 agent is shown in Fig. 3 where we see images numbered 1 through 4. Before the agent's observation is used as an input to its DQN, the agent goes through the procedures illustrated in all 4 images to determine the cells that the agent cannot see. The solid red box in the center of all 4 images represents the agent at different stages of the calculation process. The red outline surrounding each image is the window of observation for the agent, and the empty gray boxes represent walls. We can see in image 2 that there are 2 green lines traced from the center of the agent intersecting 2 corners of the wall on the top right. Those 2 corners are the left-most and right-most corners of the wall from the perspective of the agent. Once the 2 green lines are drawn, we declare the space within the 2 green lines to be the agent's cone of vision when looking specifically at the wall on the top right. We can then determine which cells are behind the wall with more than 50% of the cell's area within the agent's cone of vision. The 3 white cells behind the wall meet this criteria and are marked as "not visible" to the agent. In image 3, we can see the same process being performed on a pair of walls to the bottom right of the image. Walls next to one another can be treated as 1 unit where the edges of the agent's cone of vision intersects the left-most and right-most corners of

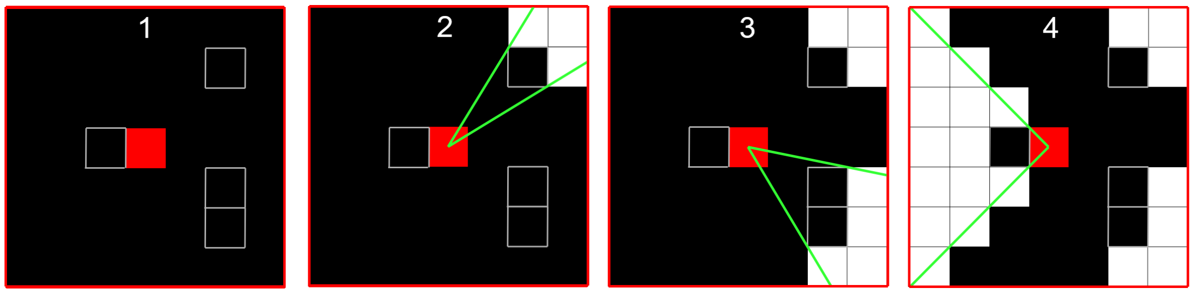


Figure 3: Ray Casting Example

the unit. In image 4 we can see the final wall being processed, and the agent's window of observation can now be used as input into the agent's DQN.

5 Proposed Method

5.1 Agent View Methods

Depending on the experiment, different aspects of the environment will be included in the agents' DQN inputs to investigate the effect of the agents' views on their coordination structure learned with DRL. We adopted two view methods: *local view* and *relative view* [13]. In addition, we employed a new view method called the *merged view*. Fig. 1 illustrates how an agent with each view type may view the environment. Red, blue, yellow, and white represent the observing agent, a task, another agent, and an invisible space, respectively.

The observations of the agents were encoded as a three-dimensional matrix containing values of either -1, 0, or 1. We call each slice of this matrix a channel. Each channel contains a specific aspect of the agent's observation, which can be the position of either the agent, other agents, tasks, or areas invisible to the agent because of a wall. Within these channels, an empty space is marked as 0, tasks and agents are marked as 1, and invisible areas or areas outside the environment are marked as -1.

5.2 Local View

The DQN input for agents with a local view is based solely on the details within the window of observation for each agent (Fig. 1b). The length and width of the DQN input are the same as those of each agent's window of observation. We divided the environment details within the agent's window of observation into three different channels for the DQN input. The first, second, and third channel encodes, respectively, the locations of other agents, tasks, and cells whose contents are invisible because the viewing agent's line of sight is blocked by walls. Furthermore, when an agent is near the edge of the environment, the agent's window of observation may overlap with a space that is outside the environment.

5.3 Relative View

The DQN input for agents with a relative view involves the window of observation for each agent as well as the outline of the entire environment (Fig. 1c). The length and width of the DQN input is the same as those of the entire environment. Within this DQN input, only the details of the environment within the viewing agent's window of observation are encoded. Areas outside of the viewing agent's window of observation are encoded as an empty space. The DQN input is separated into four channels. The first channel encodes the location of the viewing agent, the second channel encodes the locations of other agents within the viewing agent's window of observation, and the third channel encodes the locations of tasks within the viewing agent's window of observation. Finally, the fourth channel encodes the locations of cells whose contents are invisible because the viewing agent's line of sight is blocked by walls.

5.4 Merged View

The merged view involves two different DQN inputs for the same network. The size of one input is identical to that of the local view's DQN input, whereas that of the other input is identical to that of the relative view's DQN input. However, both inputs have two channels. For the local view input, the first and second channels contain information regarding the locations of other agents and tasks, respectively. For the relative view input, the first and second channels contain the agent's own position within the environment and the areas within the agent's view range that are invisible because of walls, respectively.

5.5 Neural Network Structure

We constructed our DQNs using the Keras library [1]. Tables 1, 2, and 3 show the layers and dimensions used for each model of the agent's view. We used RMSprop [19] as the optimizer of the model, and ReLu [16] for the activation function of our fully connected layer. The filter size was 2×2 for all convolutional and max pooling layers, and the stride was one and two for the convolutional and max pooling layers, respectively. The reward scheme for the agents was identical for all experiments; the agents began the episode with a reward value of 0; each agent's reward increased by 1 for each task executed during the episode.

5.6 Experience Replay

We employed experience replay [11], which has been shown to improve learning in DQNs. Experience replay is a technique that stores the experience of the agent at each step in a container called the *replay memory.* The replay memory saves experiences over multiple episodes, and when the DQN is to be updated, sections of the replay memory are selected by some criteria to be used as samples for the updating process. In our case, we use random sampling, which means the sections are randomly chosen.

Table 1: Network Architecture (Local View)

Layer	Input	Activation	Output
Convolutional	$7 \times 7 \times 3$		$7 \times 7 \times 32$
Max Pooling	$7 \times 7 \times 32$		$3 \times 3 \times 32$
Convolutional	$3 \times 3 \times 32$		$3 \times 3 \times 64$
Max Pooling	$3 \times 3 \times 64$		$1 \times 1 \times 64$
Flatten	$1 \times 1 \times 64$		64
FCN	64	ReLu	100
FCN	100	Linear	4

Table 2: Network Architecture (Relative View)

Layer	Input	Activation	Output
Convolutional	$35 \times 20 \times 4$		$35 \times 20 \times 32$
Max Pooling	$35 \times 20 \times 32$		$17 \times 10 \times 32$
Convolutional	$17 \times 10 \times 32$		$17 \times 10 \times 64$
Max Pooling	$17 \times 10 \times 64$		$8 \times 5 \times 64$
Flatten	$8 \times 5 \times 64$		2560
FCN	2560	ReLu	100
FCN	100	Linear	4

5.7 Epsilon Greedy Strategy with Decay

We adopted the ε-greedy strategy with decay, which attempts to obtain more rewarding behaviors that require investigating the environment through actions that initially appear to be not the most rewarding. During each step, each agent must choose between taking a random action or the action that's expected to maximize it's cumulative discounted reward. The probability for the agent to take a random action is determined by the ε value who's range is $0 \leq \varepsilon \leq 1$. These random actions are what allows agents to explore the environment. The ε value decrease over time with a predetermined decay rate $\gamma_\epsilon > 0$, where at the end of each episode, the new ε value for the next episode is calculated to be $\varepsilon_{new} = \varepsilon_{old} * \gamma_\epsilon$.

Table 3: Network Architecture (Merged View)

Input No.	Layer	Input.	Activation	Output
Input 1	Convolutional	$7 \times 7 \times 2$		$7 \times 7 \times 32$
	Max Pooling	$7 \times 7 \times 32$		$3 \times 3 \times 32$
	Convolutional	$3 \times 3 \times 32$		$3 \times 3 \times 32$
	Max Pooling	$3 \times 3 \times 32$		$1 \times 1 \times 32$
	Flatten	$1 \times 1 \times 32$		32
Input 2	Convolutional	$35 \times 20 \times 2$		$35 \times 20 \times 32$
	Max Pooling	$35 \times 20 \times 32$		$17 \times 10 \times 32$
	Convolutional	$17 \times 10 \times 32$		$17 \times 10 \times 32$
	Max Pooling	$17 \times 10 \times 32$		$8 \times 5 \times 32$
	Flatten	$8 \times 5 \times 32$		1280
Output	Concatenate	32, 1280		1312
	FCN	1312	ReLu	100
	FCN	100	Linear	4

Table 4: Parameters

(a) Learning Parameters

Parameter	Value
Discount Factor	0.90
Initial ε Value	1
ε Decay Rate	0.9998
RMSprop Learning Rate	0.0001
Memory Capacity	20,000
Mini-batch Size	32

(b) Experimental Parameters

Parameter	Value
Environment Width	35
Environment Height	20
No. of Agents	8
No. of Tasks	30
Reward	1
Episode Length (Steps)	300
Simulation Length (Episodes)	25,000

6 Experiment and Discussion

6.1 Experimental Setting

We conducted two experiments to analyze the performance and coordinated behaviors among agents using different DQN inputs in near identical simulations. Each experiment involved one simulation per view method. Each simulation involved eight agents performing a distributed task execution game for several episodes. The parameters for each simulation within an experiment was identical except for the view method. The parameters for these experiments are listed in Table 4.

6.2 Experiment 1: Static Spawn Location

In this experiment, we compared the behaviors and performances of agents using local and relative views. Each agent began at the same location for every episode, as shown in Fig. 2a. Each agent was assigned an ID number from 0 to 7. When the environment was initialized, agents spawn at the numbered location that matched with their ID. Consequently, each agent spawned in the exact same location at the start of each episode.

Fig. 4a shows the moving mean average of the total number of tasks completed by all agents during each episode, where the margin represents the standard deviation. As shown in the figure, as the episodes increased, more tasks were completed on average in the relative view compared with the agents with the local view; this may be because agents with a relative view can possess more information regarding their absolute locations.

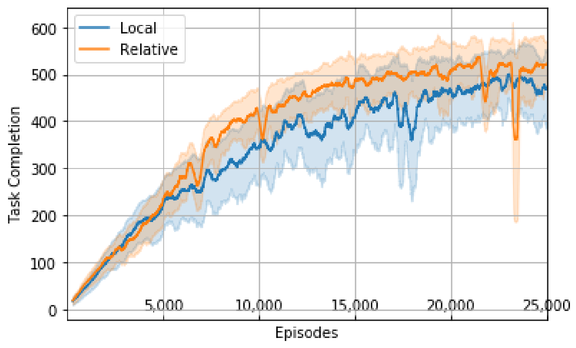
Figs. 4b and 4c are similar to Fig. 4a, except that these figures display the numbers of times the agents collided with a wall and with each other, respectively. As shown, the average number of wall collisions converged to approximately the same value for both the local and relative views; furthermore, the relative view had a slightly higher rate of collisions compared with the local view, although the performance (the number of completed tasks) of the latter was better than that of the former. Particularly, in Fig. 4b, the number of wall collisions occasionally spiked (e.g., around episodes 10,000, 22,000, and 24,000 episodes), and during these spikes, the performance deteriorated significantly. In addition, the standard deviation of agent collision numbers for the relative view increased from around episode 17,500 onward, although the standard deviation of agent collision numbers for the local view decreased to a small value. These results indicate that the behaviors of agents with relative views were relatively less stable.

Additionally, we counted the number of tasks completed by each agent at each area of the map over episodes 24001 to 25000. Figs. 5a and 5b show heatmaps of the tasks completed by each agent. As shown, for both the local and relative views, each agent completed tasks primarily in one room, i.e., the room closest to the agent at the start of the episode. It is noteworthy that two agents primarily completed the tasks in each room. Although there were two agents per room, the agents using the local view appeared to have completed tasks more evenly across the room compared with the agents using the relative view. Except for agents 1 and 3, agents using the relative view had one agent primarily complete tasks on one side of the room, while the other agent primarily completed tasks on the other side. This more detailed work division resulted in better performances when the agents used the relative view. Furthermore, we conducted the same experiment using the merged view; however, we discovered that the results were almost the same as those of the relative view.

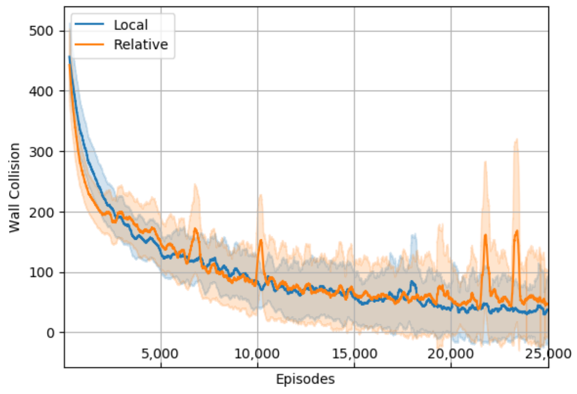
6.3 Experiment 2: Dynamic Spawn Location

In this experiment, we compared the behaviors and performances of agents using local, relative, and merged views. Agents would begin each episode at a random position within a 10×2 window at the center of the environment, as shown in Fig. 2b. We began this experiment using only simulations from the local and relative views. However, we discovered a significant performance loss from relative view; therefore, we introduced another view method called the merged view, which provided a higher performance than the local view.

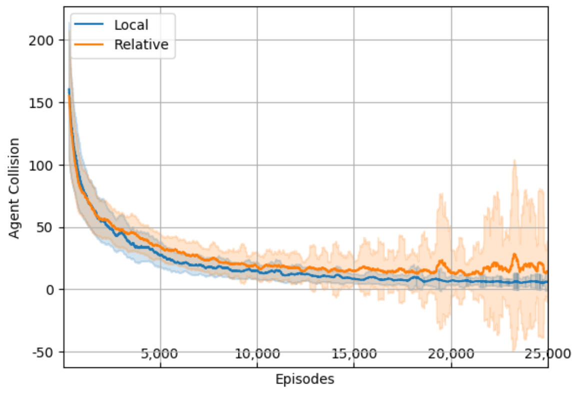
Similar to Experiment 1, Figs. 6a, 6b, and 6c show the moving mean average of the total numbers of task completions, wall collisions, and agent collisions, respectively. As shown in the figures, in terms of task completion number, the merged view was first,



(a) Total Task Completion Rate



(b) Total Wall Collision Rate

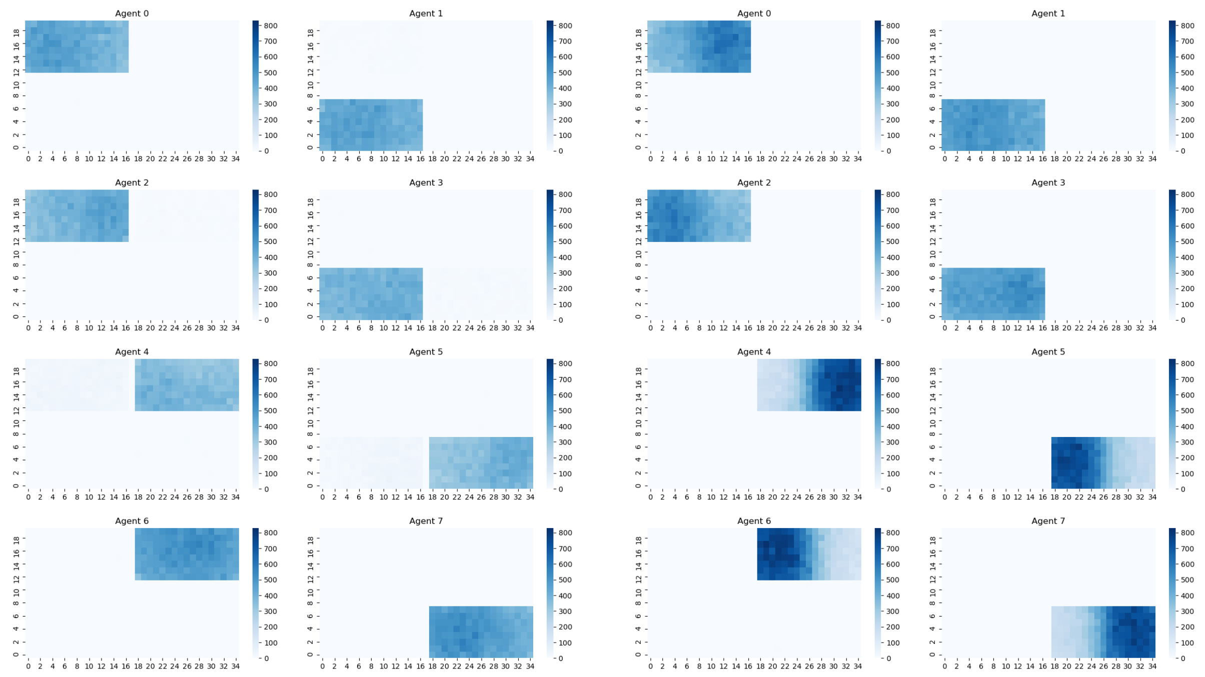


(c) Total Agent Collision Rate

Figure 4: Experiment 1 Scalars

followed by the local and relative views. However, the performances of these views were worse than those in Experiment 1. The number of collisions in the local view was higher than that of the merged view, and the wall collision rate decreased steadily over time in both view methods. Meanwhile, the wall collision numbers in the relative view fluctuated without any clear trend toward decreasing. Likewise, the agent collision number converged to approximately 0 for the local and merged views, whereas the agent collision rates for the relative view remained significantly high with the standard deviation increasing over time.

The heatmaps in Figs. 7a, 7b, and 7c show areas where each agent primarily completed their tasks in each simulation. Unlike the results from Experiment 1, the agents of all view methods indicated less preference for a single room. Additionally, the task completion areas within some rooms were not separated, as evident by agents using the relative or merged view in Experiment 1.



(a) Local View (b) Relative View

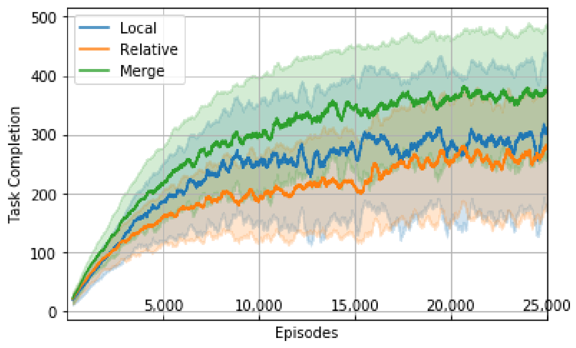
Figure 5: Experiment 1 Task Completion Heatmap.

6.4 Experiment 3: Static to Dynamic Spawn Location

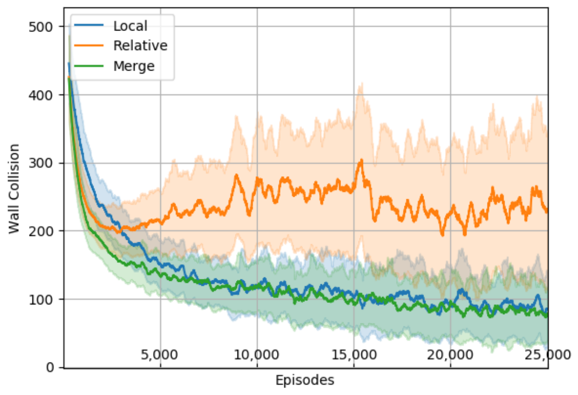
In this experiment, agents started from fixed locations as shown in Fig. 2a until episode 25,000 and subsequently started at random locations as shown in Fig. 2b for 10,000 episodes to investigate whether the behaviors learned in the former spawning configuration is applicable in the latter spawning configuration. Agents continue their episodes with the same parameters from episode 25,001 except spawning location and epsilon values. The epsilon value continues to decay with each episode, making its value lower than what it was before.

In like manner to past experiments, Figs. 8a, 8b, and 8c show the moving mean average of the total numbers of task completions, wall collisions, and agent collisions, respectively. We can see a drop in the amount of tasks completed and a rise in collisions shortly after episode 25,000 due to the sudden change in spawning locations for agents. Subsequently, we see total task completion and total collisions recover to a level similar to agents in Experiment 2; the only exceptions were the agents using relative views which showed a lower total task completion rate, a higher total wall collision rate, and a higher total agent collision rate.

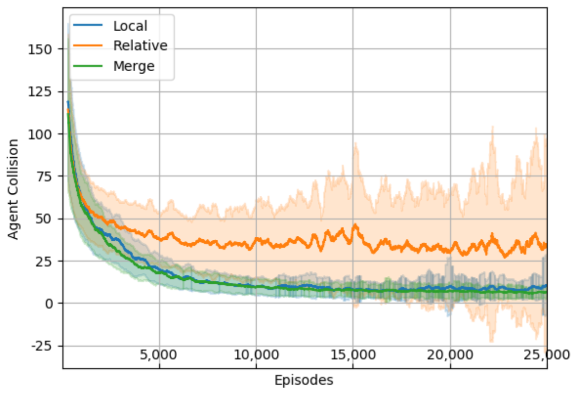
Unlike in Experiment 1, we included agents using merged views in an environment



(a) Total Task Completion Rate



(b) Total Wall Collision Rate

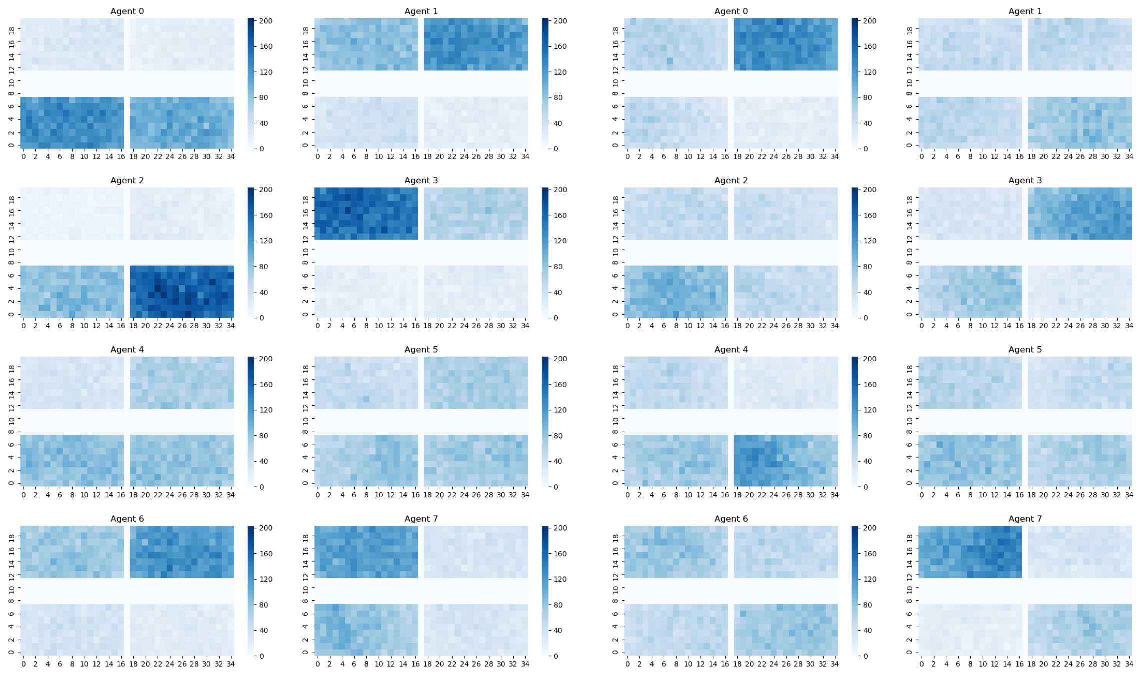


(c) Total Agent Collision Rate

Figure 6: Experiment 2 Scalars

with a static starting position. Given this new information, we can compare the performance of agents using relative views with agents using merged views. Looking at Fig. 8a, we can see that while agents were in the environment with static starting positions, the performance provided by merged views was near identical to that of relative views. Moreover, upon closer examination of episodes before 7,500, we can see that agents using merged views had a slightly higher total task completion rate, suggesting a small but noticeable performance superiority in the early stages of training. We also see that agents using merged views do not have the brief drops in performance and spikes in wall collision present in agents using relative views. This indicates a higher level of stability for agents using merged views.

Fig. 9 shows heatmaps of the areas where agents using merged views completed their tasks. We counted tasks over the same episodes (24,001 - 25,000) as Figs. 5a and 5b of Experiment 1. We can see that agents using merged views complete tasks primarily on one side of the room, although to a less significant degree compared to agents using relative views in Fig. 5b.



(a) Local View (b) Relative View

Figure 7: Experiment 2: Task Completion Heatmap

6.5 Experiment 4: Static Spawn Location With Four Agents

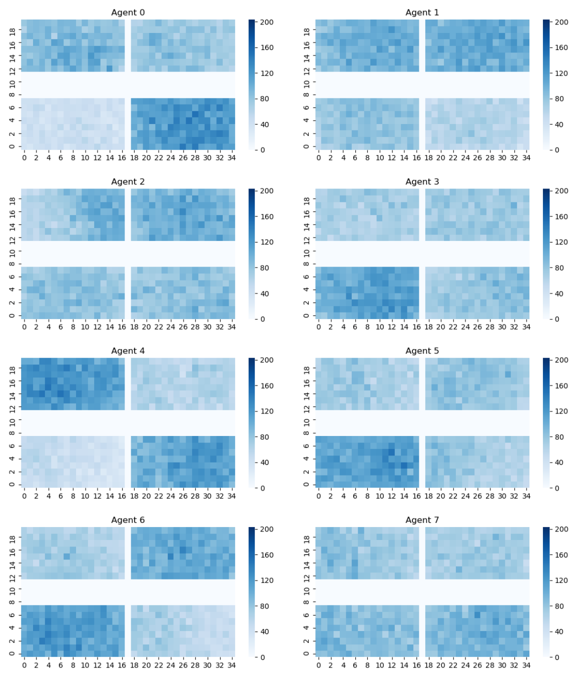
The aim of this experiment was to investigate the possible cause for the unstable performance of agents using relative views. To this end, the number of agents was reduced from 8 to 4. The 4 agents start from fixed locations in the spaces numbered 2, 3, 4, and 5 of Fig. 2a for the duration of 25,000 episodes.

Figs. 10a, 10b, and 10c show the moving mean average of the total numbers of task completions, wall collisions, and agent collisions, respectively. We can see that agents using local view and merged view increased in total task completion rate and decreased in total wall collision rate overtime while agents using relative view dropped in performance from around episode 17,500.

6.6 Discussion

6.6.1 Divisional Cooperation and Improved Performance

Several key insights can be inferred from the results regarding the behavior of agents using different view methods. First, based on the heatmaps from Experiment 1, we observed a form of divisional cooperation occurring among agents with local, relative, and merged views in terms of room selection. The agents would allocate the workload evenly by having

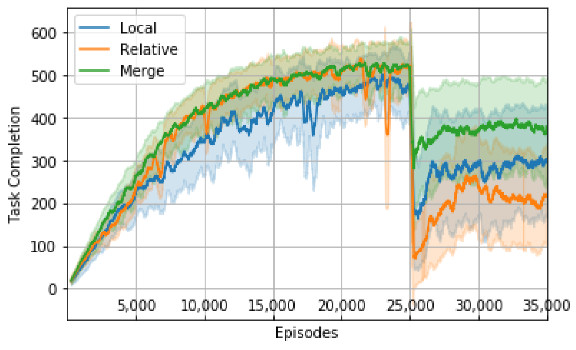


(c) Merged View

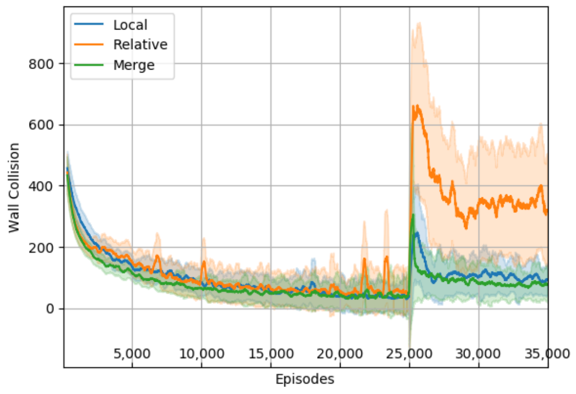
Figure 7: Experiment 2 Task Completion Heatmap (cont.)

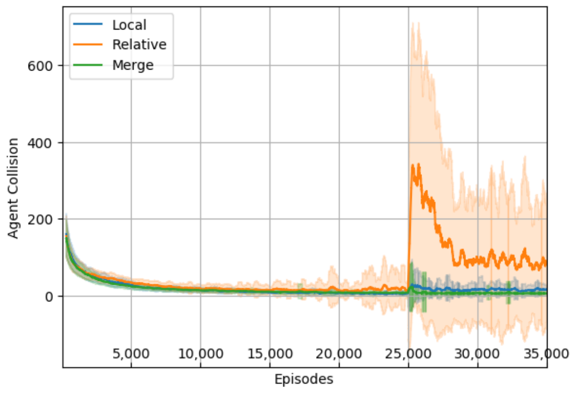
two agents visit each room. However, the relative view adds another level of divisional cooperation to the regime by having each pair of agents divide the room into halves. These findings are similar to those of Miyashita & Sugawara [13], where agents with a relative view would divide their 20×20 grid environment into different territories, but agents with a local view would not. When comparing our results with theirs, we can see that using local and relative view affects the agents' learned behaviors, i.e., because their environment consisted of a single room, agents with local views worked in only one or two specific rooms but did not divide a room into a number of subareas. On the other hand, agents with relative views worked in a room separately. This implication is consistent in both results. In addition to room separation, the relative view offers a considerably better performance than the local view in terms of task completion.

Furthermore, the heatmaps of Experiments 1 and 2 (Figs. 5 and 7, respectively) show that agents can provide significantly different behavioral outcomes in terms of divisional cooperation and room preference when spawning occurs in the same location as opposed to a different location at the start of each episode. Moreover, the line graphs of Figs. 4a and Fig. 6a suggest that the relative view's task completion numbers can be hindered significantly by the variable spawn location and high number of collisions. However, the high task completion numbers of the merged view suggest that it is not caused by



(a) Total Task Completion Rate





(b) Total Wall Collision Rate

(c) Total Agent Collision Rate

Figure 8: Experiment 3 Scalars

the agents observing their absolute location in the environment. This is because the agents using the merged and relative views observe the tasks, walls, and agents around themselves as well as their absolute locations in the environment. The only noticeable difference between the view methods is the method by which the observational information is input into the DQN, which may have contributed to the difference in the task completion number.

The difference in efficiency between these experiments can be explained partially using the heatmaps in Figs. 5 and 7. In Experiment 1, the same number of agents were in charge of each room, and a fairly adjusted coordination structure was obtained as a result of learning. Hence, the agents operated individually in the almost same-sized rooms. By contrast, in Experiment 2, the rooms to be visited were selected based on their initial locations; hence, the work became unbalanced. For example, three agents often operated in one room, while one or zero agents operated in another room, resulting in a decreased overall efficiency.

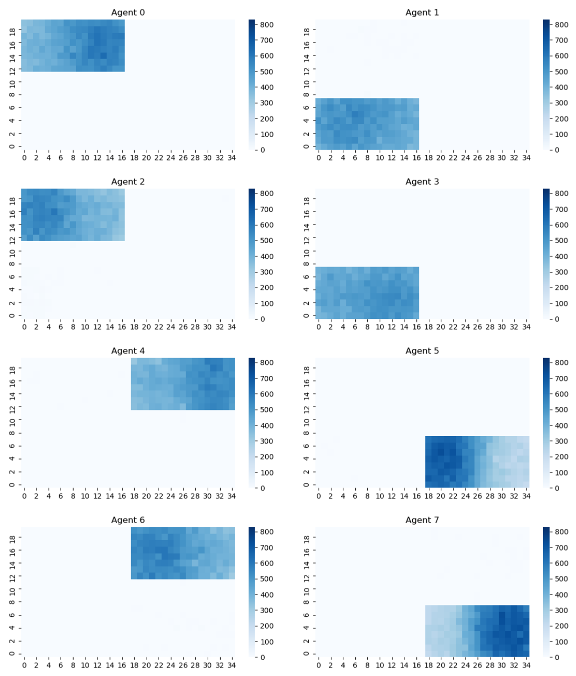


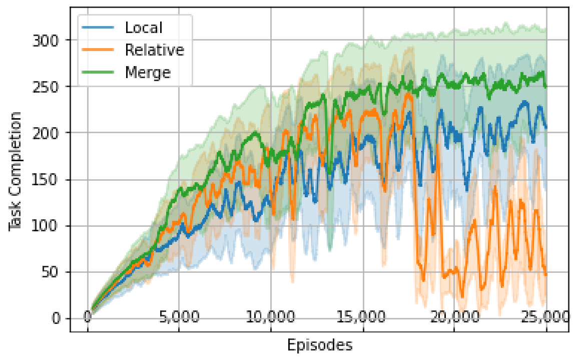
Figure 9: Experiment 3 Task Completion Heatmap (Merged View)

6.6.2 Robustness of the Merged View

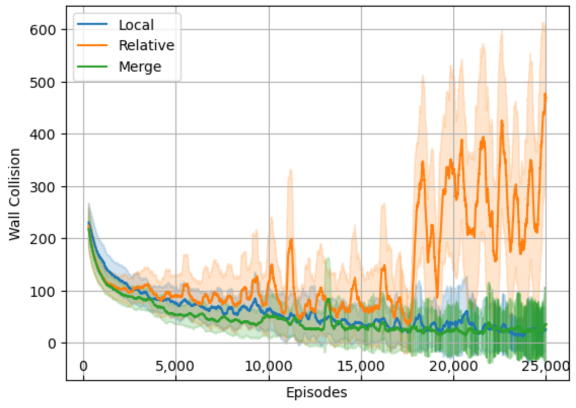
Data from Experiment 3 adds to the findings of Experiments 1 and 2 in several ways. Firstly, we can see in Fig. 8 that the performance of agents using merged views in a static spawn environment is near identical to that of agents using relative views. However, there is one key difference; agents using merged views have no performance drops and wall collision spikes that agents with relative views exhibit. Given that the only significant difference between the relative view and the merged view is the input structure, we can attribute the difference in input structure as the most plausible explanation for why the performance drops in one view method and not the other.

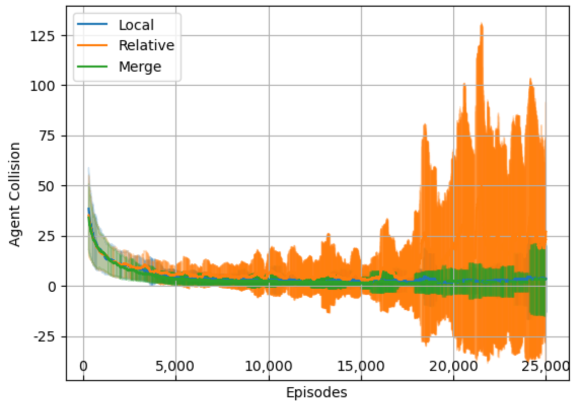
Secondly, when looking at Figs. 5a, 5b, and 9, we can see that room separation was also present in merged view when agents had a static spawn location, albeit to a lesser degree. The presence of territorial separation in the relative view and the merged view and lack thereof in the local view suggests that territorial separation can be present in view methods outside of the specific input structure of the relative view as long as information about the absolute locations of the agents is provided.

Finally, can see from how the agents adapted to the change in environment in Fig. 8 that the overall performance and number of collisions were almost identical to those in Experiment 2 except for the relative view, which performed worse overall. At least with



(a) Total Task Completion Rate





(b) Total Wall Collision Rate

(c) Total Agent Collision Rate

Figure 10: Experiment 4 Scalars

the local view and the merged view, their performance similarity to Fig. 6 of Experiment 2 indicates that the spawn locations significantly affected the agents' behaviors learned so far, and the agents reconstructed another coordination regime. In actual applications, both fixed and random spawn locations may be possible. Agents often have charging bases or storage locations. Thus, we think that fixed spawn locations are more likely to be present as the environmental setting.

6.6.3 Starvation by Other Agents

Although the difference in input structure between the relative view and the merged view appears to be the culprit behind the differences in performance, we attempted to investigate whether there were other factors contributing to the low performance exhibited by the relative view in Experiment 4. Given there were usually 2 agents for each room in past experiments, we considered the possibility that one agent may be completing too many tasks and starving the other agent in the same room of rewards. The starved agent may fail to learn to consistently visit one room, causing an imbalance in room occupancy, which may in turn lead to a decrease in overall performance. This may also explain the spikes in wall collisions because we have observed agents with learning issues repeatedly colliding with walls in the past. We can infer from 10 that having multiple agents in the same room may not be a significant factor contributing to the low performance of relative view. Given that there are only 4 agents, there must be 1 agent per room to avoid a build-up of tasks in any of the rooms. We can see that agents with relative views exhibited low performance and high wall collision starting from roughly episode 17,500. This finding strengthens the idea that the input structure of relative view may be the reason for the low performance, and not any environmental factors.

By understanding the coordination structures that we focused on in this paper, we have the advantage of being able to easily infer the impact on changes in the system. For example, since agents in Experiment 1 consistently visited one room, one room will be neglected in the event of an agent malfunction allowing identification of the malfunctioning agent. It is also easy to predict which rooms will be neglected due to a failure of a certain agent, which is especially useful in security patrol contexts where identifying areas with loose security is critical. Furthermore, we can see that the different starting positions between Experiment 1 and Experiment 2 result in different coordination regimes. This difference suggests that the starting positions of agents can have a significant impact on how they will behave. For this reason, we believe careful consideration should be taken to the starting position when increasing the number of agents within an environment. Finally, we can see that although the same environmental information is provided, the input structure can have a significant impact on the performance and stability of agents. For this reason, we recommend thought and deliberation should be taken when designing the input structure for an agent.

7 Conclusion

We analyzed the coordination structures observed in a multi-agent DRL setting, where 8 agents learned to play a task execution game in a four-room environment. We observed the effects of allowing agents to view their absolute locations within their environments on the manner in which they coordinated with one another as well as the differences arising from a consistent or inconsistent starting location for each episode. We also examined whether there was any transfer of learned behaviors between a consistent to inconsistent starting location for each episode, and we analyzed the performance when there were 4 agents in the environment.

Our experimental results indicated that with a consistent starting location, agents that can view their absolute location had highest task execution rates and territory divisions among agents sharing the same room. Furthermore, compared with agents with a consistent starting location, we discovered that agents with an inconsistent starting location behaved considerably differently in terms of performance and coordination. Finally, we introduced a new method to observe an environment that mitigated the performance loss from an inconsistent starting location while maintaining the agent's ability to view their absolute location in the environment.

Future studies may involve performing similar experiments with modified environmental parameters, such as the agent's view range, task spawn frequency, environment size, and asymmetrical environmental conditions. Further investigations could be made regarding the behavioral characteristics of agents using different input structures with the same environmental information.

References

[1] Chollet, Francois, et al. Keras. https://github.com/fchollet/keras, 2015.

[2] E. A. O. Diallo and T. Sugawara. Coordination in Adversarial Multi-Agent with Deep Reinforcement Learning Under Partial Observability. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, 2019.

[3] J. Fan, Z. Wang, Y. Xie, and Z. Yang. A theoretical analysis of deep q-learning. In A. M. Bayen, A. Jadbabaie, G. Pappas, P. A. Parrilo, B. Recht, C. Tomlin, and M. Zeilinger, editors, *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, volume 120 of *Proceedings of Machine Learning Research*, pages 486–489, The Cloud, 10–11 Jun 2020. PMLR.

[4] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks. *CoRR*, abs/1602.02672, 2016.

[5] J. K. Gupta, M. Egorov, and M. Kochenderfer. Cooperative Multi-agent Control Using Deep Reinforcement Learning. In *Autonomous Agents and Multiagent Systems*, pages 66–83. Springer International Publishing, 2017.

[6] M. Hausknecht and P. Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. *CoRR*, abs/1507.06527, 2015.

[7] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18) New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3215–3222. AAAI Press, 2018.

[8] G. Lample and D. S. Chaplot. Playing FPS Games with Deep Reinforcement Learning. *CoRR*, abs/1609.05521, 2016.

[9] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel. Multi-Agent Reinforcement Learning in Sequential Social Dilemmas. AAMAS '17, page 464–473, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[11] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, May 1992.

[12] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *CoRR*, abs/1706.02275, 2017.

[13] Y. Miyashita and T. Sugawara. Cooperation and Coordination Regimes by Deep Q-Learning in Multi-agent Task Executions. In *Artificial Neural Networks and Machine Learning – ICANN 2019: Theoretical Neural Computation*, 2019.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. *CoRR*, abs/1312.5602, 2013.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, feb 2015.

[16] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[17] D. Portugal and R. Rocha. A survey on multi-robot patrolling algorithms. In L. M. Camarinha-Matos, editor, *Technological Innovation for Sustainability*, pages 139–146, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[18] A. Sugiyama, V. Sea, and T. Sugawara. Emergence of divisional cooperation with negotiation and re-learning and evaluation of flexibility in continuous cooperative patrol problem. *Knowledge and Information Systems*, 60(3):1587–1609, Dec. 2018.

[19] T. Tieleman and G. Hinton. Neural Networks for Machine Learning - Lecture 6a - Overview of mini-batch gradient descent. 2012.

[20] H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. *CoRR*, abs/1509.06461, 2015.

[21] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1995–2003. JMLR.org, 2016.

[22] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, May 1992.