Universiteit Leiden

Master Computer Science

Multi-Agent Cooperation and Information Exchange in Partially Observable Games

Name:	Hainan Yu
Student ID:	S2505762
Date:	31/07/2021
Specialisation:	Data Science: Computer Science
1st supervisor:	Dr. Mike Preuss
2nd supervisor:	Dr. Tessa Verhoef

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

As robotic agents become widespread in our daily life these years, inter-agent and agent-human cooperation remain a challengeable problem. Combining multi-agent reinforcement learning and evolutionary linguistics, we contemplate the robotic agents can take advantage of the messages sent from partners to perform better in partially observable games. To examine the assumption, we divide this goal into four subtasks, containing the experiments on different levels of information to adaptive information exchange, and partly conduct experiments. Our experimental results show the agents cooperation emerges on sufficient observations of both Atari *Cooperative Pong* and card game *Thegame* environments. Meanwhile, *Thegame* agents with hard-coded fuzzy information exchange perform much better than partial information. These results indicate the potential of additional information to aid agent training with limited observation and show the possibility for flexible information expression learning, which we set as our future work.

Contents

1 Introduction

Cooperation commonly happens in human society. According to the specific conditions and purposes, humans can freely form teams, cooperate with teammates, and communicate within teams to tackle tasks that are very difficult or even impossible for a single person or a group without neat coordination. With the development of science and technology, robotic agents are gradually playing a significant role in various fields to assist humans. The scenarios of humans and robots working together or multiple agents working as a team are also getting popular.

This raises a meaningful question: is it possible to form a human-agent team or even a pure agent-agent team for the aforementioned tough tasks which each participant cannot complete on its own? How do agents share valuable information with their teammates to cooperate, similar to human team collaboration and communication in related tasks? Agents may find a suitable communication mode for the current task via interactions with each other. Will this also help us learn how human communicative language emerges during cooperation?

We start with two collaborative games: *Cooperative Pong* and *Thegame* [25]. We first try to examine if multi-agent cooperation is feasible with relative complete information in such environments. We deploy the DQNs model [24] and check whether the agent can cooperate with the partner and how the model performs. Both experiments show agents can cooperate with the partner using deep reinforcement learning algorithms.

Taking inspiration from Steels' Language Game [35, 34] method and corresponding behavioral cognitive basics [8], we assume the agent may utilize language signals to coordinate the task. The adaptive language for a specific task may emerge during their interactions. Like in Language Game experiments, the communication signal needs to have a direct impact on the game score. We design a hard-coded communication signal for *Thegame* to examine its importance. The experimental results show the communication signal a positive role in the agents' cooperation. This implies we can leverage the potential of the communication during the cooperative task. This could influence how we build a human-agent team based on possible communication or information exchange.

In next chapters, we are going to first review the requisite background in section 2; second, we describe the approach we design to explore how cooperative agents work in partially observable game and the potential of information exchange in section 3; third, we list the experimental environments and setups in section 4 and show the experimental results in section 5; fourth, we discuss the possible improvement and future plan in section 6 then conclude the work of this thesis in section 7.

2 Background

In this chapter, we provide the relevant background knowledge that helps go through the thesis. The content involves classic and deep reinforcement learning, multi-agent system, agents relations in team works, and evolutionary linguistics.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a type of computational approach that optimizes the state-action pairs to maximize reward returns during interacting with the environment [38, 39]. Sutton and Barto define RL problem leveraging Markov Decision Process (MDP) with three elements: states, actions, and goals with returns. It implies the formalization tuple for MDP: $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ stands for state space, $\mathcal{A}$ for action space, $\mathcal{P}$ as the probability of state-action transition, and $\mathcal{R}$ for instant rewards. Discount factor γ is used to calculate the discounted final return because we usually regard the future reward that is closer to the present are more significant than the later ones. This definition indicates the essential features of MDP: the learner, as known as the agent, observes the environment, takes action based on the observation, and tries to achieve the final goal. Based on the current state $s_t \in \mathcal{S}$ at time step t, the agent makes decision to select an action $a_t \in \mathcal{A}$. After the action taking, the state s_t transits into $s_{t+1} \in \mathcal{S}$, and the agent obtains a instant reward $r_{t+1} \in \mathcal{R}$ from the environment (here we adopt the representation similarly from Sutton and Barto [39]). In this case, what the agent needs to do is not always select an action that brings the maximum instant reward. Instead, it selects actions step by step that could lead to the maximum final return $G = \sum_{i=t+1}^{T} \gamma^{i-t-1} r_{i+1}$.

Markov decision process has an important property: the state and reward are decided only by the closest former state-action pair. Mathematically, the probability p of the state s_t at time step t and corresponding reward r_t is defined as follows:

$$p(s', r \mid s, a) \doteq \Pr\{s_t = s', r_t = r \mid s_{t-1} = s, a_{t-1} = a\}, \tag{2.1}$$

where s, s', r and a denotes a random state, next state, reward, and action, respectively.

Meanwhile, we can calculate the expected reward using the state-action pair by this way,

$$r(s,a) \doteq \mathbb{E}[r_t \mid s_{t-1} = s, a_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a). \tag{2.2}$$

In some cases, we also make use of *state-action-next_state* pair to determine the reward, especially when the agent cannot obtain the true next state immediately based on its turn state-action pair in the multi-agent turn-based scenario. Here the reward is represented as,

$$r(s,a,s') \doteq \mathbb{E}[r_t \mid s_{t-1} = s, a_{t-1} = a, s_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}. \tag{2.3}$$

Based on the updating approaches, we can categorize RL algorithms into two types: value-based (learning the value, i.e., expected return from the state-action pair) and policy-based (learning to update the policy directly). There are also methods combining these two types, e.g., Deep Deterministic Policy Gradient (DDPG) [18], which makes use of a critic network (value-based) with an actor-network (policy-based). According to the policy used during learning, there are on-policy methods (e.g., policy gradient) and off-policy methods (e.g., Q-learning).

2.1.1 Deep Q-Learning

Deep Q-networks (DQNs) are proposed by Minh et al. in their seminal work *Human-level control through deep reinforcement learning* [24]. It is the combination of a classic reinforcement learning Q-leaning and functional approximation with deep neural networks. To understand what DQN is doing, we can first go through the tabular Q-learning.

The classic tabular value-based reinforcement learning has two directions: 1) dynamic programming that exploits bootstrapping and full information of the transitions (model), 2) Monte Carlo method that uses the real final return to predict the value. Temporal Difference (TD) unites the advantages of these two methods: it bootstraps with updating the value prediction by the way Monte Carlo methods do, i.e., calculating the difference of estimated values and real returns based on the experience. TD methods do not need the final returns. Instead, they bootstrap with the value of the next step as dynamic programming does.

Generally, with the help of the Bellman equation in dynamic programming, the expected return (i.e., the value v) under policy π is defined as follows:

$$v_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})|s_t = s], \tag{2.4}$$

where $v_\pi(s)$ denotes the value of a certain state s under the policy π.

Similarly, the estimated q-value of a state-action pair under policy π turns to be:

$$Q_\pi(s, a) = \mathbb{E}_\pi[r + \gamma \max_{a'} Q_\pi(s_{t+1}, a')|s_t = s, a_t = a], \tag{2.5}$$

where the Q_π indicates the q-value under the policy π.

Accordingly, the optimality Bellman equation for the optimal policy π_* is set as $v_*(s) = \max \mathbb{E}[r_{t+1} + \gamma v_\pi(s_{t+1})]$ for the value and $Q^*(s, a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a')]$ for q-value. Utilizing the iterative Bellman equation as an updating manner, the value and q-value of of policy π_{i+1} for $i + 1$th iteration updating can bu calculated as $v_{i+1}(s) = \max \mathbb{E}[r_{t+1} + \gamma v_i(s_{t+1})]$ and $Q_{i+1}(s, a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q_i(s_{t+1}, a')]$, respectively. As $i \to \infty$, the policy π_i will converge to the optimal policy π_*.

Sutton and Barto [39] give the simplest TD a Monte Carlo style updating approach as follows,

$$v(s_t) \leftarrow v(s_t) + \alpha[r_{t+1} + \gamma v(s_{t+1}) - v(s_t)], \tag{2.6}$$

where the $v(s_t)$ and $v(s_{t+1})$ indicates the expected value of state s_t and s_{t+1}, and α is the learning rate. This method is also known as TD(0) because it only looks one step reward forward.

Q-leaning [43] is a value-based off-policy TD method. It updates the q-value as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.7}$$

This method makes use of ϵ-greedy policy to select actions. It means, during action selection, the agent has a probability ϵ to take a random move. Otherwise, it will comply with the estimation of state-action values on the table. As an off-policy method, it uses the maximum $q - value$ of the next state (i.e., $\max_a Q(s_{t+1}, a)$) to update the $q - value$ of the current state-action pair.

Since most RL tasks in our real life are difficult to train (e.g., due to the long run and enormous possible states), we cannot always use tabular methods on them. The deep Q-network upgrades the classic Q-learning with non-linear functional approximation.

Recall the Bellman equation for q-value (see Eq. 2.5) and the update method for classic Q-learning (Eq. 2.7): we need the complete episodes of agent trails and errors from the environment as $i \to \infty$ to obtain a π_i that converges to the optimal policy π_*. Whereas this is unrealistic for most real-life tasks. DQN uses deep convolutional neural networks as a functional approximation of the value function. To compensate for the instability of non-linear approximation, DQN utilizes experience replay [19] and a target network for $q - value$ updating to reduce the impact of correlation.

Nonetheless, DQN is still a simple model and has some defects in terms of stability, such as over-optimistic estimated Q-value, sampling inefficient. There are models, e.g., Double DQN [13], Deuling DQN [42], Rainbow [15], as enhancements for DQNs.

The usage of DQN involves a few important hyperparameters tuning and settings, varying from task to task. We will elaborate on the details we use in our experiments in chapter 4.

2.1.2 Proximal Policy Optimization

Proposed by Schulman et al. [31], proximal policy optimization (PPO) belongs to policy gradient (PG) in the RL methods family. It is an enhancement from the pure PG from Mnih et al. [23] and trust region policy optimization (TRPO) by Schulman [32].

Different from value-based methods updating value function to select the action, the policy-based method policy gradient updates the policy $\pi(a|s;\theta)$ directly. Due to this reason, PG is also more suitable for tasks with continuous action space. The basic PG makes use of the objective

$$L^{PG}(\theta) = \hat{\mathbb{E}}[log\pi_\theta(a_t|s_t)\hat{A}_t], \tag{2.8}$$

which is derived from the policy estimator

$$\hat{g} = \hat{\mathbb{E}}[\nabla_\theta log\pi_\theta(a_t|s_t)\hat{A}_t], \tag{2.9}$$

where the $\hat{A}_t$ is the advantage function $\hat{A}(s_t, a_t) = Q(s_t, a_t) - v(s_t)$, and π_θ is the stochastic policy. $\hat{\mathbb{E}}[\cdot]$ indicates the estimated expectation with the given parameters. Both $Q(s_t, a_t)$ and $v(s_t)$ are exactly what we use in the explanation of value-based TD methods.

TRPO uses probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ in the objective and the updating range limitation penalty (e.g. KL divergence) to prevent the policy gradient from updating with big $r_t(\theta)$. Whereas the penalty approaches are much difficult and less efficient for

training, PPO makes use of a simpler clip manner to achieve to similar or even better performance. In our experiments we exploit PPO with the *RLlib* package [17] using the default parameters. The details will be described in the chapter 4.

2.2 Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) is a series of RL methods that tackle tasks involving multiple agents. In other words, compared with single-agent reinforcement learning that is discussed more frequently in the RL industry, MARL has an agent space $\mathcal{N} = \{1, ..., N\}$ with the cardinality $n = |\mathcal{N}|$ (i.e., there are $n = |\mathcal{N}|$ agents involved in the task).

Gronauer and Diepold [12] summarize the formalized definition of (simultaneous and joint) multi-agent (MA) MDP from Littman [20]: similar as the formalization tuple for single-agent MDP in Chapter 2.1, the tuple of MA MDP is set as $(\mathcal{N}, \mathcal{S}, \{\mathcal{A}\}^i, P, \{\mathcal{R}\}^i, \gamma)$, where i indicate the index of a participating agent. Compared with general MDP, the definition of MA MDP distinguishes from a few points: 1) state space $\mathcal{S}$ stands for the set of states that consist of the states from each individual agent in the task; 2) action space $\mathcal{A} = \mathcal{A}^1 \times \cdots \times \mathcal{A}^N$ represents the joint actions taken by each agent at the same time step; 3) reward space $\{\mathcal{R}^i\}$ is the connected reward of all agents after taking the joint actions.

For MDP, we usually make use of "fully observable" which indicates the agent is aware of all information of the state to compare with "partially observable" which indicates the agent can only get access to a part of the state. Here we use observation space $\mathcal{O}$ collected from individual observation spaces $\{\mathcal{O}^i\}$ for agent i to indicate the limited visible zone in partially observable environment [12]. Thus for an agent, the input becomes 'observation' rather than 'state', and the MDP converts into partially observable MDP (POMDP) [16]. Despite the solutions such as *Nash equilibrium*, each agent taking action still makes MARL suffer from the complex simultaneous learning process and the non-stationary environment dynamics [5, 44].

Training and Execution The term *training process* indicates the learning process when the agent makes use of the environment information (e.g., observation), and execution denotes the process of action selection, according to the definition in Gronauer and Diepold's MARL survey [12]. Both training and execution can be centralized or decentralized. In decentralized training, each agent can only get access to its own observable zone and update based on the local information, while centralized learning allows the

agent to get additional information from its partner(s). During the execution process, centralized execution refers to using one joint policy to select associated actions for all participating agents. By contrast, decentralized execution means that each agent uses its own policy to choose the independent action.

The overall training and execution procedure has three types: [12]:

- Decentralized training and decentralized execution. Each participant trains the independent policy with the local observation that is usually partial and limited compared with the full information of the state. The agents then select their actions themselves;
- Centralized training and centralized execution: agents are trained by one central joint policy. The policy selects actions for each agent and based on the full information of the state [5];
- Centralized training and decentralized execution: each agent trains its policy using additional information (e.g., centralized value function [21]) by information exchange that will be discarded during testing, and select the action for itself.

2.3 The Agent Relation in Multi-Player Cooperative Game

There should be a certain relationship between the participants in multi-agent tasks. For instance, competitive, cooperative, or mixed relation (competitive and cooperative co-existing). From the perspective of cooperation (video/board) games, there are two kinds in general:

- Cooperative and competitive game: containing more than one team. The agents inside one team collaborate. Competitions happen among different teams. Multiplayer games that need to battle for a winning team belong to this type, e.g., multiplayer online battle arena[3] (MOBA) games (see Figure 2.1a);
- Pure cooperative: usually there is only one team involved. Agents cooperate with their partners in the task, e.g. building mode Minecraft; cooperative Platform game (see Figure 2.1b); card game Hanabi [2, 27], Thegame [25].

[1]The screenshot attributes to Wikipedia: https://en.wikipedia.org/wiki/League_of_Legends

[2]The screenshot attributes to *BlockBattle Theater* official Steam store: https://store.steampowered.com/app/238460/BattleBlock_Theater/

[3]In MOBA games, two teams compete for a winner.

(a) League of Legends

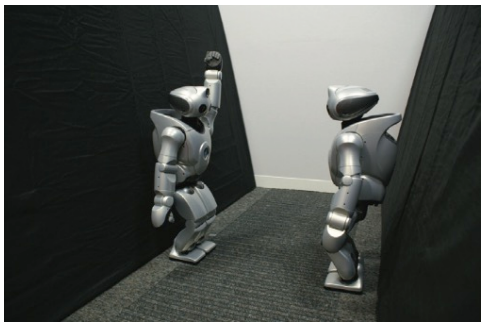
(b) BlockBattle Theater

Figure 2.1: Instances of MOBA game and Platform game. (a): A Screenshot of MOBA game *League of Legends*. There are two teams - red team and blue team - competing, two players in each team involved in this screenshot[1]. (b): A Screenshot of platform game *BlockBattle Theater*. One team with three members is involved in this screenshot[2].

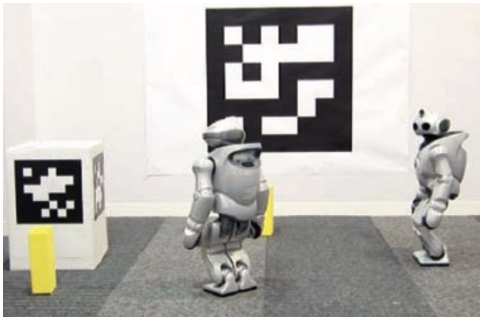
No matter in which type of cooperation game, the agents within the same team have one common goal to achieve, where certain strategies of the agent coalition might emerge. Although in real-life cases it also could be competitive among teammates, we only consider cooperation for simplicity here.

2.4 Language Game

Language Game is a paradigm for agent-based modelling of language evolution in Evolutionary Linguistics (EL) purposed by Steels [35, 34]. Inspired from Wittgenstein's *Language Game* [45], it focuses on agents language evolution and consensus emergence during interactions in the same generation (horizontal transmission) [3].



(a) Action Game



(b) Quantifier Language Game

Figure 2.2: (a): Two embodied robots playing Action Game [36]. The speaker robot (right) requests the hearer (left) to raise the left arm. (b): The interaction process of two embodied robots for quantifier emergence [29]. Both of them face several boxes (square-shaped with black and white colors) and blocks (small yellow cuboids).

Steels points out the meaning of a word or phrase emerges from interactions, and a word may have different meanings in different situations. To accomplish the negotiation task that contains words with ungrounded meanings, the participants have to adjust the communication signal to make the consensus of expressions [34]. For example, the meaning of a body gesture verb can be clarified during the Action Game [36] played between the speaker (who speaks a body gesture verb to request a body gesture) and hearer (who receives that word and performs a gesture), shown in Figure 2.2a.

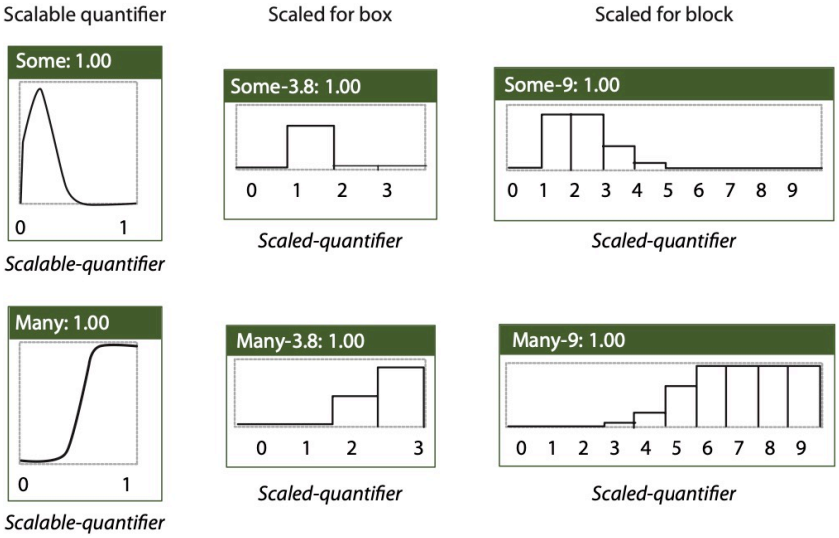


Figure 2.3: The scalable quantifier in Pauw and Hilferty's work [29].

Another instance is the emergence of quantifiers proposed by Pauw and Hilferty [29]. A certain number of boxes and blocks are placed in front of two embodied agents (see Figure 2.2b). During interactions, the speaker makes use of fuzzy quantifier (e.g. *some* and *many*, etc.) [46] to describe a scene out of two. The hearer needs to interpret the meaning of fuzzy quantifiers and point out its answer to the speaker. Based on the feedback given by the speaker, the hearer modifies its interpretations. Since the meaning of fuzzy quantifiers changing in different scenarios, e.g., *many* can describe *two* when the object norm is 3, but cannot be convincible when the norm is 10, a normalization scale of fuzzy quantifier can be learned for different norms (see Figure 2.3). In the same way, agents can even invent quantifiers and make a consensus of the usage of newly invented words [29].

3 Approach

In this chapter, we are going to elaborate on our idea and approach in detail. The central issue we are concerned about is how to help the cooperative agents work together in partially observable environments. As learning tools, the multi-agent reinforcement learning methods have shown their potential for the agent training in the cooperative tasks [9, 40, 7], whereas the partially observable collaborative environments are still hard to be solved using basic RL methods [14]. We find evolutionary linguistics and Steels' *Language Game* in particular affords an implementation approach for multi-agent collaborative negotiation: the agents can learn how to express the observation and interpret the communication signal themselves throughout the course of interactions. Reinforcement learning methods fit this idea very well: the agent can act (game playing, cooperating, and information exchanging), collect rewards (e.g., environment rewards, communication feedback from partners), then update its policy. Therefore, we assume the information exchange can assist agent cooperation, and the exchanged signals can be adapted to different environments and become more efficient through this trial and error progress. To verify our assumption step by step, we divide it into four subtasks in practice:

1 Train the cooperative agents with sufficient (relatively full) information;

2 Train the cooperative agents with partial information;

3 Add hard-coded fuzzy signal;

4 Train learnable fuzzy signal.

First, we check whether the cooperation emerges using RL methods on sufficient information environments (information provided as full as possible) for multiple agents. Compared with the mathematical definition in MDP, we regard our sufficient information as the information adequate to a certain extent that can be used for training. It is out of the considerations that 1) the emulated environments could provide global details

that contain all information we need, 2) it is not restricted within "full information" rather "sufficient", because we want our task settings as realistic as possible. Second, we exploit the same algorithms in the partial information environment, where the agent can only get access to its limited observation without information from partner(s). Third, we design a type of hard-coded fuzzy information that can be sent from an agent to its partner(s) and examine whether this kind of exchangeable message helps agents learn. Fourth, the agents are endowed to learn how to use appropriate fuzzy expressions according to the specific situation they meet.

This order is dependent on the rationale as follows: to evaluate whether agents can learn a way to cooperate and get help from partners, we need first examine whether the RL methods work for the target cooperation task, and second, analyze the role of possible communication. Regarding the first problem, sufficient information of the state is the best for training, because it serves nearly everything the agents have to know, e.g., the position and speed of every member. But this plentiful information is usually impossible to obtain. Compared with it, partial information is easy to get, whereas hard to train. Therefore, we utilize the fuzzy signal that acts as the tunnel for communication when full information is hard to get for every team member, and partial information is neither enough nor efficient for training. If sufficient information observation works better than partial information, we can use the handcrafted fuzzy signal to assist the agent get useful messages from partner(s) and ease the training. If the fuzzy signal helps the training process, we may expect to set a neural network for agents to learn how to express the situation and interpret the partner's messages based on the concrete circumstances that are more flexible for communication. For example, the agent may learn the scale of the word *small* as an indicator for the number size to describe how large a number is more precisely.

To have a clear view, imagine two players Alice and Bob are playing a number game (see Figure 3.1): there is one card placed on the table, and Alice and Bob also have one card in their hands, respectively, within the range $(0, 100)$. Their goal is to decide the order of these three cards. How could they do?

This game is cooperative: 1) the game cannot be played by only one of the players; 2) two participants need to help each other to achieve a better result (to figure out the numbers' order), where communication plays its role. There are two direct situations: 1) If they do not talk, the solution is a random guess; 2) If they can talk the exact numbers, the task is solvable one hundred percent. However, usually, we are seldom permitted to talk about the exact numbers in real-life tasks but instead allowed with limited com-

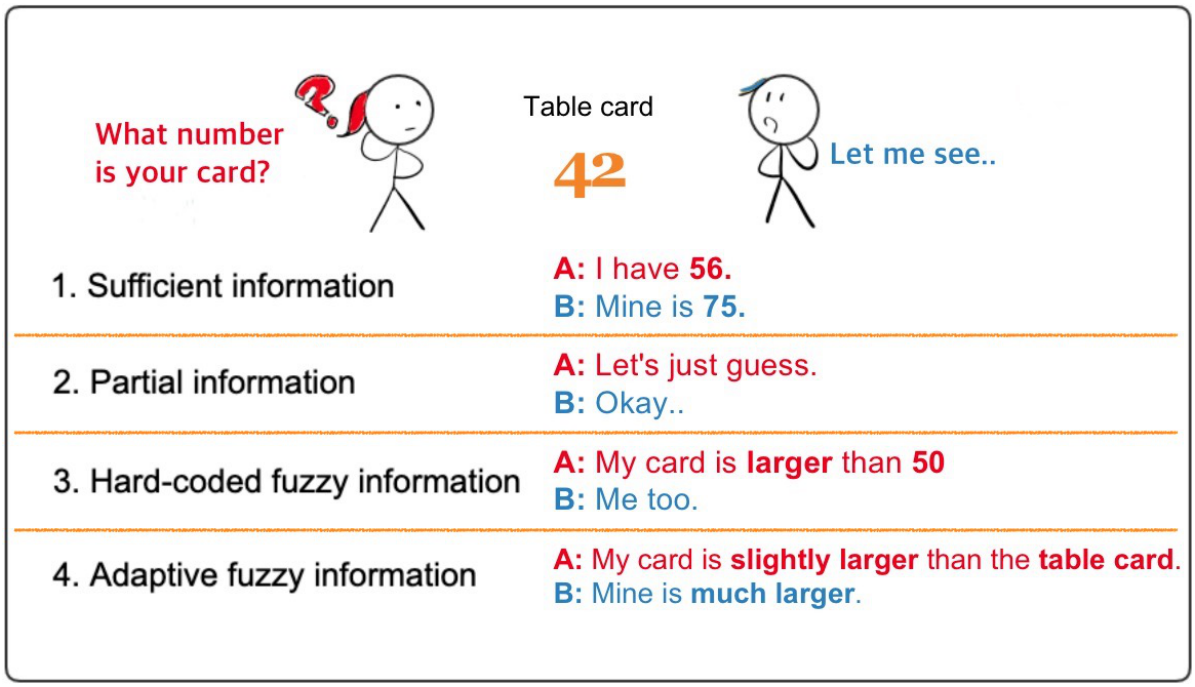


Figure 3.1: Possible dialogues of Alice (red text) and Bob (blue text) playing *number sorting* under environments of subtask 1 to 4. *Number sorting* is a simple cooperative task: two players are sorting three cards: their hand cards and one table card. Ideally, they want to know the exact numbers of cards. If that is not allowed, they need to know how large each hand card is. In this circumstance, communication is necessary to help them play better than a random guess.

munication. It leads to this situation: 3) they could talk vaguely with the hard-coded messages: if one hand card is smaller than 50, it is the smallest. Otherwise, it is the largest. Expectable, by this way, if the table card is 50, and two players' cards are on both sides of the table number, they can perform as well as exact number exchange. But if the table number changes or their cards lie on the same sides, this information might get useless. There is a more efficient way for the vague talk: 4) communicating based on the current situation using scalable expressions. During interactions, the players can learn the card number range and realize how large their cards are, compared with the table card. Their perception of the number is no longer limited to the hard-coded rules but becomes adaptive according to the situation, i.e., the table card and messages and feedback from the partner. If two hand cards are larger than the table card, they may express how large the number is in different intensities (see Figure 3.1). When the table card changes, Alice and Bob can use the same scale to describe their cards. Meanwhile, compared with hard-coded information, this 'learn expression by interactions' approach is more flexible and convenient to extend to other similar tasks, instead of handcrafting

new rules for information exchange every time. It reflects our assumption of communication emergence: during training, agents could learn how to describe their observations (e.g., the relative position of a target or agent, or relative size of a number) based on their knowledge and the current situation.

These four situations correspond to the four subtasks we design in this section. In particular, the procedure of subtask 3 to 4 follows the 'scalability' conception in *Language Game* evolution and Pauw and Hilferty's scalable quantifier [29]. We will describe the experimental setups for different environments in section 4.

4 Experiments

In this chapter, we describe the experimental design. We first discuss the game environments: what are the criteria, how do we customize the existing game or create the emulator for new games that fit our goal and subtasks? Second, we list the model and hyperparameter setups in detail for two selected environments.

4.1 Environments

According to our goal described in chapter 3, we need to carry out the experiments in cooperative environments with multiple agents. Thus, the game environments selected complies with the following criteria:

1. Player size ≥ 2 (i.e. multiple players);
2. Cooperative game. The participants in it help each other to achieve the final goal;
3. The environment provides sufficient information in terms of the state, and can be cast as partial observation as well;
4. The game allows communication to a certain extent.

We select two games: *cooperative Pong* and *Thegame* as the experimental playgrounds.

Figure 4.1: Left: Atari *Pong*. Right: NSV *Thegame*.

4.1.1 Cooperative Pong

Cooperative Pong is a two-player adaption of Atari 2600 *Pong*, proposed by Tampuu et al. [40]. In this game, two players (left and right) form a team. Each player holds one racket and keeps passing the table tennis ball. During playing, they need to prevent the ball from flying outside of the frame. There are four actions for one player: *UP, DOWN, STAY,* and *FIRE*. Tampuu et al. neatly design a reward function (see Tab. 4.1) for cooperation emergence: if one player loses the ball, both players get a -1 reward, and the game turns to be *done*; if not done, then every time step, both players get 0 points.

	Left player scores	Right player scores	No one scores
Left player reward	-1	-1	0
Right player reward	-1	-1	0

Table 4.1: Reward scheme of *Cooperative Pong* [40].

Before the experiment on the two-player environment, we first adopt the single-player game *OpenAI Gym Pong* [4] partly to test the cooperative emergence: we train a cooperative agent to play with the competitive partner. As a single-agent task, this pre-experiment is easier for training. The reward scheme is set as Table 4.2. Note that because in single player *Pong* the ball is automatically fired, the agent only has three actions: *UP, DOWN*, and *STAY*.

	Agent scores	Competitive partner scores	No one scores
Agent reward	-1	-1	0

Table 4.2: Reward scheme of *single player Cooperative Pong*.

Since a single player cannot hold the ball constantly, both single-player and multi-player versions are strictly cooperative games. As discussed by Minh et al. [24], one single screen frame in the original Atari *Pong* contains partial information since the action and speed are not known. By frame stack from the preprocessing of original DQN, the Atari *Pong* task becomes fully observable. Hausknecht and Stone also mention this discussion in their work [14] of customized partially observable Atari game. To test partially observable *Pong*, they cast the game frames as flickering screens.

We make use of the original two-player *Pong* with *Pettingzoo* Python API [41] to be the environment for the first subtask (sufficient information). we also design a partially observable version of this game for subtasks 2 to 4: we split the frame into two parts (see Figure 4.2). Each part of the frame contains one player who is not aware of the partner's

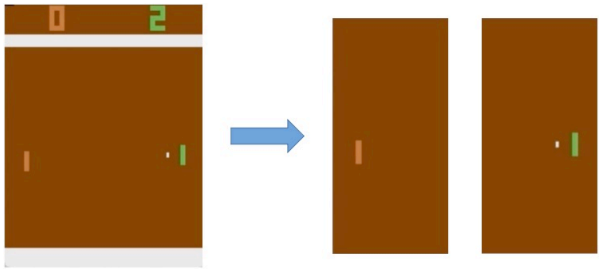
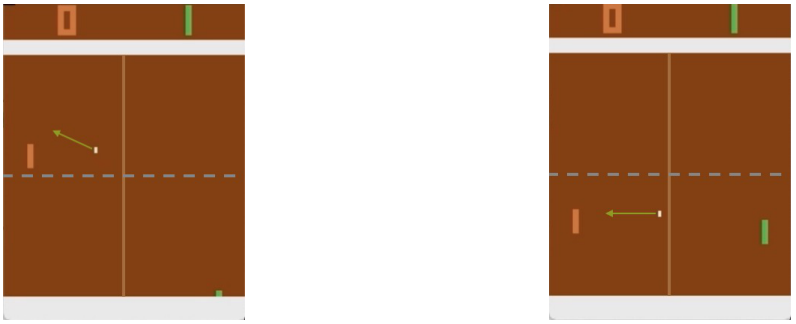


Figure 4.2: Split frame *Pong*

position and speed. If the frame stack is used, the observation of each player retains partial. This version allows the agent to exchange its position or action information that is essential for training but not accessible with the partner. We do the experiments on sufficient information version and set the partial version in the future plan, while here we provide the possible design regarding the adaptive information exchange.



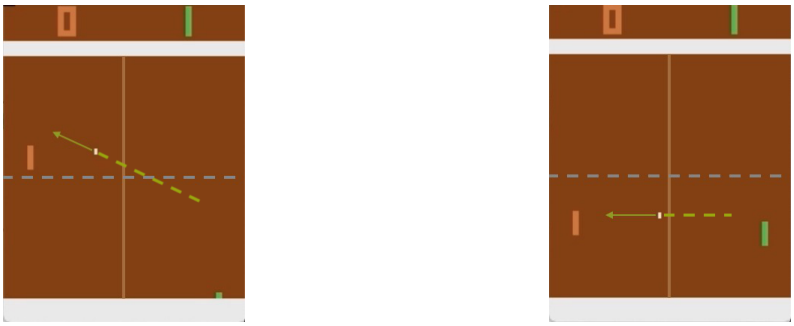
(a) (b)

Figure 4.3: Two situations of cooperative *Pong* with the split frame. The vertical brown line indicates the split wall, which is the borderline of the player's visible area. The olivine arrow indicates the direction of the ball. The dashed horizontal grey line is the hard-coded position boundary. (a) The left and right players are distributed on both sides of the position boundary. (b) Two players are on the same side.

As we mentioned before, in the split frame *Pong*, each player does not know the position of the partner and sometimes the velocity of the ball either, which hinders the decision making. For the hard-coded fuzzy signal exchange, we can handcraft a boundary for agents to describe their positions, shown as the dashed horizontal grey line in Figure 4.3. There are two kinds of situations these two players may encounter: both players are on the different sides or the same side of the position boundary (e.g., Figure 4.3b). When players are on both sides, our hard-coded border can work as the indicator of positions. For example, in Figure 4.3a, the left player is facing the ball and going to conduct an action. It may perform *UP* successively to capture the ball. But

to help the partner catch the ball without struggle, the left player may send a message to the partner about the upcoming direction of the ball. When the partner (the right player) receives this message, it can utilize this information and take necessary actions (e.g., going up) before the ball flying into its visible zone when it might be too late for the right player to take the right actions.

In contrast, if the situation is the second one, e.g., two players are on the same side, the agent may get confused by the hard-coded information. For instance, the left player tells the partner that the ball will be below the boundary on the next time step. This message comes from the direction of the ball and the handcrafted information. But it is inefficient because it does not convey the relative position of the left player. If this signal turns to be learnable, the agent may send the indications with the ball's direction based on its prediction of the partner's position (dashed olivine lines in Figure 4.4).



(a) (b)

Figure 4.4: Two situations in cooperative *Pong* with the split frame and predicted partner's positions from the perspective of the left player.

4.1.2 Thegame

Thegame - Quick and Easy [25] (hereinafter refers to *Thegame*) is a cooperative card game designed by Nürnberger-Spielkarten-Verlag (NSV). It is usually played by 2 to 5 players, containing 50 cards total, with five colors and ten cards in one color. It has two ordering stacks indicated by two sequence cards (ascending and descending stacks, see Figure 4.5 left). Players need to lay their cards on these two stacks according to the orders. In the two-player version we use, each player has two cards at most in hand. At each turn, one player can lay one or two cards or wait on this turn. If the player lays out one or two cards on the ordering stack(s), it needs to draw the same number of cards from the card stack, unless there are no cards to draw anymore. The goal of the team is to lay as many cards as possible on two ordering stacks. During gameplay, players can

Figure 4.5: Left: sequence cards (indicating ascending and descending stacks) and number cards (5 colors × 10 cards per color) in *Thegame*; Right: reverse trick: order limits of the stack can be ignored when the hand card and the topmost stack card have the same color. E.g., card green 8 has the same color as the topmost card (green 2) on descending stack, one can ignore the order limit and lay green 8 on top of green 2 on the descending stack.

only get access to the topmost ordering stack cards. Logically speaking, the maximum number of cards a team can lay out on these two stacks is 20 in total, but there is a reverse trick (see Figure 4.5 right): if the hand card one wants to lay on a stack (e.g., card green 8 in the right subfigure of Figure. 4.5) has the same color with the topmost card on that stack, then one can ignore the order limit.

There is a rule of import that increases the difficulty of this game: players can talk about the color(s) of their hand card(s), whereas they are not allowed to speak the exact number(s). While the hint messages are permitted, players can give hints like "I have a big blue" to indicate owning a blue card with a large number. As it forces the players to communicate with each other, this rule makes fuzzy information exchange possible. Due to the high dynamics of the card initialization and player actions, the sensation of each player on the number size will change based on different situations. For example, the player will think of the hand card four as a small card when one stack card is seven, and consider it as a big card when one stack card is three, which allows the training of adaptive fuzzy signals (subtask 4). These features all fit our environment-design criteria.

We set the player size as 2 in our experiments. For each player, the action space $\mathcal{A}$ contains $|\mathcal{A}| = 11$ different actions. Specifically, this space size comes from playing two hand-cards to different order stacks (playing single card on the same stack (4) + playing two cards on the same stack (4) + playing two cards on different stacks (2), and stay (1)). We make use of a two-dimension tuple to represent a card: the first dimension for color c

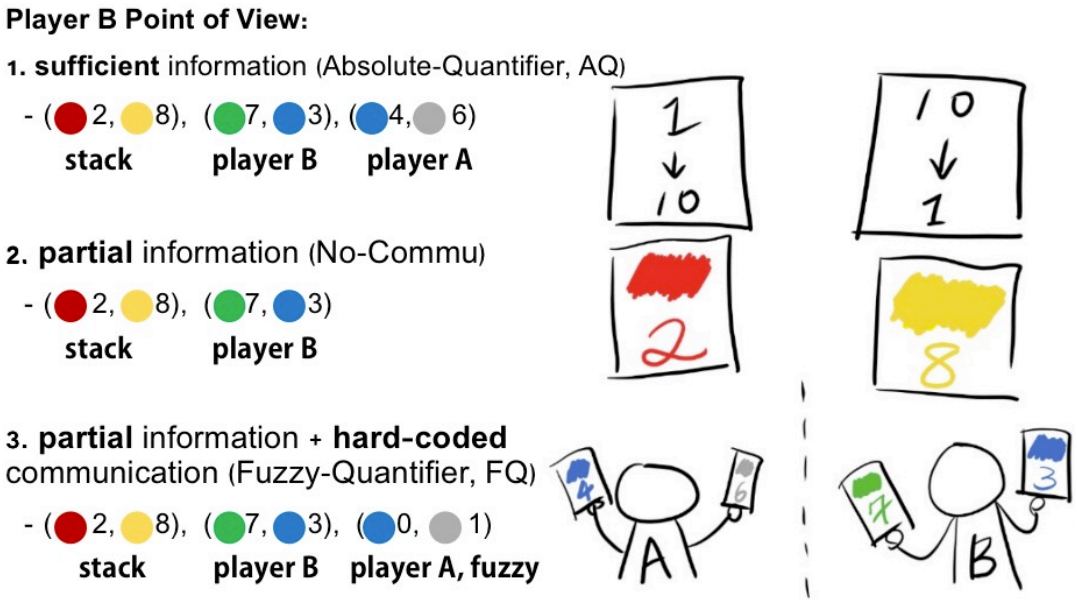


Figure 4.6: An illustrative state of *Thegame* and the corresponding vectorized observations of three environments in the point of view of player B.

(digitally depicted by 1 to 5 for five colors *red, yellow, blue, green*, and *grey*, respectively) and the second dimension for the digit number x, where $1 \leq x \leq 10,\ x \in \mathbb{N}^+$. Refer to Figure 4.6 as an gameplay example. According to our subtasks 1 to 3 and environment design criteria, there are three types of agent observation:

1. sufficient information version, each agent is aware of the topmost cards of two stacks, together with its two hand-cards and two partner's cards. This version is also called the absolute-quantifier (AQ) version since the exact numbers, a.k.a. absolute quantifiers from partners are known. Because players are not aware of the card history, this version is not fully observable;

2. partial information version, each agent only knows the two topmost stack cards and their two hand cards. The player has no idea of what number(s) the partner has. This version is also known as the no-communication version;

3. hard-coded fuzzy communication signal version, where each agent can get access to the two topmost stack cards, its hand cards, and the hint sent from the partner player. One hint is a two-bit signal and constituted by handcrafted rules: $\mathbb{1}\{5 < x \leq 10\}$, which means if the partner's card number $x \in \{X | 1 \leq X \leq 5,\ X \in \mathbb{N}^+\}$, x is a low number and the hint is 0. Otherwise, x is a large number and the hint

turns to be 1. This version is named the hard-coded fuzzy-quantifier (FQ) version as well.

As a multi-agent game, *Thegame* has its specialties compared with *cooperative Pong*: it is a turn-based game, i.e., only one agent takes action in a single time step. It means the agent cannot obtain its *next-state* immediately (for a two-player game, the state agent A gets after taking a certain action is the real *next-state* for after agent B taking its last action). In other words, the reward is not instantly obtained after the agent taking action. We can make use of *state-action-next_state* pair (see Eq. 2.3) we discussed in Sec. 2.1 to describe it more conveniently (also see discussion from RLlib Github issue [30]). This 'delayed' *next-state* often happens on multi-player turn-based games. Usually the multi-agent RL environment treats it with cumulative reward (agent A obtains reward after all partners taking actions and the environment returns the real next-state for A), e.g., *Pettingzoo* introducing Agent Environment Cycle (ALE) [41], OpenAI Five [26]. Metz et al. treat the reward in a similar way, and they also propose a state composition representation to mathematically describe turn-based states [22].

Player who finishes the game	A	B	No one
Player A reward	1	1	0
Player B reward	1	1	0

Table 4.3: Reward scheme of *Thegame*.

Regarding the reward scheme, we set it similar to the reward schemes of *Cooperative Pong* (see Table 4.3). If the team wins the game, i.e., two players lay all 50 cards, no matter who finished the game (who is the last one to lay card(s)), each player obtains 1 point as a reward. Otherwise, the reward is 0. It endows *Thegame* to be a sparse reward task. The game emulator has a $max\_step = 200$ parameter. If two players do not lay out all cards and the time limit is reached, the game round will be forced to end and remains *undone*.

4.2 Training Setups

We use DQN [24] for both games. Especially, according to the work of Tampuu et al. [40], we deploy independent DQN for *Cooperative Pong*. For *Thegame*, we utilize DQN self-play for subtask 1 to 3, and we also try PPO [31] for subtask 1.

4.2.1 Cooperative Pong

As an attempt, we first start from single-agent *Pong* to check the DQN model for cooperation emergence. We set the number of training steps as 10M and the hyperparameters for DQN similar as Minh et al. [24], except the ones in Table 4.4. The training procedure contains four epochs, 250000 time steps within one epoch. We are going to discuss the hyperparameter selection in the next chapter.

Hyperparameter	Value
minibatch size	64
final exploration frame	100000
learning rate	0.0001
update frequency	1
target network update frequency	1000

Table 4.4: Hyperparameter values of single player *Cooperative Pong* that differ from *Nature* DQN [24].

For the two-player *Cooperative Pong*, the training procedure of subtask 1 is as same as Independent DQN [40], containing 50 epoch and 250000M time steps in one epoch. Each agent is treated as an individual. These hyperparameters are set as *Nature* DQN [24] as well. Here we list the values we use that are different from *Nature* DQN in the experiments:

Hyperparameter	Value
minibatch size	64
final exploration frame	100000
learning rate	0.0001
target network update frequency	1000

Table 4.5: Hyperparameter values of two-player *Cooperative Pong* that differ from *Nature* DQN [24].

4.2.2 Thegame

Since *Thegame* emulator is a brand new environment, we first design a heuristic agent as a benchmark test. The heuristic method does not share information between agents, and it does not contain the learning process of cooperation. It is directly based on the distance: according to the game's rule, each player has two hand cards at most in one time step (note that only there are not enough cards from the card pale, can

one player holds less than two cards), and we want to lay as many cards as possible, which means after one turn playing, the topmost card on ascending stack is favorable to be small, and on descending stack is better to be large. Recall that the agent has 11 different candidate actions. For heuristic agents, we discard the $STAY$ action and enforce agents select one 'real' action during playing. For one turn, we list all the results of ten actions and calculate the distance for each stack. For instance, if the topmost cards are "Ascending: 3, Descending: 5" after taking a certain action, the distance of this action is $(3-1)+(10-5)=7$. We compare ten distances and select the action with the shortest distance (if there are multiple such actions, the agent picks one randomly).

We use DQN self-play for subtasks 1 to 3. Each agent takes action based on its own observation, while agents share the learning models (both behavioral and target models) together. We run the experiments for 1000 epochs, with 100 complete games within one epoch, using three different random seeds. Because the input is digit number as state, rather than RGB color, the approximation function we use is the multi-layer perceptron with a single hidden layer. Since there is no previous work for hyperparameter settings, we test the influence of the number of hidden units, learning rate, minibatch size. We are going to analyze the results in section 5. Besides, we select the best hyperparameter setting and list the differences compared with *Nature* DQN as follows:

Hyperparameter	Value
minibatch size	64
discount factor	0.95
final exploration	0.02
target network update frequency	40000
learning rate	0.0001
update frequency	16

Table 4.6: Important DQN hyperparameters for *Thegame*.

As comparison of value-based models, we also train our absolute-quantifier agent using default PPO model from *RLlib* pytorch API [17]. The environment are built with *Pettingzoo* and *RLlib* customized environment wrappers. Except the learning rate $\alpha = 0.0001$, discounting factor $\gamma = 0.99$ and horizon $hz = 200$ (horizon is equivalent to max_step we set for DQN agents), all hyperparameter we used are the default ones from the API.

5 Results

In this chapter, we describe the results of these two games. First, we show the cooperation emergence of single-player and two-player *Cooperative Pong*. Second, we compare the results of three environment settings of *Thegame* and analyze the importance of information exchange between players for cooperation and how does it provide the potential of adaptive signals learning.

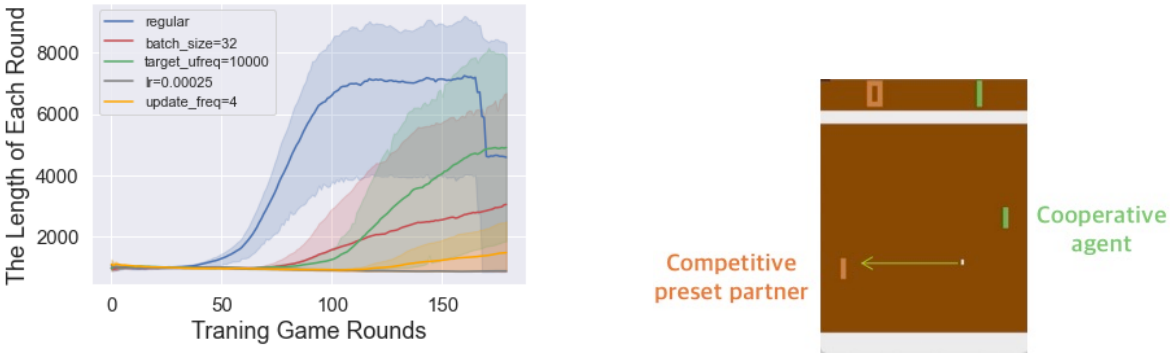
5.1 Cooperative Pong

With the help of the collaborative reward scheme, experiments on both single and two-player *Pong* show the emergence of cooperative behaviors. Guided by the reward scheme, the agent has learned how to behave to prevent a minus reward, reflecting on the learned skillful actions for catching the ball and helping the partner.

5.1.1 Cooperative Player with Competitive Partner

As shown in Table 4.4, we use $replay\ buffer = 100000$ rather than $replay\ buffer = 1M$ for training under the restriction of computing resources. Meanwhile, other hyperparameters also need to be tuned. Figure 5.1a shows how we determine the hyperparameters for single cooperative Pong experiments (the results are averaged from 5 different random seeds). We illustrate the performances of different settings for 180 game rounds training. The Y-coordinate indicates the length of each game round. A higher round length means the cooperative agent learned better skills to keep the ball with the aggressive partner. As shown, the $batchsize$, $targetupdatefrequency$, $learningrate$, and $updatefrequency$ from the original DQN settings [24] do not work on this task, therefore we manage them as the values for regular player, listed in Table 4.4.

The results show regular player has the best average performance and the earlier convergence. Before around 50 game rounds of training, the agent is unskillful to keep the ball and unable to extend the length of a game round. As the training process



(a) (b)

Figure 5.1: (a): The game length (total time steps) of one game during training. (b): An instance screenshot in single-player *Cooperative Pong*. The olivine arrow indicates the direction of the ball passing from the cooperative agent to the competitive preset partner.

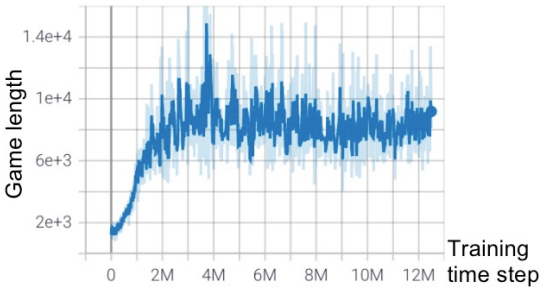
gets longer, the agent has learned to catch the ball and pass it tenderly to the partner. Figure 5.1b is a screenshot of regular player during test [1]). The exploration probability is $\epsilon = 0.01$, which means it has the probability of 0.01 to choose a random action, otherwise using the action selected by the behavior policy. Playing with a competitive player who tries the best to win, our (regular) cooperative agent finds a way to help the partner and behave gently. When the aggressive partner passes the ball at an acute angle and high speed, our cooperative agent catches the ball easily and lowers the speed of the ball, and changes it into the horizontal direction to help the partner catch it (see Figure 5.1b. After 150 rounds of training, sometimes the agent forgets what it has learned and performs worse than before (see the curve of regular player that suddenly drops after around 170 rounds of training).

5.1.2 Two-player Cooperative Pong

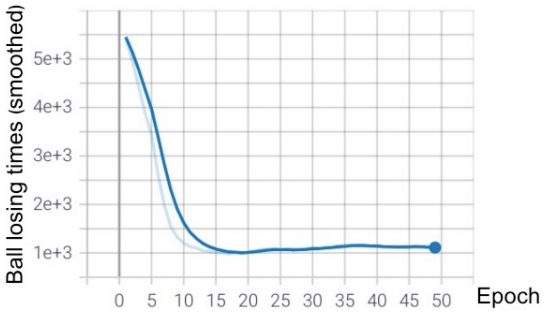
We train the independent DQN models for the two-player environment utilizing similar hyperparameters compared to the single-agent player (we tune the update frequency value equals to four, which is as same as the original DQN setup [24]. Contrasted with the cooperative agent playing with the aggressive partner in the single-player version, two cooperative players learn to work together from scratch in this task.

Figure 5.2 shows the experimental plots and a screenshot after 50 epochs training for two-player *Cooperative Pong* with full-screen inputs. During training, two cooperative

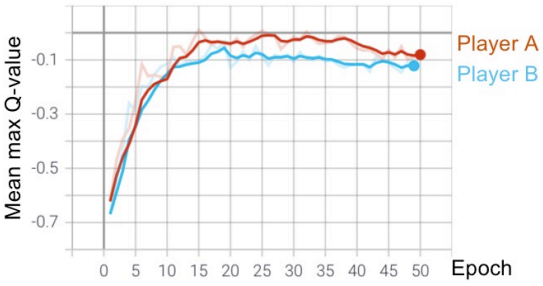
[1]The rendered video: `https://drive.google.com/file/d/1M1x1NjktJBzcpy1K3joAVoeQGE1SdhsP/view?usp=sharing`



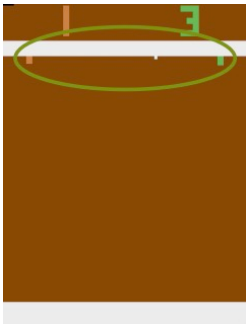
(a) The length of one game during training.



(b) Times of ball losing in one epoch (250000 time steps)during training



(c) Q-value of player A and B during training.



(d) Screenshot of two-player *Cooperative Pong*.

Figure 5.2: Experimental results of two-player *Cooperative Pong*.

players have learned skills to keep passing the ball for a long time. Graphically, this is manifested by the increase in the duration of one game round (see Figure 5.2a), the decrease of ball losing times (Figure 5.2b), as well as the change of mean max Q-value (Figure 5.2c). We monitor the mean max Q-value in the same way as Tampuu et al. [40] to present the learning process. Before the training begins, we randomly save 500 states. After each epoch, we calculate the maximum Q-value of these 500 states using the behavioral network to store the average value for both players. Because we set the reward function as Table 4.1 (the maximum reward our agents can get is 0), the behavioral mean max Q-value shows the reliability of keeping the ball calculated by agents and the possibility to select the good action on a certain state. As the training progresses, the Q-values of the best moves estimated by two players approach zero. It indicates that both players are aware of how to cooperate with the partner and avoid losing the ball.

The screenshot in Figure 5.2d is taken from the rending video[2] during testing phase using the models after training 50 epoch. In the test stage, we set $\epsilon = 0.01$ for exploration. Therefore, the agent still has *probability* $= 0.01$ to select a random action.

[2] https://drive.google.com/file/d/1ohCbe-KqpCgRfasPGQa-LaBcneyQ_OOH/view?usp=sharing

To pass the ball continuously, and avoid the influence of random action efficiently, two players find the best position to collaborate: getting stuck at the top of the screen frame.

These results indicate the DQN works well on our first subtask with fully observable *Pong*. The agent also shows cooperative behaviors on both single and multi-player versions. Due to limited game API and computing resources, subtasks 2 to 4 on this environment are planned as future works.

5.2 Thegame

Before training the deep neural network models, we test our heuristic players with 100000 game rounds, i.e., 100000 different random initial card stacks. Figure 5.3 shows the heuristic player performance. The game win rate in the plot is the percentage of game wins. A higher rate implies the agent is more proficient at playing and cooperating. According to the result, the heuristic players finish 37.0% of the game rounds.

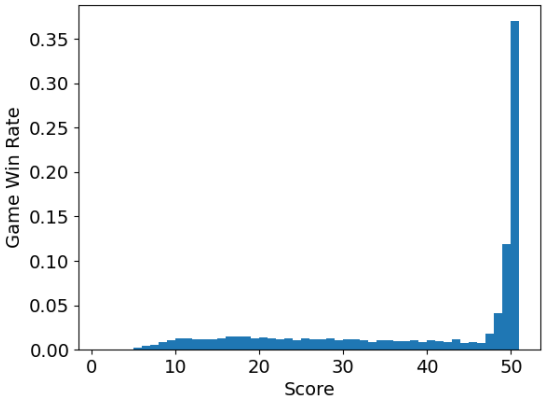
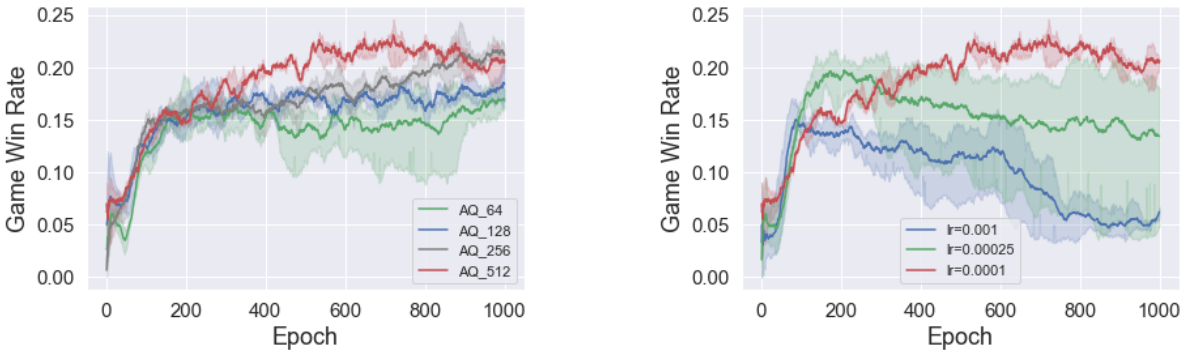


Figure 5.3: Heuristic team's performance on *Thegame*. The Y-coordinate indicates the percentage of wins. A higher rate implies the agent is more proficient at playing and cooperating.

We try to find a suitable hyperparameter setting for sufficient information (absolute-quantifier) version. The hidden unit size hd and learning rate α are two important ones. Therefore we start from these two hyperparameters.

The hyperparameter selection plot for absolute quantifier version (see Figure 5.4) shows $hd = 512$ and learning rate $\alpha = 0.0001$ works best among all candidate ones. Especially, a higher learning rate greatly spoils the model performance after 100 to 200 epochs of training. Figure 5.4b also implies that DQN is not stable. The higher learning



(a) Variable: hidden unit size

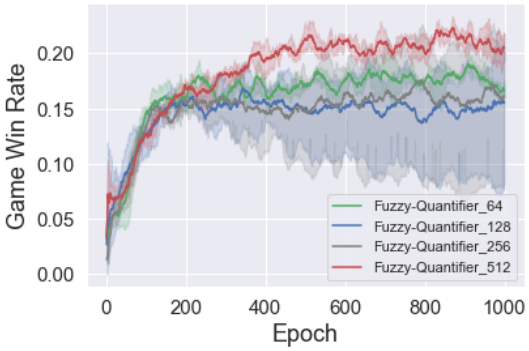
(b) Variable: learning rate

Figure 5.4: Hyperparameter selection for *Thegame* sufficient information ver. The results are averaged using three different random seeds.

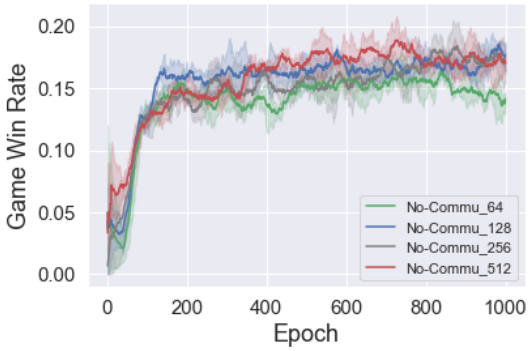
rate α makes the model fail to maintain the learned behaviors (a.k.a., Catastrophic Forgetting) and get damaged.

Figure 5.5 presents the parameter comparison for the hard-coded fuzzy quantifier agents and no-communication agents. Similar as absolute quantifier version, hidden unit size $hd = 512$ is the best value for fuzzy quantifier as well (see Figure 5.5a). For the no-commu agent, $hd = 512$ shows a slight advantage (see 5.5c). Regarding the learning rate, $lr = 0.0001$ works the best in both environments. Therefore we use $hd = 512$ and $lr = 0.0001$ for fuzzy quantifier and no-communication versions as well. Figure 5.5a shows the model performance would fluctuate using hidden unit size $hd = 64$ and $hd = 256$ in fuzzy quantifier agents. Meanwhile, different hidden unit sizes do not present many differences for no-communication agents. It may be due to the limitation of the inputs. The experiments on the learning rate for these two versions (Figure 5.5b and 5.5d) also indicate the harms of an overhigh learning rate: agents get stuck in the local optima with it.

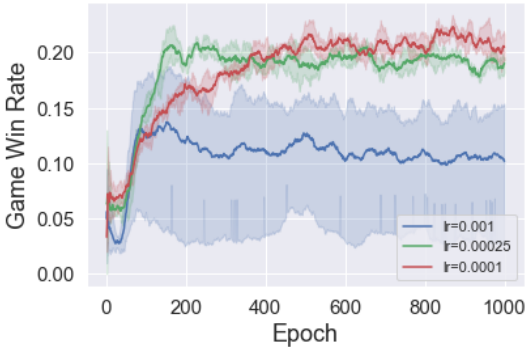
The experiments of three environment settings (see Figure 5.6) show all of the trained agents work better than the random players, and both absolute quantifier and hard-coded fuzzy quantifier agents outperform no-communication agents. It indicates additional information exchange helps agents to gain a better understanding of the current situation. Similar to the *Number Sorting* game in the chapter 3, the agent may also learn the scales of fuzzy quantifiers to describe their cards when the stack card number changes. Based on this indirect information, one player can take advantages of action *STAY* to give the partner opportunity to act more beneficially for the team (e.g., perform reverse trick), which is the purpose of our subtask 4 in future work.



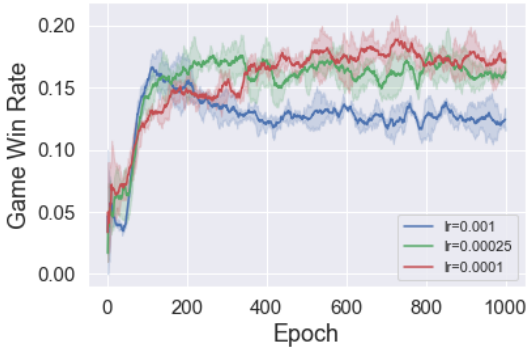
(a) Hard-coded fuzzy quantifier, variable: hidden unit size



(b) No-communication, variable: hidden unit size



(c) Hard-coded fuzzy quantifier, variable: learning rate



(d) No-communication, variable: learning rate

Figure 5.5: Parameter (hidden unit size and learning rate) comparison for fuzzy quantifier and no-communication *Thegame* agents. The results are averaged using three different random seeds.

Figure 5.6: Game win rates of three *Thegame* environment settings.

In Figure 5.6, we also notice the fuzzy quantifier observation gives a comparable performance contrasted with the absolute quantifier version. However, this does not

directly demonstrate absolute and hard-coded fuzzy quantifiers have equivalent effects generally. Because there are a few possible reasons: 1) this might be an environment specialty, which means in *Thegame*, fuzzy quantifier information is enough for training; 2) model limitations: as we discussed before, DQN is a simple model which might limit the performance of the agent. Besides, we also notice best DQN results among the three types of agents are still not as good as the heuristic agent.

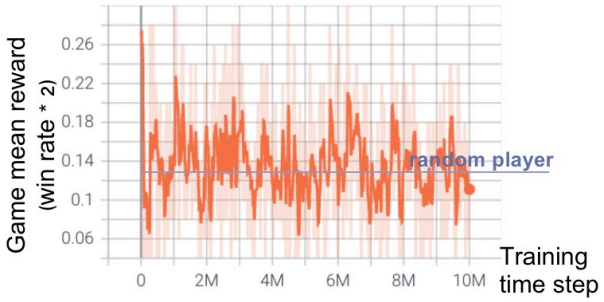


Figure 5.7: The game mean reward (double of game win rate) on sufficient input *Thegame* using PPO. The customized cooperative environment made by Pettingzoo with RLlib counts the total rewards of all team members. Therefore, the game mean reward is two times of game win rate we used before.

In comparison to DQN models, the PPO model with default hyperparameters does not work in our environments (see Figure 5.7): the model performs similarly as the random player. It has multiple possible causes as well. First, default PPO from RLlib needs hyperparameter tuning. Second, our environment is designed with sparse reward, whereas PPO is an on-policy method that trains agents with full trajectories. If the team does not hit the final point to get a positive reward, the model struggles with all zero rewards and can hardly obtain any useful information for updating.

6 Discussion

In this section, we are going to discuss the distinctive characteristics of our tasks, the unfinished part of our work, and the potential future plan.

6.1 The Goal and Subtasks

Our current experimental results show the positive effect of reinforcement learning and information exchange for cooperative tasks. The experiments for *Thegame* also present the potential of fuzzy information and the prospect of fuzzy quantifier emergence. As mentioned before, we divide our goal - the communication emergence - into four subtasks. We verify the first subtask for *Cooperative Pong* and the first three for *Thegame*. Recall in chapter 3 we discuss that, if hard-coded fuzzy information helps the agents perform better than non-communication agents, it indicates the possibility of learnable fuzzy information, where agents could become more flexible in expressing the situation to partners and interpret partners' messages. As a part of the concept in Steels' *Language game* for the human linguistic evolution model and the language acquirement of embodied robots, this ability (we call it the situation expression module) can be learned during interactions between agents. In the perspective of reinforcement learning, this expression module might be deployed by a neural network with reinforcement learning algorithms as well, for example, Foerster et al.'s DIAL agent [11], and Cao et al.'s Cheap Talk agent [6]. Whereas compared with other tasks that focus on adaptive communication emergence, our subtask 4 is slightly different: the agents need to tackle two tasks during training in the long run: 1) learn how to play the basic game (e.g., how to use the racket in *Pong*, or learn the rule in *Thegame*); 2) learn how to communicate with the partner(s). These two missions might be learned at the same time. Otherwise, we can utilize transfer learning methods to train these two abilities in different learning stages then equip them for every agent.

Second, the communication in our task is ideally designed task-free. Contrasted with other works [6, 11, 37] that constrain the signal range into task-specific limited spaces,

the learnable fuzzy signal in our experiment design is expected to be formed by the agents themselves and can be reused for similar tasks, based on the idea of formation experiments in *Language Game* [29]. It also endows the agents to be autonomous and can self-organize the communication approaches according to different partners and even different tasks. Furthermore, the decentralized procedure may help agents learn to cooperate and exchange information in a human-understandable way [10], which may be used in human-computer interaction tasks.

6.2 Possible Improvements on Experiments

Except the reward scheme modification can be a direction for improvement, for *Thegame*, there are some potential ways as well, such as global environment information and input encoding: for *Thegame*, although the agent can become aware of the partner's cards, it is still a partially observable game, due to the lack of information regarding the card history. This history can be a part of input together with the observation. Besides, we can make use of the time limitation [28] as well to inform the agent how long it has taken in a game round. Similar to Silver et al.'s AlphaZero [33] and Bard et al.'s Hanabi agent [1], we can also encode the agent's observation together with helpful information to help it make decisions.

7 Conclusion

In this thesis, we propose a task flow for multiple cooperative agents training in the partially observable environment with information exchange. Inspired by Steels' Language Game, we assume agents can learn how to use signals to convey information by interactions with the partner(s) during game playing. To achieve this goal step by step, we divide it into a series of subtasks: first, compare the performance of relatively full and limited information input to check the potential of information exchange, then utilize hard-coded and even adaptive signals to help the agent learning. We exploit the subtask 1 on *Cooperative Pong* and *Thegame* and subtask 2 and 3 for the latter game with DQN. In the first subtask, the experiments of these two games with sufficient inputs both show cooperation emergence within a team. The results on the hard-coded fuzzy signal (subtask 3) of *Thegame* show the agents take advantage of the valuable information of fuzzy quantifiers and outperform the partially limited information agents (subtask 2). Though we do not test *Cooperative Pong* on subtask 2, 3, 4, and *Thegame* for subtask 4, we provide the possible environmental and experimental design for them and analyze the structure of the adaptive communication learning model for subtask 4. These subtasks, especially the subtask for adaptive communication signal learning, together with RL models comparison, agent input encoding, and other possible improvement approaches are set as future works.

Bibliography

References

[1] N. Bard et al. "The Hanabi Challenge: A New Frontier for AI Research". In: *Artif. Intell.* 280 (2020), p. 103216.

[2] A. Bauza. *Hanabi & Ikebana | Board Game | BoardGameGeek*. https://www.boardgamegeek.com/boardgame/70918/hanabi-ikebana. 2010.

[3] B. D. Boer. "Computer modelling as a tool for understanding language evolution". In: 2006.

[4] G. Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[5] L. Buşoniu, R. Babuška, and B. D. Schutter. "A Comprehensive Survey of Multiagent Reinforcement Learning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38 (2008), pp. 156–172.

[6] K. Cao et al. "Emergent Communication through Negotiation". In: *ArXiv* abs/1804.03980 (2018).

[7] Y. Chen et al. "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)* (2017), pp. 285–292.

[8] H. H. Clark and S. Brennan. "Grounding in communication". In: *Perspectives on socially shared cognition*. 1991.

[9] C. Claus and Craig Boutilier. "The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems". In: *AAAI/IAAI*. 1998.

[10] P. V. Eecke and K. Beuls. "Re-conceptualising the Language Game Paradigm in the Framework of Multi-Agent Reinforcement Learning". In: *ArXiv* abs/2004.04722 (2020).

[11] J. N. Foerster et al. "Learning to Communicate with Deep Multi-Agent Reinforcement Learning". In: *NIPS*. 2016.

[12] S. Gronauer and K. Diepold. “Multi-agent deep reinforcement learning: a survey”. In: *Artificial Intelligence Review* (2021), pp. 1–49.

[13] H. V. Hasselt, A. Guez, and D. Silver. “Deep Reinforcement Learning with Double Q-Learning”. In: *AAAI*. 2016.

[14] M. Hausknecht and P. Stone. “Deep Recurrent Q-Learning for Partially Observable MDPs”. In: *AAAI Fall Symposia*. 2015.

[15] M. Hessel et al. “Rainbow: Combining Improvements in Deep Reinforcement Learning”. In: *AAAI*. 2018.

[16] L. Kaelbling, M. Littman, and A. Cassandra. “Planning and Acting in Partially Observable Stochastic Domains”. In: *Artif. Intell.* 101 (1998), pp. 99–134.

[17] E. Liang et al. “RLlib: Abstractions for Distributed Reinforcement Learning”. In: *ICML*. 2018.

[18] T. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *CoRR* abs/1509.02971 (2016).

[19] L. Lin. “Reinforcement Learning for Robots Using Neural Networks”. UMI Order No. GAX93-22750. PhD thesis. USA, 1992.

[20] M. Littman. “Markov Games as a Framework for Multi-Agent Reinforcement Learning”. In: *ICML*. 1994.

[21] Ryan Lowe et al. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”. In: *NIPS*. 2017.

[22] L. Metz et al. “Discrete Sequential Prediction of Continuous Actions for Deep RL”. In: *ArXiv* abs/1705.05035 (2017).

[23] V. Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *ICML*. 2016.

[24] V. Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518 (2015), pp. 529–533.

[25] NSV. *THE GAME – QUICK & EASY*. https://nsv-games.com/the-game-quick-easy/. 2021.

[26] OpenAI. *OpenAI Five*. https://blog.openai.com/openai-five/. 2018.

[27] H. Osawa. “Solving Hanabi: Estimating Hands by Opponent’s Actions in Cooperative Game with Incomplete Information”. In: *AAAI Workshop: Computer Poker and Imperfect Information*. 2015.

[28] F. Pardo et al. “Time Limits in Reinforcement Learning”. In: *ArXiv* abs/1712.00378 (2018).

[29] S. Pauw and J. Hilferty. “The Emergence of Quantifiers”. In: *Experiments in Cultural Language Evolution*. Ed. by Luc Steels. Vol. 3. Advances in interaction studies. Philadelphia: John Benjamins Publishing Company, 2012, pp. 277–303.

[30] rusu24edward. *[rllib] turn-based multi-agent environments usage #9029*. https://github.com/ray-project/ray/issues/9029. 2020.

[31] J. Schulman et al. “Proximal Policy Optimization Algorithms”. In: *ArXiv* abs/1707.06347 (2017).

[32] J. Schulman et al. “Trust Region Policy Optimization”. In: *ArXiv* abs/1502.05477 (2015).

[33] D. Silver et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In: *ArXiv* abs/1712.01815 (2017).

[34] L. Steels. “Language games for autonomous robots”. In: *IEEE Intelligent Systems* 16 (2001), pp. 16–22.

[35] L. Steels. “The synthetic modeling of language origins”. In: *Evolution of Communication* 1 (1997), pp. 1–34.

[36] L. Steels and M. Spranger. “How Experience of the Body Shapes Language about Space”. In: *IJCAI*. 2009.

[37] S. Sukhbaatar, A. D. Szlam, and R. Fergus. “Learning Multiagent Communication with Backpropagation”. In: *NIPS*. 2016.

[38] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0-262-19398-1. URL: http://www.cs.ualberta.ca/%7Esutton/book/ebook/the-book.html.

[39] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[40] A. Tampuu et al. “Multiagent cooperation and competition with deep reinforcement learning”. In: *PLoS ONE* 12 (2017).

[41] J. K. Terry et al. “PettingZoo: Gym for Multi-Agent Reinforcement Learning”. In: *arXiv preprint arXiv:2009.14471* (2020).

[42] Z. Wang et al. “Dueling Network Architectures for Deep Reinforcement Learning”. In: *ArXiv* abs/1511.06581 (2016).

[43] C. J. C. H. Watkins. “Learning from Delayed Rewards”. PhD thesis. King’s College, Oxford, 1989.

[44] R. P. Wiegand and K. A. Jong. “An analysis of cooperative coevolutionary algorithms”. In: 2004.

[45] L. Wittgenstein. *Philosophical Investigations*. Oxford: Basil Blackwell, 1953. ISBN: 0631119000.

[46] L. Zadeh. “A Computational Approach to Fuzzy Quantifiers in Natural Languages”. In: *Computational Linguistics* (1983), pp. 149–184.