University of Liège
School of Engineering
Montefiore Institute

CONTRIBUTIONS TO MULTI-AGENT REINFORCEMENT LEARNING

a PhD dissertation
by Pascal Leroy

LIÈGE université
Sciences Appliquées

Advisors:
Professor Damien Ernst
Doctor Jonathan Pisane
April 2024

This dissertation has been submitted in partial fulfilment of the requirements for the Degree of Doctor of Philosophy in Engineering Science.

ABSTRACT

In the realm of machine learning, multi-agent reinforcement learning (MARL) is the setting where several agents learn to act by receiving rewards after deciding on actions based on their perception of their environment. It takes its foundation in game theory and reinforcement learning (RL), both fields developed for decades. The success of deep neural networks is leading to unprecedented progress in the variety of problems that can be solved by such methods. Indeed, many real-world applications, such as autonomous vehicles, swarms of drones, warehouse robots, cyber securities, traffic management, or smart grids, can be framed as MARL ones. In this thesis, we present contributions in this domain, particularly in training a team of agents to cooperate alone or against an opposing team. This manuscript starts with the fundamentals, defining the general MARL framework and how it is divided into the settings of cooperation, competition, and general-sum. It also provides the necessary background for the unfamiliar reader with RL.

The second part of the thesis is dedicated to the cooperative setting, where agents share the same goal. Such a setting is commonly framed as a decentralised partially observable Markov decision process (Dec-POMDP). This part begins with a background chapter defining the Dec-POMDP and how it is solved in the literature. The following chapters present two contributions to the cooperative MARL field. One concerns methods extending the Deep-Quality-Value family of algorithms to the cooperative setting, demonstrating competitive performance with the state of the art. The other presents IMP-MARL, an open-source suite of MARL environments for large-scale infrastructure management planning (IMP). In IMP, inspections, repairs and/or retrofits should be decided to control the risk of potential system failures while minimising costs, such as bridge or wind turbine failures.

The third part of the thesis addresses the problem of training a team against an opposing one. This setting extends the competition setting, framed as agents having opposing goals, to teams of agents having opposing goals. It is composed of two chapters. The former presents historical stories and solutions from game theory to the competition and general-sum settings. The latter presents the third contribution, which is a study on how to pair cooperative and competitive methods to train a team in a two-team Markov game with the objective to be resilient to many strategies.

Finally, the manuscript concludes with a retrospective of the scientific findings provided by the contributions at the foundation of this thesis and a discussion on the societal impact that MARL has the potential to provide in the upcoming years.

CONTENTS

1 INTRODUCTION 1
1.1 Outline . . . 6
1.2 Publications . . . 7
1.3 Context . . . 7

I BACKGROUND 9
2 FOUNDATIONS OF MULTI-AGENT REINFORCEMENT LEARNING 11
2.1 Introduction . . . 11
2.2 Stochastic game . . . 12
2.3 Multi-agent settings . . . 14
2.3.1 Cooperation . . . 15
2.3.2 Competition . . . 15
2.3.3 General-sum . . . 16
2.4 Single-agent reinforcement learning . . . 16
2.4.1 Markov decision process . . . 17
2.4.2 Model-based or model-free . . . 18
2.4.3 Dynamic programming . . . 18
2.4.4 Value-based methods . . . 20
2.4.5 Policy-based methods . . . 22
2.5 Partial observability . . . 23

II LEARN TO COOPERATE 25
3 COOPERATION 27
3.1 Introduction . . . 27
3.2 Decentralised partially observable Markov decision process . . . 28
3.3 Environments . . . 30
3.3.1 StarCraft multi-agent challenge . . . 32
3.4 Value-based methods . . . 34
3.4.1 QMIX . . . 35
3.4.2 MAVEN . . . 36
3.4.3 QPLEX . . . 38
3.4.4 Other value-based methods . . . 39
3.5 Policy-based methods . . . 39
3.5.1 COMA . . . 40
3.5.2 FACMAC . . . 41
3.5.3 Other policy-based methods . . . 42
3.6 Other approaches . . . 42

4 THE DEEP QUALITY-VALUE FAMILY IN DEC-POMDP 45
4.1 Introduction . . . 45
4.2 Methods . . . 46
4.3 Experiments . . . 47
4.4 Results . . . 48
4.5 Discussion and future work . . . 52
5 INFRASTRUCTURE MANAGEMENT PLANNING 53
5.1 Introduction . . . 53
5.2 Related work . . . 56
5.3 IMP-MARL: A suite of Infrastructure Management Planning environments 56
5.3.1 Environments formulation . . . 56
5.3.1.1 States and observations . . . 56
5.3.1.2 Actions and rewards . . . 57
5.3.1.3 Real-world data . . . 57
5.3.2 IMP-MARL environments . . . 57
5.3.2.1 k-out-of-n system . . . 58
5.3.2.2 Correlated k-out-of-n system . . . 59
5.3.2.3 Offshore wind farm . . . 59
5.4 Modelling infrastructure management in IMP-MARL . . . 60
5.4.1 Deterioration models . . . 60
5.4.1.1 Correlated and uncorrelated k-out-of-n systems . . . 60
5.4.1.2 Offshore wind farm . . . 61
5.4.2 Inspection models . . . 61
5.4.2.1 Correlated and uncorrelated k-out-of-n systems . . . 61
5.4.2.2 Offshore wind farm . . . 62
5.4.3 Transition models . . . 62
5.4.4 Reward model . . . 64
5.5 Experiments . . . 64
5.5.1 Tested methods . . . 64
5.5.2 Experimental setup . . . 65
5.6 Results . . . 66
5.7 Discussion and future work . . . 69

III COOPERATE AGAINST AN OPPOSING TEAM 71
6 COMPETITION 73
6.1 Introduction . . . 73
6.2 Solutions . . . 74
6.3 History . . . 75
6.4 Self-play and population-based training . . . 78
7 TWO-TEAM MARKOV GAME 81
7.1 Introduction . . . 81

7.2 Two-team Markov game . . . 82
7.3 Competitive StarCraft Multi-agent challenge . . . 82
7.4 Learning scenarios and performances criteria . . . 83
7.5 Experiments . . . 84
7.6 Results . . . 85
7.7 Discussion and future work . . . 90

IV CONCLUSION 93
8 CONCLUSION 95

V APPENDIX 99
A IMP-MARL SUPPLEMENTARY MATERIALS 101
A.1 Repository, license, data, and documentation . . . 101
A.2 Implemented options . . . 101
A.3 Experimental details . . . 102
A.3.1 Description of the parameters set up in the experiments . . . 102
A.3.2 Statistical analysis of the variance associated with the number of test episodes . . . 105
A.3.3 Hardware and experiments duration . . . 106
A.4 Additional benchmark results . . . 110
B TWO-TEAM MARKOV GAME SUPPLEMENTARY MATERIALS 115
B.1 Training parameters . . . 115
B.2 Training time . . . 115
C REFERENCES 117

1 INTRODUCTION

We make decisions every day. So, let us build some intuition about this decision-making process through the following example. Consider yourself driving a car and arriving at a crossroad. In front of you, there is a traffic sign. You predict it is a stop sign based on your perception and knowledge. This prediction involves reasoning and has no impact on your environment. As suggested by the traffic sign, you then take the action to stop the car. This also involves reasoning but impacts your environment. Indeed, you push on the brakes, and the car decelerates until it stops.

These decisions are based on what you learned, even if some decisions can be made without knowledge. Indeed, you probably studied the traffic signs to know it was a stop one. You practised your driving skills to obtain your driver's license by trial and error, but hopefully also with the help of a supervisor. You did acquire these skills by learning. You learned with supervision, having examples of decisions and feedback to highlight whether some decisions were right or wrong. However, you also learned by interacting with and testing your environment. Many may recall a child throwing objects off a table, discovering gravity in action, possibly motivating the quote from Sutton and Barto [2018] "The idea that we learn by interacting with our environment is probably the first to occur to us when we think about nature of learning". In summary, you learned by yourself and with the help of others. Nevertheless, this manuscript is not about how humans learn to make decisions. It is about how computers can learn to make decisions.

When a computer makes decisions, it is called artificial intelligence (AI). However, AI is not only about computers making decisions. Nowadays, AI has many definitions. Choosing which one is better is beyond the scope of this manuscript. However, we will provide some before focusing on the one of interest. In "Artificial Intelligence: A Modern Approach", Russell and Norvig [2010] rely on several definitions to propose a classification of AI into four categories. They classify the definitions of AI as making computers either think like humans or think rationally or act like humans or act rationally. Computers thinking like humans is the field of cognitive sciences where theories of the human mind are built, and computers thinking rationally is the field of logic, based on knowledge representation and inspired by the syllogisms invented by Aristotle. This manuscript does not address these two types of AI. It also does not address acting humanly, which we justify hereafter, but focuses on AI acting rationally. Rationality means that it maximises a measure of the performance of the decision. Therefore, these types of AI are built to make decisions to achieve a predefined goal. We commonly define an AI making decisions as an agent.

When considering whether AI can act like humans, a well-known proposition coming up is the Turing test. An agent succeeds in the Turing test if, during a written conversation with it, the human cannot tell whether its interlocutor is an AI. This test has been extended to include video signals or physical interaction to test more complex problems. However, many researchers did not focus on creating AI to succeed in this test. The following self-explanatory quote from Russell and Norvig [2010] may convince many: "Aeronautics is not defined as the field of making machines that fly so exactly like pigeons that they can fool even other pigeons".

This manuscript concerns agents acting rationally but also concerns learning. We still delay the definition of learning, and hereafter, we present that acting rationally can be achieved without learning. Indeed, creating an agent to solve Sudoku puzzles is possible by exploiting the game rules. It will sequentially enumerate all cells' available numbers and assign a number to cells that end with only one possible number. Many problems can be solved without learning, but some may be challenging. For example, an agent can play chess by enumerating all the possibilities from a given board situation to choose the best move. But the number of possible chess games is 10^{120} [Shannon, 1950], and enumerating all of them is intractable. However, creating an agent that plays chess is possible if it enumerates only the games of interest and does it in parallel to make it swift. This is how DeepBlue [Campbell et al., 2002] plays chess at the level of the best human players without learning. Even though it is possible to solve complex problems by ingeniously exploiting game rules, the complexity of some others leads to the development of agents learning to make decisions.

When there is learning in AI, we call it machine learning (ML): "Machine learning is a subset of AI that learns to make decisions by fitting mathematical models to observed data" [Prince, 2023]. ML is divided into different areas based on available data and goals. For example, an agent can learn to predict the type of a traffic sign based on its picture. To achieve this, a model can be built with the help of a dataset of pictures and their corresponding labels. In this case, the model is a mathematical function that maps a picture, a collection of pixel values, to a type of traffic sign. The model performance is initially poor but improves during training. It learns! During the training phase, the system predicts the types of images from the dataset. By using actual and predicted types of traffic signs, the model is refined to improve it. Such a solution is called supervised learning (SL) because the labels supervise the learning. Another area of ML is unsupervised learning when labels are not provided with the data. Indeed, the lack of labels induces the absence of supervision, but the model can still learn patterns in the image structure instead of learning to predict its label. Removing part of the image and training the model to predict the missing pixel is an example of unsupervised learning called inpainting. Unsupervised learning serves many purposes. For example, it is common to pre-train a model with unlabelled data, which are typically abundant, to establish a foundation of knowledge before introducing labels, which are often more challenging to acquire, to train the model on a specific task using supervised learning.

At the beginning of this introduction, we presented the decision-making process by highlighting parts based on their impact. Learning to predict something via supervised learning allows one to make decisions that impact the environment. Indeed, it is possible to train a model that predicts the strength to apply to your car pedals based on a camera in the car. We also considered chess. A collection of chess games would also allow an agent to learn to predict the best move based on the ones in the dataset. Aside from the complexity of acquiring such a dataset, these approaches do not allow AI to learn to make decisions but to learn to replicate the decisions, called imitation learning. Doesn't this remind you of the acting humanly approach of AI defined previously? Moreover, one crucial aspect left aside until now is the sequential aspect of this rational decision-making. Indeed, decisions might impact the future. Most of the time, as in chess, a sequence of decisions is required to achieve a goal. We need a systematic approach to train an agent to make sequences of rational decisions. Such an approach exists and is called reinforcement learning (RL). This is a third area of machine learning, and most of the time, any ML problem is classified as one of the three approaches presented: supervised, unsupervised or reinforcement learning.

In reinforcement learning, an agent acts and receives a reward after taking action. The reward is the action's outcome, a numerical feedback. The agent aims to perform actions that provide the maximum possible accumulated reward. It learns it by trial and error, typically updating its policy based on the received reward after each action. Back to our example, when not stopping at a stop sign at a crossroad, you can cause an accident, an excellent example of a situation that would provide a negative reward. But whether you decide to break or not to do anything, your environment is evolving. Indeed, in addition to receiving rewards, the agent's environment can evolve due to its action. We then say that the state of the environment evolves based on the action. The agent is part of the environment and interacts with it, somehow defined inside and outside of it. Nevertheless, this dynamic nature of the environment adds a new dimension to the learning process because the agent needs to learn to make a sequence of optimal decisions. Note that it is possible to design an RL problem without any environment evolution.

Sudoku puzzles can be solved and chess played without learning, but the agent needs to know the game's rules. In RL, this is not usually the case. The agent does not know how an action will affect its state neither the reward it will receive. It just receives the reward after taking an action and learns to maximise it. One of the challenges of RL is that agents face the exploration-exploitation dilemma. They must balance trying out new actions to discover potentially better policies (exploration) and sticking to actions that have yielded high rewards in the past (exploitation). Another common challenge lies in the environment, which can also involve delayed rewards. The consequences of an action may not be immediate, typically because a sequence of precise actions is required to achieve something. This delay makes it more challenging for the agent that has to learn whether a reward is due to a previous action or the last one. This is referred to as the credit assignment challenge.

Reinforcement learning has applications in various domains. The recommender system [McInerney et al., 2018] is an excellent example of an RL application. Robotics is another application where an agent learns to control a robot, such as training a robotic hand to solve a Rubik's cube [Akkaya et al., 2019]. Designing chips is a complex sequential decision-making problem that can also be solved with the help of RL [Mirhoseini et al., 2021]. RL can be applied in healthcare, typically to decide treatment dosage through time [Miotto et al., 2018]. Finally, games have been a testbed for RL for a long time. Some significant breakthroughs in RL have been achieved thanks to games, such as attaining human performance in Atari games [Mnih et al., 2015]. They offer the best environment for RL because they are natural simulators. Indeed, the trial-and-error nature of RL requires that an agent extensively interacts with its environment.

We have presented above applications of RL where a single agent acts rationally to achieve a predefined goal. However, back to our first example, when driving your car you are not alone in that environment. Other people are also sharing the roads, and your decisions are also based on them, on their behaviour. When there are multiple agents in the environment, many things change. How these agents influence the environment must be considered because a multi-agent system has several types of dynamics. Typically, agents can take actions simultaneously or sequentially. For example, everyone makes decisions simultaneously on the road, while chess is a turn-based game in which agents make their decisions one after the other. Moreover, the number of agents is also different in these two examples. In chess, there are only two agents, while on the road, there can be many more. This illustrates the wide range of problems that a multi-agent system can represent, from a game with two agents to a real-world environment with an unknown number of agents.

Multi-agent reinforcement learning (MARL) extends single-agent reinforcement learning (SARL) when multiple agents learn in the environment. In such a system, each agent receives a reward and learns to act to maximise its rewards. Learning is the crucial differentiation between multi-agent systems and MARL. Suppose only one agent is learning in an environment comprising several others. The other agents then have a stationary policy that does not change over time, and they can be considered part of the environment. However, when several agents learn, their policies change over time. From the point of view of each agent, the environment dynamic is constantly changing. This induces non-stationarity in the learning process. Typically, in a rock-paper-scissor game, if one always chooses rock, one may always choose paper. But if the first changes to scissors, the second may finally change to rock. This cycle can last forever, and one needs to decide when to stop. Finding an equilibrium where strategies stop evolving is indeed a challenging solution. This highlights a first challenge in MARL.

A second challenge is determining the criterion required to assess the optimality of policies. In SARL, it is straightforward. the agent must receive the highest sum of rewards, achieved when it finds the optimal policy achieving the maximum sum of rewards. But how can this be posed in a multi-agent environment when several agents want to maximise

their reward? The solution to finding an equilibrium is achieved when agents do not benefit from a better reward by changing their policy. How to define this equilibrium and whether there are several of them is a dedicated research problem in MARL. Moreover, finding the equilibrium providing the maximum sum of rewards to all agents can be challenging.

The third challenge is the number of agents. The environment's dynamic is often a function of every agent's action, leading to a combination of actions that scales exponentially with the number of agents. Designing methods to train these agents that scale is challenging because it usually implies functions based on all these possibilities.

A fourth challenge is related to the credit assignment previously introduced. In addition to understanding that a reward may be caused by a series of actions, a challenge in MARL is determining the actions of which agents caused the current reward. Back to the example. If you stop because of a stop sign but your vehicle is damaged by the one following you which did not stop, the corresponding negative reward is not your fault, but the fault is that of the car's driver behind you. Other challenges exist, but these four highlight the additional challenges when extending to MARL [Albrecht et al., 2023].

MARL has been divided into three settings. This division comes from game theory and allows for simplifying the general case to a particular one to reduce the complexity arising from the challenges previously mentioned. Each setting is based on the goal of agents. The first setting, cooperation, involves agents cooperating to achieve a common goal. In the second setting, competition, agents compete to achieve an opposing goal, typically named the zero-sum setting. The third setting is named general-sum and encompasses everything else. This is the general setting and maybe the most challenging because no hypothesis can be exploited to mitigate the problem.

This introduction clearly identifies that we must consider other learning agents when training one in MARL. The reader should be confident that training an agent with an SARL method, ignoring other learning agents, might not yield the best solution to these problems. This research question has existed in the MARL community for quite some time, and we go beyond by providing insights in this manuscript into how we should consider other learning agents in an environment. We focus on two specific settings: training a team of agents to cooperate and training a team of agents to compete against an opposing one. In the cooperative setting, we present two contributions that provide methods designed to train agents to fulfil a cooperative task and how they can be applied in a specific real-world setting. When training a team in an adversarial scenario, we can use the same methods but, again, by considering that there are opponents. This highlights the two main parts of the manuscript, each starting with a background section that provides perspective on the novelties of the presented work. Before diving into the details of these two settings, we provide the necessary background and notations of RL. In the following sections, we provide a more detailed outline of this manuscript, followed by a disclaimer on the publications that form its foundation and, finally, the industrial context in which some of the work presented has been conducted.

1.1 OUTLINE

This manuscript is divided into four parts. The necessary background is presented in Part I, providing an overview of RL. In Part II, we present works related to cooperation, one of the MARL settings, while Part III extends these topics to a specific framework where two teams compete. Finally, Part IV concludes this manuscript, retrospectively reviewing the contributions presented in this manuscript.

PART I Background:

Chapter 2 presents the general MARL framework and details its different settings and challenges. More importantly, this chapter presents the foundations of MARL issued from SARL. This chapter is a must-read for those unfamiliar with reinforcement learning. In contrast, these concepts should be well-known for the familiar ones, and Chapter 2 serves as a notation introduction.

PART II Learn to cooperate:

Chapter 3 provides the necessary material to understand the challenges of cooperation in MARL. It is a mix of background introduction and literature review, as this chapter introduces many concepts and methods from the literature.

Chapter 4 presents four methods based on the Deep-Quality-Value family of algorithms designed to train cooperating agents.

Chapter 5 introduces infrastructure management planning, a real-world application that can be tackled in this cooperative framework. In this application, inspections, repairs, and/or retrofits should be timely planned to control the risk of potential system failures.

PART III Cooperate against an opposing team:

Chapter 6 presents how competition is classically modelled and solved in MARL. It serves the same purpose as Chapter 3 by being a mix between background and literature review.

Chapter 7 presents a particular case of the general-sum settings where two teams compete and how to train them. The goal is to study how to train the team to be resilient against multiple strategies.

PART IV Conlusion:

Chapter 8 is the final chapter and addresses the closing remarks.

1.2 PUBLICATIONS

This manuscript is built upon existing research, integrating insights from numerous cited sources. Notably, three central chapters represent adaptations of specific peer-reviewed publications.

- Chapter 4 is an adapted version of the publication *QVMix and QVMix-Max: extending the deep quality-value family of algorithms to cooperative multi-agent reinforcement learning*, P. Leroy, D. Ernst, P. Geurts, G. Louppe, J. Pisane, and M. Sabatelli. AAAI-21 Workshop on Reinforcement Learning in Games, 2021 [Leroy et al., 2021].

- Chapter 5 is an adapted version of the publication *IMP-MARL: a suite of environments for large-scale infrastructure management planning via MARL*, P. Leroy, P. G. Morato, J. Pisane, A. Kolios, and D. Ernst. Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2023 [Leroy et al., 2023].

- Chapter 7 is an adapted version of the publication *Value-based CTDE methods in symmetric two-team Markov game: from cooperation to team competition*, P. Leroy, J. Pisane, and D. Ernst. Deep Reinforcement Learning Workshop NeurIPS, 2022 [Leroy et al., 2022].

1.3 CONTEXT

To provide a backstory, part of the work presented in this manuscript has been developed during the IRIS project with an industrial consortium of Belgian companies financed by the Walloon region under the convention 7977. IRIS stands for Intelligent Recognition Information System, and the partners were John Cockerill Défense, ACIC, Multitel, Belgian Royal Military Academy and the University of Liège. The IRIS project aims to enhance surveillance capabilities by integrating machine learning. Through detection, recognition, and behaviour analysis modules, IRIS seeks to empower operators with timely, actionable information to make optimal decisions in both military and civilian contexts, impacting the safety of individuals, sensitive sites, and equipment integrity. Specifically, we explored multi-agent reinforcement learning to provide decision-aid to operators in the context of military missions. The IRIS project served as a case in our studies on training a team against an opposing team to learn how military assets should cooperate.

Part I

BACKGROUND

2

FOUNDATIONS OF MULTI-AGENT REINFORCEMENT LEARNING

Outline

This chapter provides a broader overview of reinforcement learning. We first define reinforcement learning in Section 2.1 and define a stochastic game, a multi-agent framework, in Section 2.2. Multi-agent reinforcement learning is commonly divided into three settings depending on the agents' relative goals described in Section 2.3. Section 2.4 provides essential details on single-agent reinforcement learning, the **not so** particular case lying at the foundations of multi-agent reinforcement learning. Section 2.5 concludes this chapter with a discussion on partial-observability.

2.1 INTRODUCTION

As introduced in Chapter 1, reinforcement learning (RL) is a machine learning setting to solve decision-making problems, and we hereafter rephrase several RL definitions:

- Reinforcement learning is learning solutions of a sequential decision process by repeatedly interacting with an environment [Albrecht et al., 2023].
- Reinforcement learning is a framework for sequential decision-making, one core topic of ML [François-Lavet et al., 2018].
- Reinforcement learning is a method to solve problems where decisions are applied to a system over time to achieve a desired goal [Buşoniu et al., 2010].
- Reinforcement learning is learning by interacting in its environment to maximise a numerical signal called reward [Sutton and Barto, 2018].

We denote several keywords guiding most RL journeys: environment, interaction, sequence, goal and reward. But we intentionally rephrase the definitions by leaving one missing: the agent. Commonly, an agent is anything capable of acting upon information it perceives from its environment [Russell and Norvig, 2010]. In RL, an agent learns to act by interacting with its environment. Trying to summarise all definitions, we obtain that RL agents learn to act by interacting with their environment and sequentially taking actions that will modify the environment and provide them with a reward.

After defining an agent, we must also consider the number of agents in the environment. Indeed, environments exist where several agents act. Single-agent reinforcement learning (SARL) becomes multi-agent reinforcement learning (MARL) when more than one agent **learns**. We insist on the fact that we consider the number of agents learning. This chapter defines the general MARL framework and follows it with the three standard settings: cooperation, competition, and general-sum. We then provide an in-depth overview of SARL methods, intending to give enough background to the reader unfamiliar with RL and the classical SARL algorithms. We finish with a discussion on partial observability, an essential topic in RL, because agents often have uncertainty about their perception of the environment, especially in MARL.

Since this chapter aims to provide a broader overview of RL and its central concepts, we intentionally skip some details, such as mathematical developments, demonstrations, or definitions. However, we always try to refer the reader to references that go beyond our introductions. We acknowledge that the background chapters of this manuscript take a lot of inspiration from the cited works, especially from two of them: "Reinforcement learning: An introduction" [Sutton and Barto, 2018], well established in the community and "Multi-Agent Reinforcement Learning: Foundations and Modern Approaches" [Albrecht et al., 2023], a recent book on the foundations of multi-agent reinforcement learning.

2.2 STOCHASTIC GAME

The stochastic game (SG) [Shapley, 1953] is at the foundation of MARL. In an SG, a set of agents interact with the environment by observing its state, choosing actions, and receiving rewards over a sequence of time steps. We define a stochastic game by a tuple $[n, \mathcal{S}, \mathcal{U}, R, P, \gamma, p, T]$. The interaction of agents in an SG is presented in Figure 2.1. The set of agents is $\mathcal{A} \equiv \{1, .., n\}$ so that a specific agent is denoted by a_i, or directly by i, with $i \in \mathcal{A}$. When not referring to a specific agent, an agent is denoted by a. At each time step t, each agent a selects an action $u_t^a \in \mathcal{U}^a$ based on the state of the environment $s_t \in \mathcal{S}$ with a probability given by its policy $\pi^a(u_t^a|s_t) : \mathcal{S} \to \Delta(\mathcal{U}^a)$, where $\mathcal{S}$ is the state space, and $\mathcal{U}^a$ is the action space of agent a. The n actions selected by each agent form the joint action $\boldsymbol{u_t} \in \mathcal{U}$, where the joint action space is $\mathcal{U} \equiv \bigtimes_{i \in \mathcal{A}} \mathcal{U}^{a_i}$. We also denote the joint policy by $\boldsymbol{\pi} = (\pi^{a_1}, ..., \pi^{a_n})$. As a consequence of agents taking $\boldsymbol{u_t}$, the state of the environment s_t transits to a new state s_{t+1} with a probability $P(s_{t+1}, s_t, \boldsymbol{u_t})$ defined by the stochastic transition function $P : \mathcal{S} \times \mathcal{U} \to \Delta(\mathcal{S})$. At the same time as the state transitions, each agent receives a reward denoted $r_t^{a_i} = R(s_{t+1}, s_t, \boldsymbol{u_t}, i)$ defined by the reward function $R : \mathcal{S} \times \mathcal{S} \times \mathcal{U} \times \mathcal{A} \to \mathbb{R}$. The goal of each agent a_i is to maximise its expected return, which is the expected sum of discounted rewards $\mathbb{E}_{\boldsymbol{\pi},p,P}\left[G_0^{a_i}\right] = \mathbb{E}_{\boldsymbol{\pi},p,P}\left[\sum_{t=0}^{T-1} \gamma^t r_t^{a_i}\right]$, where T is the time horizon, p is the initial state distribution, $\gamma \in [0, 1]$ the discount factor and $G_t^a = \sum_{j=t}^{T-1} \gamma^{t-j} r_j^a$. The time horizon T can be infinite but in this thesis, it is considered finite, defining the length of an episode.

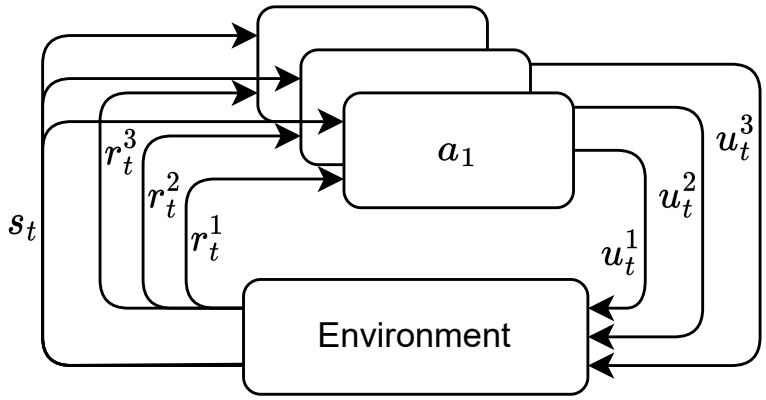


Figure 2.1: Interaction of three agents with the environment in a stochastic game [Shapley, 1953]. Agents $i \in \{1, .., 3\}$ have access to the state s_t to select actions u_t^i. As a consequence, they each receive a reward r_t^i and the environment transitions in a new state s_{t+1}.

Intuitively, the discount factor γ defines the importance of future reward. Finally, we insist that an agent's return depends on the joint action taken and not only on its own.

Before discussing the challenges of learning in an SG, we hereafter provide some comments about its definition. Since the transition function is defined as stochastic, the reward function is always based on the following state obtained. Still, we sometimes denote it $R(s_t, \boldsymbol{u_t}, i)$ irrespective of this new state to enhance readability. Moreover, the expected return is often denoted $E_{\boldsymbol{\pi}}[G_0]$ leaving the initial state distribution p and the transition function P dependencies implicit. An essential characteristic of MARL is that agents do not always observe the state s_t of the environment to select an action. This partial observability is further developed in Section 2.5. Finally, in the literature, an SG is sometimes called a Markov game (e.g. [Littman, 1994]).

When multiple agents learn in the environment, many challenges arise, and we describe four. One challenge is the non-stationarity of the learning agents. Since all agents learn, they all update their strategy over time. One typical risk is that agents may adapt to the strategy of others in an infinite cycle, as in the rock-paper-scissor example of Chapter 1. One example solution to this problem is to find a Nash equilibrium [Nash Jr, 1950]. Such equilibrium is obtained when no agents are interested in changing their policy, meaning that if one changes its policy, its sum of discounted rewards will decrease. There is always at least one Nash equilibrium. When there are several, one may provide a better sum of discounted rewards to all agents than another. Identifying the optimality of policies, which is connected to finding the best equilibria, is a significant challenge in MARL. This task is often split into two challenges: finding and evaluating equilibria. Both aspects remain pretty difficult. To reach such an equilibrium, agents' objectives may change from finding the maximum sum of discounted rewards to obtaining an equilibrium. This manuscript details these alternative solutions and their optimality in Part III. Another challenge is called credit assignment. As highlighted in the SG definition, despite provid-

		Agent 1	
		A	B
Agent 0	A	$(-1, -1)$	$(-5, 0)$
	B	$(0, -5)$	$(-2, -2)$

Table 2.1: Prisoner dilemma payoff matrix.

ing individual rewards, they provide feedback characterising all agent's actions. It is often challenging for an agent to credit its actions or the ones of others. Finally, the number of agents is critical in a MARL problem and represents the fourth challenge. Indeed, the size of the joint action space $|\mathcal{U}|$ scales exponentially with the number of agents. As it will be highlighted in this manuscript, many components of the MARL problem are built as a function of the joint action space. A typical example is to decide if a joint policy is an equilibrium. These four challenges represent the main challenges in MARL identified by Albrecht et al. [2023]. As mentioned in the introduction, specific settings exist in MARL, and addressing these challenges can be mitigated when considering additional hypotheses on the structure of the stochastic game, as detailed in the next section.

2.3 MULTI-AGENT SETTINGS

The stochastic game definition proposed in Section 2.2 provides a general framework for multi-agent systems. A particular case is the stateless SG played by selecting a single action called the normal-form game. These games are also called matrix games because a payoff matrix, a matrix of reward, can represent them. This section defines the three specific settings commonly distinguished in MARL, and normal-form game examples will be provided to help provide intuition. A typical example is the prisoner dilemma, and its payoff matrix is provided in Table 2.1. When agent 0 chooses the action B, it receives a reward equal to 0 when agent 1 chooses the action A, provided in the bottom left corner.

While MARL takes foundation in RL, it also takes foundation in game theory (GT) [Von Neumann and Morgenstern, 1944] that provided the following classification. The difference in these settings is the relation between the rewards of agents [Albrecht et al., 2023]. The first setting, cooperation, involves agents acting to achieve a common goal. In this setting, agents typically share the same reward. The second setting, competition, is the opposite, where agents pursue an opposing goal. In this case, agents typically receive rewards that sum to a constant since the gain of one is equal to the loss of others. The third setting is named general-sum and encompasses everything else. The stochastic game provides a typical framework for a general-sum problem, while its definition is adapted when it is cooperative or competitive. Not much detail will be provided here because these settings are the concern of further chapters. Following this section, we develop more in-depth details about the single-agent setting in the next Section 2.4.

		Agent 1	
		A	B
Agent 0	A	$(-1, -1)$	$(1, 1)$
	B	$(1, 1)$	$(3, 3)$

Table 2.2: Payoff matrix of a cooperative normal-form game.

2.3.1 *Cooperation*

When agents share the same goal, they cooperate, and it is possible to model with a reward function providing a single reward for all agents instead of a different one per agent. This is called "common reward games" in [Albrecht et al., 2023]. Many problems can be considered as being a cooperative multi-agent setting. Examples include robot coordination (e.g. in [Papoudakis et al., 2021]), train scheduling (e.g., in [Mohanty et al., 2020]), traffic control (e.g. in [Zhang et al., 2019]) but also games (e.g., Hanabi [Bard et al., 2020]). Oroojlooy and Hajinezhad [2023] provide a review of cooperative MARL, including a more detailed list of applications. An example of a normal-form cooperative game with a payoff matrix is provided in Table 2.2.

Part II of this manuscript is dedicated to cooperative settings with a common reward. In Chapter 3, we provide the adaptation of the stochastic game definition to a cooperative setting, followed by examples of environments and methods to solve them. In Chapter 4, we present methods from a specific contribution to solve such a setting, while in Chapter 5, we present a suite of environments which frame a real-world application as a cooperative MARL problem.

2.3.2 *Competition*

When agents have opposite goals, they compete. In competition, any action that benefits one agent incurs a loss to other ones. This setting is also called a zero-sum game [Albrecht et al., 2023] because it is often modelled such that the rewards of all agents sum to a constant at any time. In a two-agent zero-sum game, commonly known as a two-player zero-sum game in the GT literature [Russell and Norvig, 2010], a classical implementation of the stochastic game is to have the reward function providing the same reward to all agents. One agent aims to maximise it while the other tries to minimise it. We also refer to this setting as fully competitive. An example of a normal-form cooperative game with a payoff matrix is provided in Table 2.3. A concrete example is chess with a single reward at the end of the game: 1 if player 1 won, 0 if player 1 lost or 1/2 if players drew. Player 2's goal would be to minimise this reward, while Player 1 wants to maximise it. Other examples include card games (e.g. Poker [Brown and Sandholm, 2018]), board games (e.g. chess, shogi, and Go [Silver et al., 2018] or Stratego [Perolat et al., 2022]) and video

		Agent 1	
		A	B
Agent 0	A	$(0, 0)$	$(-2, 2)$
	B	$(1, -1)$	$(2, -2)$

Table 2.3: Payoff matrix of a competitive normal-form game.

games (e.g. StarCraft II [Vinyals et al., 2019]). Part III of this manuscript is not only dedicated to competitive settings but provides, in Chapter 6, many details on training agents in such setting.

2.3.3 *General-sum*

The third setting includes everything that is not fully cooperative or fully competitive. Apriori, it is impossible to adapt the reward function of the general definition as no hypothesis can be made on the corresponding goals of agents. The prisoner dilemma, whose payoff matrix is presented in Table 2.1, is an example of a not cooperative nor competitive normal-form game. One of the best examples of the general-sum setting is autonomous driving [Dinneweth et al., 2022], simulated, for example, in Nocturne [Vinitsky et al., 2022].

In this manuscript, we are interested in the mixed cooperative-competitive setting where two teams compete against each other. Examples of this particular setting include games where two teams face each other (e.g., Dota 2 [OpenAI et al., 2019]). We model such a setting with a reward function providing only two rewards, one per team. Part III is dedicated to this setting, and we study how methods designed for competition can work alongside cooperation ones. We provide background in Chapter 6, including necessary background from the competitive literature, and then detail one specific contribution in Chapter 7.

2.4 SINGLE-AGENT REINFORCEMENT LEARNING

When a single agent learns in the environment, it is called single-agent reinforcement learning (SARL). The most significant part of the history of RL lies in this setting. Indeed, the foundation of MARL relies on many works proposed initially in the single-agent framework. In this section, we introduce the Markov decision process and discuss some fundamentals of RL, including model-based versus model-free, dynamic programming, value-based and policy-based methods.

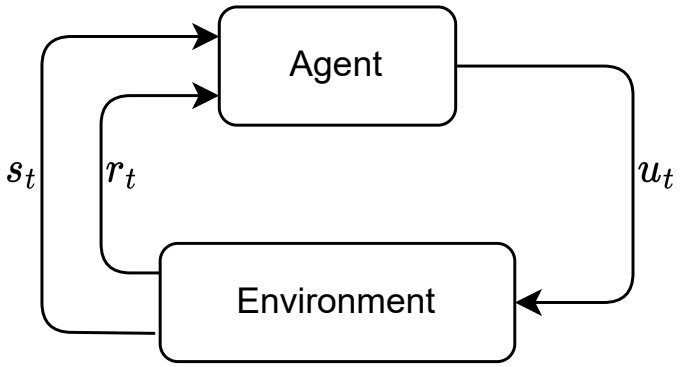


Figure 2.2: Interaction of one agent with the environment in a Markov decision process [Sutton and Barto, 2018]. The agent selects one action u_t based on the state s_t. As a consequence, it receives a reward r_t and the environment transitions in a new state s_{t+1}.

2.4.1 *Markov decision process*

The definition of a Markov decision process (MDP) can be obtained by considering a single agent in a stochastic game (see Section 2.2). We, therefore, do not redefine the SG by removing the agent consideration but still present how this single agent interacts in an MDP in Figure 2.2. Despite being alone, the agent's goal is the same as in an SG, to maximise the expected discounted return over a finite episode $\mathbb{E}_\pi[G_0]$. To maximise its discounted return, the agent needs to learn an optimal policy $\pi^* = \arg\max_\pi \mathbb{E}_\pi[G_0]$. To evaluate a policy, we define the state value function $V^\pi(s) = \mathbb{E}_\pi[G_t|s_t = s]$ and the state-action value function $Q^\pi(s,u) = \mathbb{E}_\pi[G_t|s_t = s, u_t = u]$. While it may appear meaningless for a single-agent environment, it is interesting to observe that the optimal policy is a Nash equilibrium. Indeed, the agent can not profit from changing its policy.

An important property in MDP and SG that is not mentioned in the SG definition is that the next state depends only on the current state and action. This is called the Markovian property. It is important because it implies that a policy based solely on the current state is as good as one based on the history of states and actions. Moreover, we defined the policy as the probability of taking action in a given state. A particular case is a deterministic policy, denoted $\pi(s) : \mathcal{S} \to \mathcal{U}$, or standardly $\mu(s)$.

Usually, two families of methods are identified in RL to learn this optimal policy. Value-based methods learn a value function and derive the policy from it, while policy-based methods directly learn a policy. But before diving into these details, we first discuss whether the model is known, leading to model-based RL, sometimes categorised as a third family of RL methods.

2.4.2 *Model-based or model-free*

When solving an MDP, we distinguish methods based on the knowledge of action outcomes. Indeed, knowing the model of the MDP, one can simulate the environment to evaluate policies [Sutton and Barto, 2018]. "A model is a form of reversible access to the MDP dynamics (known or learned)" [Moerland et al., 2023]. This reversible access means that the agent can execute an action in any state of the MDP and access the outcome anytime. Learning by trial and error without a model is referred to as model-free RL, and this manuscript focuses on methods following this approach.

Planning and RL are two different approaches to finding an MDP's solution when the model is known. They differ in how they represent the solution. Planning is the historical one, where methods build a local representation of their solution, typically by evaluating policies or resolving an optimisation problem around a given state and discarding them after taking action. It typically does not involve learning anything. This is how DeepBlue [Campbell et al., 2002] defeated human champions without learning by evaluating the outcome of many actions before taking each action. Conversely, the RL approach stores a global solution, typically a learned policy or a state-action value function, whether it can access the model. This leads to the distinction between model-based RL and planning of Moerland et al. [2023]. Both have access to the model, but the former learns a global solution while the latter computes local ones. Knowing the model or learning it is not considered in these definitions of planning and RL, and both exist: learning the model and the solution or learning the model and planning a solution based on it.

This distinction between planning and RL does not always agree because, in some methods, a global value function is learned to compute a local representation of the solution, such as in AlphaZero [Silver et al., 2018], considered as model-based RL by some and planning by others. We finally refer to the work of Moerland et al. [2023] that provides many keys to bridge the gap between RL and planning, between model-based and model-free. Nevertheless, the following section introduces dynamic programming, methods requiring a model to compute the optimal policy and providing foundations for the model-free methods.

2.4.3 *Dynamic programming*

Dynamic programming (DP) [Bellman, 1966] methods compute the optimal policies given a model of an MDP [Sutton and Barto, 2018]. We present the Bellman equations to provide some intuitions to help understand later methods and discuss dynamic programming ones. These equations are obtained by developing value functions, highlighting their recursive relationships, for any policy π and $\forall s, u$,

$$\begin{aligned} V^\pi(s) = \mathbb{E}_\pi\left[G_t | s_t = s\right] &= \mathbb{E}_\pi\left[r_t + \gamma V^\pi(s_{t+1}) | s_t = s\right] \\ &= \sum_u \pi(u|s) \sum_{s'} P(s', s, u)(R(s', s, u) + \gamma V^\pi(s')), \end{aligned} \tag{2.1}$$

and,

$$Q^{\pi}(s,u) = \mathbb{E}_{\pi}\left[G_t | s_t = s, u_t = u\right] = \mathbb{E}_{\pi}\left[r_t + \gamma V^{\pi}(s_{t+1}) | s_t = s, u_t = u\right]. \tag{2.2}$$

A policy π' is better than another π if $V^{\pi'}(s) > V^{\pi}(s)$. The optimal policy is the one that is better than all others. While there can be several optimal policies, all are denoted by π^*, and their state value function $V^{\pi^*}(s)$ is the same $\forall s$. Specifically, optimal policies are the unique solution of the Bellman optimality equations, $\forall s, u$,

$$V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s) = \max_{u} \mathbb{E}_{\pi^*}[r_t + \gamma V^{\pi^*}(s_{t+1}) | s_t = s, u_t = u], \tag{2.3}$$

and,

$$Q^{\pi^*}(s,u) = \max_{\pi} Q^{\pi}(s,u) = \mathbb{E}_{\pi^*}[r_t + \gamma \max_{u'} Q^{\pi^*}(s_{t+1}, u') | s_t = s, u_t = u]. \tag{2.4}$$

The optimality equations given in Equation 2.3 represent a system of $|\mathcal{S}|$ equations with $|\mathcal{S}|$ unknown, and solving this system would allow one to compute the optimal policy. This is also the case with the system defined in Equation 2.4. The condition is to have reversible access to the MDP model. Solving either system is usually not considered for computational reasons. This is why, in the following, we consider methods for computing an approximate solution to the Belman optimality equations.

DP is an approach that computes the solutions by iteratively updating approximations until they converge to the real ones. DP can solve Equation 2.1 by updating estimates $v_k(s) \forall s$ with $v_{k+1}(s) = \sum_u \pi(u|s) \sum_{s'} P(s', s, u)(R(s', s, u) + \gamma v_k(s'))$, where $v_0(s)$ can have arbitrary values except for terminal states that are required to equal 0. This is called iterative policy evaluation and is proven to converge if $k \to \infty$. Policy evaluation allows the refinement of any given policy π by exploiting Equation 2.2 to evaluate other actions than $\pi(s)$ using $Q^{\pi}(s,u) = \sum_{s'} P(s', s, u)(R(s', s, u) + \gamma v_k(s'))$. This is called policy improvement. Alternating policy evaluation and improvement allows the iterative computation of the optimal policy, called policy iteration. The convergence time can be a significant drawback in policy iteration. However, the convergence guarantee can be kept while combining evaluation and improvement, leading to the update $v_{k+1}(s) = \max_u \sum_{s'} P(s', s, u)(R(s', s, u) + \gamma v_k(s'))$, called the value iteration method. Sutton and Barto [2018] provides many more details about these DP methods.

In addition to requiring the model knowledge P and R, the complexity of computing all these values increases with the size of the state and the action spaces, often referred to as the curse of dimensionality, which may limit their usage. Sutton and Barto [2018] claim that this take is somehow overrated, explaining that with some assumptions, DP can have a polynomial complexity with $|\mathcal{S}|$ and $|\mathcal{U}|$ and could scale to large spaces. Nevertheless, computing them is impossible for continuous spaces without discretising the spaces, and we are interested in model-free methods because accessing the model knowledge can be challenging. The following section details methods to approximate these value functions without knowing the transition and reward functions.

2.4.4 *Value-based methods*

As introduced, value-based methods are designed to approximate value functions. In a model-free setting, one possibility is to estimate the state value function of a given policy with Monte Carlo estimations. Such evaluation relies on playing many episodes following the policy to determine the average return of a state. This method also converges according to the law of large numbers. Since the model is unknown, improving policies like in DP is impossible, but Monte Carlo simulations can be adapted to estimate state-action value functions similarly. Again, we refer the reader to [Sutton and Barto, 2018] for many more details, such as how to update the value of a state visited several times during an episode. One important drawback of the Monte Carlo method is that it requires waiting for episodes to complete to update estimations. An alternative approach, also model-free, is called temporal difference (TD) learning. It immedialy updates the estimation using $v_{k+1}(s_t) = v_k(s_t) + \alpha[R(s_{t+1}, s_t, u_t) + \gamma v_k(s_{t+1}) - v_k(s_t)]$ where α controls the update size. TD-Learning is the foundation of the value-based methods introduced in this section.

Maybe one of the most popular methods in model-free RL is Q-learning [Watkins and Dayan, 1992], where the state-action value function learned is the optimal one defined as $Q^{\pi^*}(s, u) = \max_\pi Q^\pi(s, u)$, allowing the agent to select actions with the greedy deterministic policy $\pi^*(s) = \arg\max_u Q^{\pi^*}(s, u)$. Q-learning is a tabular method because it maintains $Q(s, u)$ estimations in a table, one value for each state-action pair. It updates these estimations based on themselves, called bootstrapping, like TD-Learning [Sutton and Barto, 2018]. Following the update

$$Q(s_t, u_t) \leftarrow Q(s_t, u_t) + \alpha \left[r_t + \gamma \max_u Q(s_{t+1}, u) - Q(s_t, u_t) \right], \tag{2.5}$$

it is possible to repeatedly update the estimation $Q(s, u)$ while observing new transitions by adding the temporal difference weighted by a learning rate α controlling the update size.

It is important to denote that this algorithm allows approximating $Q^{\pi^*}(s, u)$ independently of the policy used to sample transitions (s_t, u_t, r_t, s_{t+1}). These transition samples are typically generated with an ϵ-greedy policy that takes a random action instead of the greedy one with a probability ϵ, whereas the greedy policy selects the action maximising Q. This is a characteristic of the off-policy methods, as opposed to the on-policy methods, which improves the current policy based on samples only from the current policy. This manuscript considers only value-based methods that are off-policy, but on-policy methods are discussed further in Section 2.4.5. To cite one, SARSA is a well-known on-policy value-based method [Sutton and Barto, 2018].

Despite being off-policy, this iterative process highlights the exploration-exploitation dilemma in RL. Either the agent only plays the action that maximises its currently learned value function, which it exploits, or it chooses a different action and explores the possible outcomes. Balancing between exploration and exploitation is a crucial parameter to train agents.

Back to Q-learning, the table size increases as the state-action space size increases. Therefore, it can become impractical to compute an estimate of $Q(s, u)$ for each state-action pair, requiring function approximations. Aside from the generalisation problem, storing a table for continuous spaces is also impossible. Various function approximators exist, but we restrict ourselves to neural networks in this manuscript.

A neural network is a function $f_\theta : \mathcal{X} \to \mathcal{Y}$ that maps an input ($\in \mathcal{X}$) to an output ($\in \mathcal{Y}$) based on its parameters θ: $y = f(x; \theta)$. These parameters define a composition of differentiable functions, linear or not, allowing optimising the parameters by following the gradient of an objective, commonly called a loss function $\mathcal{L}(\theta)$. To minimise the loss, parameters can be updated by gradient descent: $\theta_{k+1} = \theta_k - \alpha \nabla \mathcal{L}(\theta_k)$. Optimising a neural network is also referred to as training it. Many loss functions exist to train neural networks, depending on the function to be approximated. Nowadays, neural networks are very large, leading to the name of deep learning. We refer to [Zhang et al., 2023] or [Prince, 2023] for many more details. Finally, RL with neural networks is called deep reinforcement learning, e.g. in [François-Lavet et al., 2018]. As it became common to use neural networks in RL, we decided to remove the word "deep" from the taxonomy, as the title of this manuscript should be "Contributions to deep multi-agent reinforcement learning". In the following, we introduce how to train such networks to approximate a value function and, in Section 2.4.5, to approximate a policy.

A standard method in RL, referred to as deep Q-network (DQN) [Mnih et al., 2015], is to approximate $Q(s, u)$ with a neural network θ. This can be achieved by minimising the loss

$$\mathcal{L}(\theta) = \mathbb{E}_B\left[\left(r_t + \gamma \max_u Q(s_{t+1}, u; \theta') - Q(s_t, u_t; \theta)\right)^2\right] \tag{2.6}$$

where B is the replay buffer and θ' is the target network. The replay buffer B stores transitions (s_t, u_t, r_t, s_{t+1}) from which batches of transitions are sampled to update θ [Lin, 1992]. This replay buffer allows updating the neural network with past transitions. The target network θ' is a copy of θ updated periodically that reduces the moving target problem as θ is updated several times before updating θ', e.g., in [Mnih et al., 2015].

For some environments, the max operator in Equations 2.5 and 2.6 can introduce some positive bias, referred to as the overestimation bias. To overcome this bias, a method called Double Q-learning [Van Hasselt, 2010], and adapted to Q-learning with neural networks [Van Hasselt et al., 2016], consists in selecting the action that maximises the updated $Q(., \theta)$ to compute the target state-action value. The corresponding loss is

$$\mathcal{L}(\theta) = \mathbb{E}_B\left[\left(r_t + \gamma Q(s_{t+1}, \arg\max_u Q(s_{t+1}, u; \theta); \theta') - Q(s_t, u_t; \theta)\right)^2\right]. \tag{2.7}$$

Double Q-learning, also called DDQN, is one of the possible improvements of DQN, and we refer to the Rainbow paper [Hessel et al., 2017] that addresses several others. To cite one, the extension to distributional RL, which approximates distributions instead of expected returns, can be of interest [Bellemare et al., 2017; Théate et al., 2023]. Chapter

4 presents the Deep Quality Value algorithms, also standing on the foundation provided by this section.

2.4.5 *Policy-based methods*

Policy-based methods are designed to learn the policy. In this manuscript, we restrict to the subclass of policy gradient methods where a neural network parametrised by θ approximates a differentiable policy $\pi_\theta = \pi(u|s;\theta)$. Policy gradient methods hence update θ to find the optimal policy that maximises the expected return denoted as $J(\pi_\theta) = \mathbb{E}_{\pi_\theta,p,P}[G_0]$. Maybe one of the first methods is REINFORCE [Williams, 1992], which updates $\theta = \theta + \alpha\nabla_\theta J(\pi_\theta)$ by estimating the gradient with Monte Carlo given the policy gradient theorem [Sutton et al., 1999]

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \mathbb{E}_{\pi_\theta}[G_0] = \mathbb{E}\left[\sum_{t=0}^{T-1} Q^{\pi_\theta}(s_t, u_t)\nabla_\theta \log \pi(u_t|s_t;\theta)\right]. \tag{2.8}$$

Estimating $Q(s_t, u_t)$ instead of computing it is a solution proposed by the actor-critic methods [Sutton et al., 1999; Konda and Tsitsiklis, 1999]. This type of method expands upon REINFORCE by incorporating a second neural network, called the critic and denoted by ϕ, that estimates $Q(s_t, u_t;\phi)$ while the actor is the parametrised policy $\pi(u|s;\theta)$. The new gradient provided by incorporating the critic is

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{T-1} Q(s_t, u_t;\phi)\nabla_\theta \log \pi(u_t|s_t;\theta)\right]. \tag{2.9}$$

Moreover, a baseline can be injected into the gradient to reduce variance without changing the gradient's expectation. Usually, the baseline is $V(s)$, independent of the action taken, and $Q(s, u;\phi)$ is replaced by the advantage function $A(s, u;\phi)$ [Weaver and Tao, 2001], leading to

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \mathbb{E}\left[\sum_{t=0}^{T-1} [Q(s_t, u_t) - V(s_t)]\nabla_\theta \log \pi(u_t|s_t;\theta)\right] \\ &= \mathbb{E}\left[\sum_{t=0}^{T-1} A(s_t, u_t;\phi)\nabla_\theta \pi(s_t, u_t;\theta)\right].\end{aligned} \tag{2.10}$$

To avoid approximating both Q and V, it is possible to estimate the advantage with only one neural network either by $A(s_t, u_t;\phi) = Q(s_t, u_t;\phi) - \sum_u \pi(u|s_t;\theta)Q(s_t, u;\phi)$ or by $A(s_t, u_t;\phi) = r_t + \gamma V(s_{t+1};\phi) - V(s_t;\phi)$. This critic is often trained with value-based methods, such as the ones defined in Section 2.4.4. This is why actor-critic methods are sometimes described as a mix between value-based and policy-based methods.

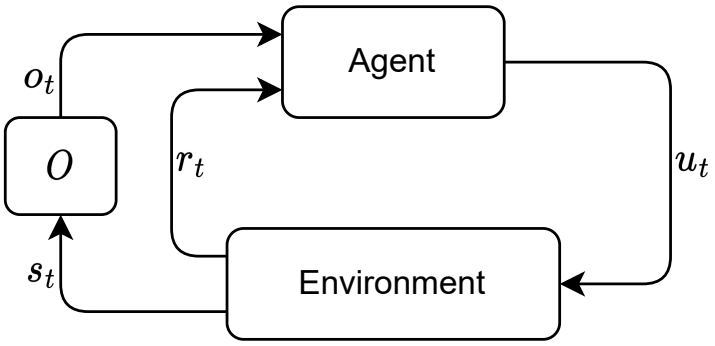


Figure 2.3: Interaction of one agent with the environment in a partially observable Markov decision process [Kaelbling et al., 1998]. The agent does not have access to the state s_t but to an observation o_t defined by the observation function O. Based on o_t, it selects one action u_t. As a consequence, it receives a reward r_t and the environment transitions in a new state s_{t+1}.

Nowadays, advanced policy-based methods relying on the actor-critic paradigm appear to be the most successful. We can cite trust region policy optimisation (TRPO) [Schulman et al., 2015] and its variant proximal policy optimisation (PPO) [Schulman et al., 2017]. Both methods rely on a controlled policy update by constraining the loss of REINFORCE defined in Equation 2.8.

Finally, policy gradient and actor-critic algorithms also need some form of exploration, usually implemented by adding a penalisation term in the loss function. An example is entropy regularisation [Williams and Peng, 1991], where a low entropy of the policy outcomes is penalised. In this context, Bolland et al. [2024] demonstrates that such techniques change the learning objective and increase the probability of updating the policy toward the optimal one.

2.5 PARTIAL OBSERVABILITY

As defined in previous sections, the Markov decision process and the stochastic game are fully observable. Agents have complete access to the state s of the environment and perceive it without uncertainty. In real-world applications, it is not always possible to consider this feasible. Anyone can develop ideas of a partially observable environment, especially given our definition of agents "acting upon information it perceives".

Starting with SARL, the Markov decision process is said to be a partially observable MDP (POMDP) [Kaelbling et al., 1998] when the agent can only access incomplete information about the state. The definition of a POMDP is obtained easily from the MDP definition of Section 2.4.1 by adding an observation space $\mathcal{Z}$ and an observation function $O : \mathcal{S} \to \Delta(\mathcal{Z})$, mapping a state and an observation to the probability of observing the latest. The corresponding interaction diagram of a POMDP is presented in Figure 2.3.

As explained in Section 2.4.1, policies are solely based on the current state in an MDP, thanks to the Markovian property. In a POMDP, the agent's policy π can no longer be a function of the state. Moreover, the observation does not necessarily hold the Markovian property, and acting only based on the current observation could be suboptimal. Therefore, the policy is commonly a function of the history of past observations and past actions $\tau_t = (\mathcal{Z} \times \mathcal{U})^t$, in addition to the current observation, and is denoted $\pi(u_t|\tau_t, o_t) : (\mathcal{Z} \times \mathcal{U})^t \times \mathcal{Z} \to \Delta(\mathcal{U})$. Some authors implicitly include the current observation in τ, allowing them to write $\pi(u|\tau)$ and $Q(\tau, u)$. We sometimes do it to improve readability as well.

To solve a POMDP, a solution is to compute the policy based on a belief $b(s) = \mathbb{P}(s|\tau, o)$, the probability of being in a given state, knowing the history of observations and actions. As in the different methods previously defined, the belief can be implicitly approximated with recurrent neural networks (RNN), such as GRU [Chung et al., 2014] or LSTM [Hochreiter and Schmidhuber, 1997]. These networks typically take time series as input and maintain a hidden state, updated at each time step of one time series, which can be considered a memory. RNNs have many applications, and in POMDP, their hidden state allows to maintain a memory akin to a belief without processing the whole history at each time step. Using RNNs to compute policies is thus a common practice in POMDP, resulting in recurrent policies. It has demonstrated convincing results, such as in recurrent policy gradients [Wierstra et al., 2010] or in deep recurrent Q-network (DRQN) [Hausknecht and Stone, 2015]. We suggest readers interested in details read the pedagogical paper of Lambrechts et al. [2022]. This paper demonstrates that the correlation between the hidden state of RNNs, used to approximate policies, and the belief increases as the training progresses.

Adding partial observability in the definition of the stochastic game leads to the most general framework of MARL, the partially observable stochastic game (POSG) [Hansen et al., 2004]. Its definition is obtained by adding a set of n observation spaces $\mathcal{Z}$ and a set of n observation functions $\mathcal{O}$ in the definition of the SG. A shortcut in this manuscript, and sometimes in the literature, is to consider that these two sets $\mathcal{Z}$ and $\mathcal{O}$ are singleton, such that the observation function $O : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{Z})$ is the same for all agents. An agent's belief in a POSG should be considered differently from that in a POMDP [Oliehoek and Amato, 2016]. This is because, with only the agent's history, it is impossible to compute a belief of the state that is sufficient to take optimal actions. However, it would be possible to achieve this by having access to the histories of all agents. Even when agents can fully observe the current state, they may not observe the actions previously taken by others. However, this manuscript does not address multi-agent belief, and we refer to [Oliehoek and Amato, 2016] for more details. In a POSG, we consider that an agent's policy is a function $\pi^a(u_t^a|\tau_t^a, o_t^a) : (\mathcal{Z}^a \times \mathcal{U}^a)^t \times \mathcal{Z} \to \Delta(\mathcal{U}^a)$, which maps its history $\tau_t^a \in (\mathcal{Z}^a \times \mathcal{U}^a)^t$ and its current observation o_t^a to the probability of taking action u_t^a. As in SARL, such a policy is commonly a recurrent policy approximated with RNN.

Part II

LEARN TO COOPERATE

3

COOPERATION

Outline

This chapter covers the required basics and related works in the cooperative setting. After an introduction in Section 3.1, we define in Section 3.2 the cooperative framework called decentralised partially observable Markov decision process. Section 3.3 presents examples of its application, followed by a precise description of the StarCraft multi-agent challenge, a popular environment suite in this manuscript. We then detail several value-based methods in Section 3.4 and policy-based methods in Section 3.5. We finally discuss other approaches of interest in Section 3.6.

3.1 INTRODUCTION

As introduced in Part I, cooperation is the multi-agent setting of agents sharing a common goal. This second part of the thesis considers the decentralised POMDP (Dec-POMDP) [Oliehoek and Amato, 2016], a framework where all agents receive the same reward. Such a framework is also called common reward games, e.g. by Albrecht et al. [2023]. Its definition is provided in Section 3.2.

Nevertheless, when it comes to cooperation in a multi-agent system, one topic to discuss is whether action selection and agent training are centralised or decentralised. This is also referred to as the modes of execution and training in [Albrecht et al., 2023]. This manuscript considers three combinations: centralised training and execution, decentralised training and execution, and centralised training with decentralised execution.

Since all agents receive the same reward, training a single agent that centrally selects joint actions is possible and would benefit from sharing common knowledge. This is the centralised mode, where one agent is trained with SARL methods. For example, controlling a robotic hand composed of several actuators can be done with a single agent controlling every actuator. However, as presented in Section 2.5, some setting induces partial observability. All agents may not access the same information or perceive only a part of the environment's state. It would then be impossible to consider that one agent can replace all agents and select actions in a centralised mode.

On the contrary, the decentralised execution mode considers that each agent selects its action independently. This can be done irrespective of the information they can access. The corresponding decentralised training mode is when these agents are trained independently. In other words, these agents assume they are the single agent learning in

the environment. We refer to the decentralised mode when training agents independently with SARL to select an action based on their observation.

Finally, the third mode unifies the two formers and is centralised training with decentralised execution (CTDE). It allows decentralised execution, with each agent selecting actions based on their observations, but allows for the exploitation of more information during training. Indeed, RL agents are usually trained in a simulator, having access to every piece of information. Typically, this mode benefits from the state of the environment or exploits the actions made by other agents during training.

From the research question highlighted in Chapter 1, these different modes showcase several ways to consider, or not, the other learning agents when training one. This Part II focuses on methods that exploit the CTDE mode to solve cooperative tasks. This chapter presents Dec-POMDP environments and CTDE methods issued from the literature. The performance of these methods is not compared in this chapter but in both next ones. They present two specific contributions: additional CTDE methods in Chapter 4 and a real-world application of Dec-POMDP in Chapter 5.

3.2 DECENTRALISED PARTIALLY OBSERVABLE MARKOV DECISION PROCESS

As introduced, the definition of the decentralised partially observable Markov decision process (Dec-POMDP) [Oliehoek and Amato, 2016] can be derived from the partially observable stochastic game definition. It is the same, except the reward function maps to a single reward common to all agents.

We define the Dec-POMDP by a tuple $[\mathcal{S}, \mathcal{Z}, \mathcal{U}, n, O, R, P, \gamma, T, p]$, where n agents a_i, $i \in \mathcal{A} \equiv \{1, .., n\}$, simultaneously choose an action at every time step t. The interaction of the agents with the environment in a Dec-POMDP is presented in Figure 3.1. The state of the environment is $s_t \in \mathcal{S}$ where $\mathcal{S}$ is the set of states. The observation function $O : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{Z})$ maps the state to the probability of agent a to perceive the observation $o_t^a \in \mathcal{Z}$ at time t, where $\mathcal{Z}$ is the observation space. Each agent selects an action $u_t^a \in \mathcal{U}^a$ based on its policy $\pi^a(u_t^a|\tau_t^a, o_t^a) : (\mathcal{Z} \times \mathcal{U}^a)^t \times \mathcal{Z} \rightarrow \Delta(\mathcal{U}^a)$, which maps its history $\tau_t^a \in (\mathcal{Z} \times \mathcal{U}^a)^{t-1}$ and its observation o_t^a to the probability of taking action u_t^a. The joint action space is $\mathcal{U} \equiv \bigtimes_{i \in \mathcal{A}} \mathcal{U}^{a_i}$ and the joint or team policy is denoted by $\boldsymbol{\pi} = (\pi^{a_1}, ..., \pi^{a_n})$. After the joint action $\boldsymbol{u_t} \in \mathcal{U}$ is executed, the transition function determines the new state with probability $P(s_{t+1}|s_t, \boldsymbol{u_t}) : \mathcal{S} \times \mathcal{U} \rightarrow \Delta(\mathcal{S})$, and $r_t = R(s_{t+1}, s_t, \boldsymbol{u_t}) : \mathcal{S} \times \mathcal{S} \times \mathcal{U} \rightarrow \mathbb{R}$ is the team reward obtained by all agents. The goal of agents is to find their optimal policy that maximises the expected return during the entire episode of T time step $\mathbb{E}[G_0|\boldsymbol{\pi}, p, P]$ where $G_0 = \sum_{t=0}^{T-1} \gamma^t r_t$ and p is the initial state distribution. The optimal joint policy is denoted by $\boldsymbol{\pi^*} = \arg\max_{\boldsymbol{\pi}} \mathbb{E}[G_0|\boldsymbol{\pi}, p, P]$, achieved if all agents plays the optimal policy.

We introduced the main challenges in MARL and will discuss hereafter the main impact of providing a common reward, which is to increase the complexity of the credit assignment problem. Indeed, this can be considered the most complex case because all agents receive the same feedback on their actions. The non-stationarity of learning agents

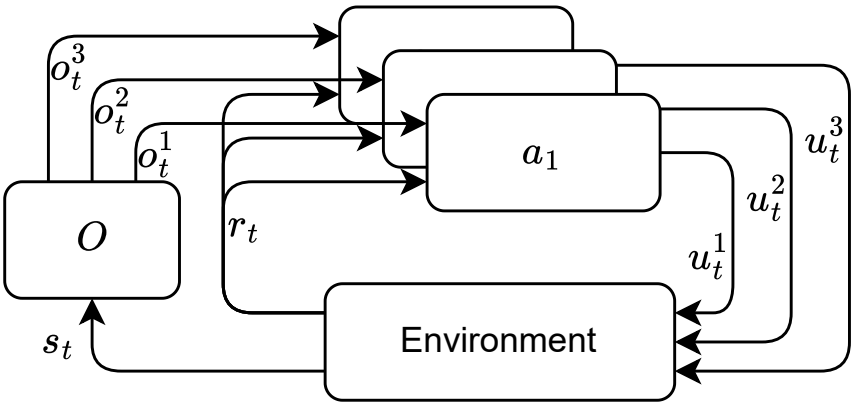


Figure 3.1: Interaction of three agents with the environment in a decentralised partially observable Markov decision process [Oliehoek and Amato, 2016]. Agent do not have access to the state s_t but to an observation o_t^a defined by the observation function O. Based on o_t^a, each agent selects one action u_t^a. As a consequence, they all receive the same reward r_t and the environment transitions in a new state s_{t+1}.

and their number remains a problem. However, choosing between two equilibria has become a trivial problem because one agent will never be disadvantaged compared to others. Nevertheless, the common reward is not the single assumption already discussed. Indeed, the centralised training mode also helps alleviate the complexity of these challenges. In contrast, the decentralised mode does not consider additional assumptions besides the common reward. The introduction may already provide some insights about this claim, and the following sections will demonstrate it.

Like MDP is the fully observable POMDP, there exist several particular cases of Dec-POMDP based on the observability of the agents. This allows us to define the required assumptions to train a single agent to decide the joint action to make the centralised execution mode possible. When the joint observation can identify the state while the observation of one agent can not, this is a jointly observable Dec-POMDP, also referred to as a decentralised MDP by Oliehoek and Amato [2016]. However, to allow centralised execution, it should be considered that agents can share their observations to make the decision centrally by identifying the state with the joint observation. Note that such an assumption allows centralised execution but does not make the problem of decentralised execution simpler because each agent still does not observe the state [Bernstein et al., 2002]. The direct extreme case is the fully observable Dec-POMDP, where all agents observe the state directly. These assumptions can lead to the specific framework called a multi-agent Markov decision process [Boutilier, 1996].

There are also several variations of factored Dec-POMDP described in [Oliehoek and Amato, 2016] whether the transition, reward, or observation functions can be considered factored, meaning they can be decomposed in agent-wise independent factors. While other particular cases can be found in [Oliehoek and Amato, 2016], their definitions allow

to solve them by benefiting from additional hypotheses. For example, in a completely factored Dec-POMDP, the decentralised mode allows to find the optimal policy, while in complex Dec-POMDP, without such a strong hypothesis, this is not always the case. In the following, we consider CTDE algorithms that tackle the general case of Dec-POMDP, but they may have some foundation in these particular cases.

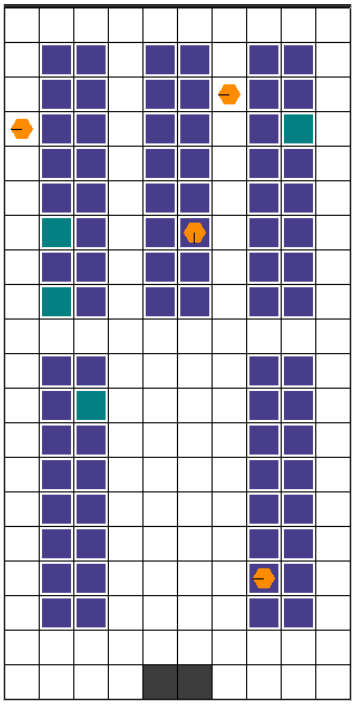
Highlighted in Section 2.5, it is not possible to compute a belief of the state as a function of the agent's history in a Dec-POMDP because agents only access their observations and actions. Not being able to compute a Markovian statistic induces the complexity of solving Dec-POMDP to become challenging, even more than POMDP [Oliehoek and Amato, 2016]. For example, this forces planning methods to consider policies that map histories to action, unlike belief to action in POMDP and RL approaches defined later based on RNN, like in POMDP. Thus, finding the best by enumerating and evaluating all possible deterministic joint policies is only possible for small problems. This is because the number of possible histories grows with the length of episodes, so the number of policies to learn grows doubly exponentially with this length.

Finally, Oliehoek and Amato [2016] present methods that find optimal deterministic joint policy since at least one exists in finite time horizon Dec-POMDP [Oliehoek et al., 2008]. These methods include dynamic programming, which solves the problem iteratively backwards, starting from the last time step of the finite horizon, pruning computed policies along the way to reduce the large number of unnecessary ones. They also include multi-agent A*, taking foundation in search methods [Russell and Norvig, 2010] described in Chapter 6. In short, it is a search in the space of joint policy that uses value functions as heuristics to explore the tree of possible joint policies. Oliehoek and Amato [2016] also present methods that do not guarantee optimality but allow to scale to larger time horizons for the finite horizon while presenting challenges and corresponding methods to the infinite one. Aside from the scalability issues, these methods rely on the model of the environment, motivating the use of model-free RL methods presented later in this chapter.

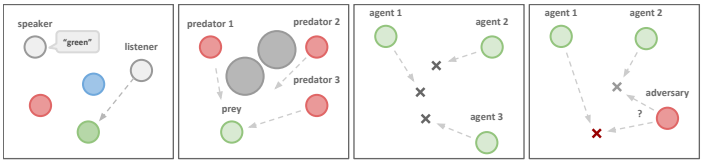
3.3 ENVIRONMENTS

Before discussing CTDE methods in detail, we provide an overview of the existing suites[1]of environments in the literature. We then describe the StarCraft multi-agent challenge (SMAC), perhaps one of the most studied environment suites in the community, but also used in our experiments in Chapter 4. Moreover, Chapter 5 is dedicated to a specific suite of environments, and we leave its description there. Illustrations of the environments described hereafter are provided in Figure 3.2.

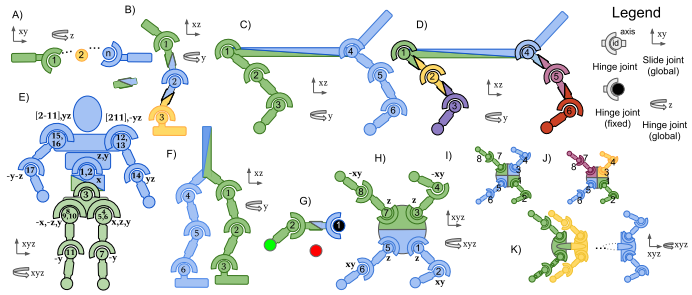
1 In this manuscript, an environment refers to an instance of a particular framework (here, of a Dec-POMDP). We commonly see it in the literature as either a collection of environments or an instance of one. We use "suite" to distinguish between these two.



(a) Robot warehouse.

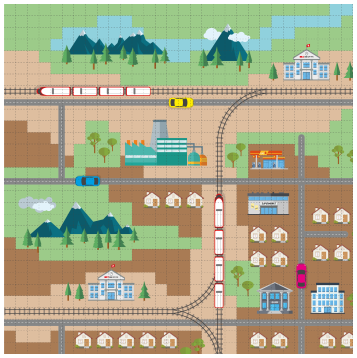


(b) Multi-agent particle environments.

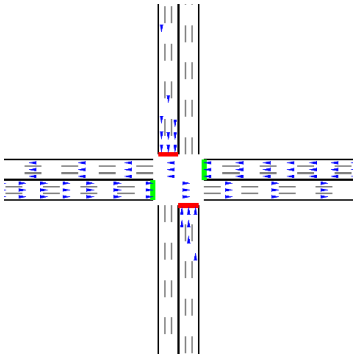


(c) Multi-agent MuJoCo.

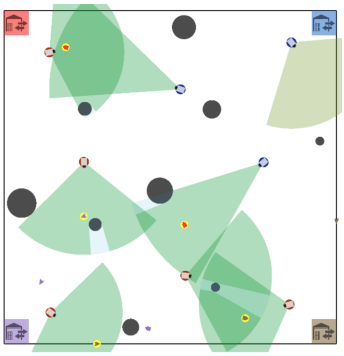
(d) Google research football.



(e) Flatland.

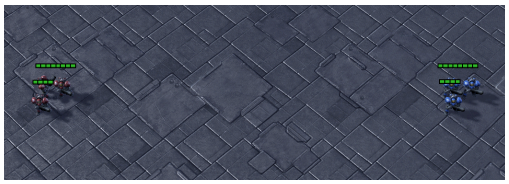


(f) CityFlow.

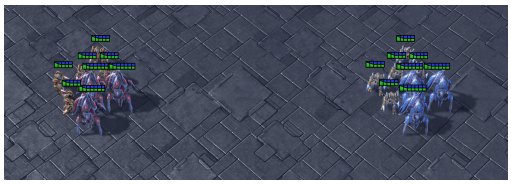


(g) MATE.

Figure 3.2: Cooperative suites of environments defined as Dec-POMDP.



(a) 3m



(b) 3s5z

Figure 3.3: Two environments of the StarCraft multi-agent challenges.

The multi-agent particle environments (MPE) [Lowe et al., 2017] is a popular suite in the MARL community with cooperative scenarios. In MPE, particles move in a 2D grid with continuous action and state spaces. This suite comes with all types of multi-agent settings: competition, cooperation, with and without communication. A second environment with continual action space is MaMuJoCo [Peng et al., 2021]. MuJoCo [Todorov et al., 2012] stands for multi-joint dynamics with contact. It is a famous physics-based simulator to learn to control, for example, a humanoid to run. MaMuJoCo essentially factorises MuJoCo's decisions by decentralising the decision, typically having a different agent for each leg. It is an example of a SARL environment extended to MARL. Other cooperative environments based on game simulators include the Hanabi Challenge [Bard et al., 2020], a "cooperative solitaire" between two and five players, and Google Research Football [Kurach et al., 2020], a football game simulator.

Cooperative MARL methods are primarily benchmarked on these games and simulators, but real-world applications also exist. CityFlow [Zhang et al., 2019] are traffic control environments where agents control traffic lights in a city. In Flatland [Mohanty et al., 2020], agents control trains to solve a scheduling problem, avoiding collisions by coordinating trains not to take the same routes. The multi-robot warehouse [Papoudakis et al., 2021; Christianos et al., 2020] simulates warehouse agents needing to deliver requested goods. Multi-Agent Tracking Environment (MATE) [Pan et al., 2022] is a target coverage control problem where cameras are controlled to detect all targets. A mixed cooperative-competitive game can be made by controlling the targets. Many more examples of cooperative MARL applications exist. Many can be found in the books of Oliehoek and Amato [2016] and Albrecht et al. [2023], in the review on cooperative MARL done by Oroojlooy and Hajinezhad [2023], such as resource allocation, stock market,...

3.3.1 *StarCraft multi-agent challenge*

The StarCraft multi-agent challenge (SMAC) [Samvelyan et al., 2019] and its improved version SMACv2 [Ellis et al., 2023] are probably the most studied suite of environments with CTDE methods. SMAC is based on the StarCraft II Learning Environment [Vinyals et al., 2017], an RL environment to play StarCraft II (SC2). StarCraft is a strategy video game in which players compete by managing units, gathering resources, building an army, and defeating opponents. In such games, micro-management refers to unit management, unlike macro-management, which involves resource management. Unlike playing the real game of StraCraft like in AlphaStar [Vinyals et al., 2019], SMAC is a suite of micro-management challenges where an independent agent controls each game unit. Many scenarios exist in SMAC, also called "maps", and all involve training a team to achieve a common goal. It is to defeat an adversarial team controlled by the game's deterministic and stationary policy built-in AI. Figure 3.3 shows the initial configurations of two maps where two teams face each other. Also note that in Chapter 7, we present a contribution where SMAC has been modified to train both competing teams.

Hereafter, we thoroughly define the elements of the Dec-POMDP implemented in SMAC. Agents have partial observability, characterised by a sight range, a circle inside which they can observe other agents. An agent observes information about itself: its remaining hit points and shield points and four booleans representing the direction it can move in (NSWE). It also observes information about other agents within its sight range: the relative distance, relative x, relative y and the remaining hit points and shield points. If the other agent is an ally, it also observes its last performed action performed. When the other agent is an enemy, it observes if this agent is within shooting range. The shooting range is smaller than the sight range and depends on the unit type, and some units even attack in melee.

The state of the environment accumulates all information from agents' observations. As described, an agent perceives other agents with distances relative to itself and does not know where it is on the map. In the state, the agents' positions are encoded with their coordinates relative to the centre of the map.

Within all scenarios, the agent has the choice of eight actions: do nothing, move in one of four directions (NSWE) or attack one of its three opponents. Some actions are forbidden in SMAC, such as an attack if the opponent is not within shooting range. Therefore, agents must consider which ones are available before choosing an action.

At each time step, agents receive the same zero or positive reward. This reward is a sum of three factors: a zero or positive reward for the damage dealt, a positive reward if an enemy unit's hit points reach zero, and a positive reward if all enemy units are defeated. Maximisingg the reward forces the team toneutralisee every unit of the opposing team. Neutralising every opponent unit is commonly called a win in SMAC experiments.

We hereafter describe both scenarios of Figure 3.3. There are other types of units in different scenarios, and we refer the reader to [Samvelyan et al., 2019]. In the $3m$ scenario, represented in Figure 3.3a, six marines compete in two teams of three. A marine has 45 hit points and shoots at range, inflicting 6 damage points to an opponent for each attack. In the $3s5z$ scenario, represented in Figure 3.3b, six stalkers and ten zealots compete in two teams of eight. Both units have shield points in addition to hit points. A shield receives a different amount of damage and regenerates over time if the unit is not attacked again for a given time. A stalker has 80 hit points and 80 shield points. It shoots at range, inflicting 13 damage points to the shield, 12 damage points to a zealot's hit points and 17 to a stalker's hit points. A zealot has 100 hits points and 50 shield points. It attacks in melee and inflicts 16 damage points to the shield and 14 damage points to the hit points of a zealot or a stalker. All maps are presented in a video from the author[2].

Finally, it can be intriguing to see an environment where two teams compete is considered cooperative. This is because the built-in AI is stationary. These agents are not learning agents and can be considered part of the environment. The built-in AI strategy is a rule-based policy. Precisely, each agent moves toward the starting point of the opponent's team until it reaches the opposite side of the map and stops. If they encounter

2 https://youtu.be/VZ7zmQ_obZ0

opponents in their sight range, they select one as their target based on a priority score. They will choose to attack the closest unit with the highest priority, which will remain the target until its priority drops or until it can no longer be attacked. A unit's priority score is based on its type and current action. For example, if two of the same units attack and the targeted unit stops attacking, its priority score will drop, and the built-in AI agents will select the other unit to attack. One of the weaknesses of this built-in AI strategy that the learning agents need to learn is to stop attacking to stay alive, with the condition that an ally should be attacking. Agents must cooperate to attack and share the possible damage inflicted by the opponents. The following sections cover the MARL methods to achieve this coordination.

3.4 VALUE-BASED METHODS

As detailed in Chapter 2, value-based methods aim to learn the optimal state-action value function $Q^{\pi^*}(s, u)$, such that the optimal policy is $\pi^*(s) = \arg\max_u Q^{\pi^*}(s, u)$. In SARL, one solution, called DQN, is to approximate Q with a neural network θ and learn $Q(s, u; \theta)$ by minimising the loss defined in Equation 2.6. We hereafter describe how to train agents with value-based methods.

As introduced in Section 3.1, there exist different modes of training and execution. In the first mode, one possible centralised training and execution approach is to train a centralised learner with DQN in a Dec-POMDP. Remind that it is only possible if the Dec-POMDP is a multi-agent Markov decision process [Boutilier, 1996]. In this setting, the centralised agent learns the state-joint-action value function $Q^{\boldsymbol{\pi}}(s, \boldsymbol{u}; \theta)$ by following the adapted DQN loss

$$\mathcal{L}(\theta) = \mathbb{E}_B\left[\left(r_t + \gamma \max_{\boldsymbol{u}} Q(s_{t+1}, \boldsymbol{u}; \theta') - Q(s_t, \boldsymbol{u_t}; \theta)\right)^2\right]. \tag{3.1}$$

Issues are that the joint action space scales exponentially with n. Also, in practice, agents select their action based only on their history (o, τ) and not the state s.

For the second mode, the decentralised training and execution, each agent can learn its own Q-value independently $Q_a = Q(\tau^a, u^a)$, agnostically of the existence of other agents. IQL [Tan, 1993] is the extension of Q-Learning (see Chapter 2) to this mode. The equivalent extension has been performed with DQN [Tampuu et al., 2017], also called IQL. Typically, in a Dec-POMDP, a recurrent network approximates the independent Q-value, as explained in Section 2.5. One problem with IQL is that agents must select actions which maximise $Q(s_t, \boldsymbol{u_t})$ while ignoring, at any time, actions taken by other agents.

This is where centralised training with decentralised execution (CTDE), the third mode, comes in handy. The objective is to ensure that actions $\boldsymbol{u_t}$ maximising the individual $Q_a(\tau_t^a, u_t^a))$ functions also maximise $Q(s_t, \boldsymbol{u_t})$. With CTDE methods, this is made possible by approximating this $Q(s_t, \boldsymbol{u_t})$ as a factorisation of individual Q_a functions

during training. However, individual Q_a functions must satisfy the individual-global-max condition (IGM) [Son et al., 2019]

$$\arg\max_{\boldsymbol{u_t}} Q(s_t, \boldsymbol{u_t}) = (\arg\max_{u_t^1} Q_1(\tau_t^1, u_t^1), ..., \arg\max_{u_t^n} Q_n(\tau_t^n, u_t^n)). \tag{3.2}$$

Satisfying the IGM property allows agents to greedily select actions that maximise their individual Q_a and the state-joint-action value function $Q(s_t, \boldsymbol{u_t})$. Therefore, during training, an approximation of $Q(s_t, \boldsymbol{u_t})$, sometimes denoted Q_{tot}, is built as a factorisation of Q_a and then dropped at execution. Agents select actions based on these Q_a, satisfying IGM, and only these networks are kept for the execution. Note that these Q_a are now utility functions because they do not approximate the expected sum of discounted rewards. In this manuscript, we express Q_{tot} as a function of the state s and consider that the state is known. However, it can be rewritten as a function of the joint history $\boldsymbol{\tau}$ if the state s is unknown during training.

Maybe one of the first CTDE methods of value-based factorisation is called value decomposition network (VDN) [Sunehag et al., 2018]. This method factories Q_{tot} using the addition, an operation which satisfies IGM: $Q_{tot}^{VDN}(s_t, \boldsymbol{u_t}) = \sum_{i=1}^{n} Q_{a_i}(\tau_t^{a_i}, u_t^{a_i})$. One limitation of VDN, leaving the details of the training procedure for later, is that factorising through addition does not allow the creation of a complex approximation of the joint-action-state Q functions. Another limitation is that Q_{tot} is needed only during training, and in VDN, it does not benefit from additional information, such as the state. In the following, we present three methods for improving VDN. These three have been tested in the contributions presented in later chapters. Other methods exist, and we finish this section with a related work discussion.

3.4.1 *QMIX*

QMIX [Rashid et al., 2018] is a CTDE method where the approximation of Q_{tot} is performed by a monotonefactorisationn of the individual Q_a functions while also being a function of the state:

$$Q_{tot}^{mix}(s_t, \boldsymbol{u_t}) = \text{Mixer}\left(Q_{a_1}(\tau_t^{a_1}, u_t^{a_1}), .., Q_{a_n}(\tau_t^{a_n}, u_t^{a_n}), s_t\right). \tag{3.3}$$

The monotonicity is ensured by a hypernetwork [Ha et al., 2016] $h_p(.) : \mathcal{S} \rightarrow \mathbb{R}^{|\phi|+}$ which computes, from the state s_t, the parameters ϕ of a mixer network $h_m(.;\phi) : \mathbb{R}^n \times \phi \rightarrow \mathbb{R}$. To ensure monotonicity, the weights (and not the biases) defined by ϕ are constrained to be positive. Together, h_p and h_m defines the mixer such that $Q_{tot}^{mix}(s_t, \boldsymbol{u_t}) = h_m\left(Q_{a_1}, .., Q_{a_n}; h_p(s_t)\right)$.

The monotonicity of Q_{tot}^{mix} with respect to the individual Q_a functions,

$$\frac{\partial Q_{tot}^{mix}(s_t, \boldsymbol{u_t})}{\partial Q_a(s_t, u_t^a)} \geq 0 \ \forall a, \tag{3.4}$$

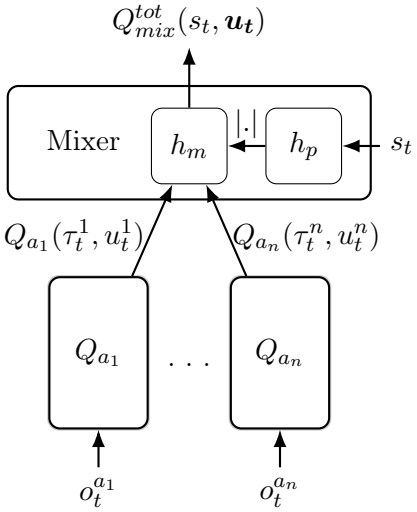


Figure 3.4: QMIX architecture.

is satisfied because a neural network comprised of monotonic functions (h_m) and strictly positive weights (h_p) is monotonic with respect to its inputs (Q_a). The entire QMIX architecture is presented in Figure 3.4.

The optimisation procedure follows the same principles of the DQN algorithm, and the loss applied to $Q_{mix}(s_t, \boldsymbol{u_t})$ is

$$\mathcal{L}(\theta) = \mathbb{E}_B \left[\left(r_t + \gamma \max_{\boldsymbol{u} \in \mathcal{U}} Q^{mix}_{tot}(s_{t+1}, \boldsymbol{u}; \theta') - Q^{mix}_{tot}(s_t, \boldsymbol{u_t}; \theta) \right)^2 \right]. \tag{3.5}$$

During training, actions are selected with an epsilon greedy policy from Q_a. At testing, actions are selected with a greedy policy. Individual Q_a networks and the mixer are all copied to produce target networks represented by θ'.

Since the Dec-POMDP induces partial observability, individual Q_a networks are commonly RNNs made of GRU [Chung et al., 2014] and the replay buffer stores sequences of contiguous transitions instead of isolated transitions $(s_t, \boldsymbol{u_t}, r_t, s_{t+1})$. A typical architecture of such a network is presented in Figure 3.5. When evaluating CTDE methods, IQL is commonly tested by training the same architecture as the individual network, allowing for comparison with a fully decentralised training of such a network.

3.4.2 *MAVEN*

Mahajan et al. [2019] defined the class of state-joint-action value functions that QMIX cannot represent due to its exploration strategy and the monotonicity constraint. They demonstrated the existence of payoff matrices in an n-player normal-form game with more than three actions per agent, for which QMIX learns a suboptimal policy for any

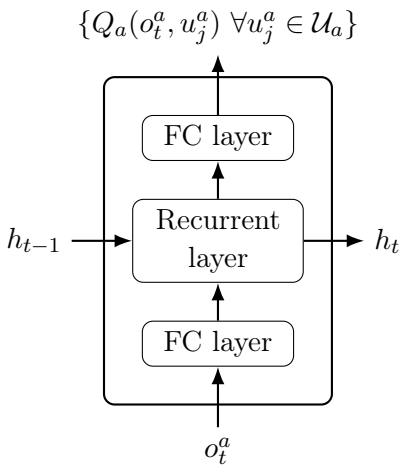


Figure 3.5: Common Q_a network implementation. The hidden state h embeds the history, and the action space size defines the number of outputs of the network.

training duration, using both epsilon greedy and uniform exploration. To tackle this problem, they modified the individual Q_a architecture, which is now a function of a latent space to influence agent behaviour. The underlying objective is to train agents to learn an ensemble of policies to improve their exploration capabilities.

Specifically, the latent variable is the input of a second and new hypernetwork that computes parameters of the fully connected layer linking recurrent cells to the outputs of the individual Q_a networks. This latent variable z is generated per episode by a hierarchical policy network, taking as input the initial state of the environment and a random variable, typically a discrete uniform. The idea is that the latent variable corresponds to different learnt strategies. The goal of the hierarchical policy network is to select the best strategy based on the initial state s_0, which is considered known at testing. The new architecture of the individual Q_a network of MAVEN is represented in Figure 3.6.

MAVEN's network objective function comprises three parts. Some parameters must be fixed when computing some parts, meaning all parameters are not updated based on all objectives. The first part of the objective is the loss of QMIX defined in Equation 3.5, and it optimises both hypernetworks and individual networks. This loss is computed by fixing the hierarchical policy network and, thus, the latent variable z.

The hierarchical policy network can be optimised with any policy-based method, such as policy gradient maximising the sum of rewards per episode, defined in Section 2.4.5. This second objective is computed by fixing both hypernetworks and individual networks.

The third part of the objective ensures that different values of z imply different behaviours. It is a mutual information loss between the latent variable and the trajectories performed by agents to favour different behaviours for different latent variable values. For

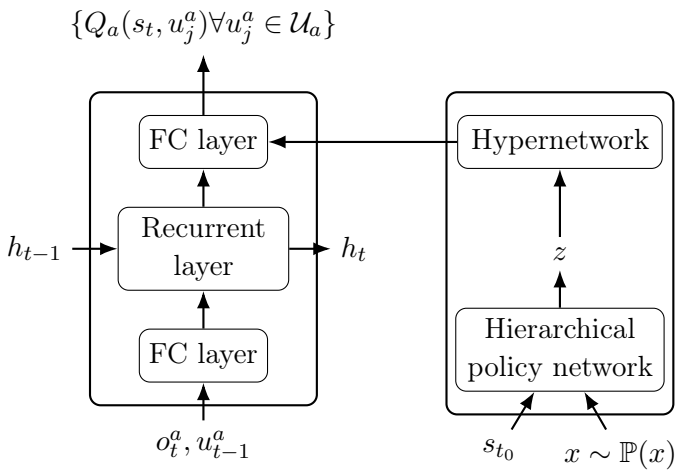


Figure 3.6: MAVEN modification of the individual Q_a network.

further details on the MAVEN optimisation procedure and especially on constructing the mutual information objective, we refer the reader to Mahajan et al. [2019].

3.4.3 *QPLEX*

QPLEX [Wang et al., 2021] extends QMIX with the dueling structure $Q(s_t, u_t) = V(s_t) + A(s_t, u_t)$ [Wang et al., 2016], learning a factorisation of V and A with transformers [Vaswani et al., 2017]. The advantage function A has been introduced in Section 2.4.5. The duelling structure involves learning $V(s_t)$ and $A(s_t, u_t)$, typically with a single neural network with a common backbone and two different heads. By separating the computation of the value of a given state and the contributions of different actions in that given state, Wang et al. [2016] demonstrated better results than DQN.

Back to QPLEX, Wang et al. [2021] demonstrated that if the advantage function A respects the advantage-IGM, then the state action value function $Q = V + A$ respects IGM while removing constraints on the state value function V. The individual Q_a satisfy the advantage-IGM if $Q_{tot} = V_{tot} + A_{tot}$ and $V_{tot}(s) = \max_{\boldsymbol{u}} Q_{tot}(s, \boldsymbol{u})$ and $Q_a = V_a + A_a$ and $V_a(s) = \max_u Q_a(s, u)$ such that

$$\arg\max_{\boldsymbol{u_t}} A_{tot}(s_t, \boldsymbol{u_t}) = (\arg\max_{u_t^1} A_1(\tau_t^1, u_t^1), ..., \arg\max_{u_t^n} A_n(\tau_t^n, u_t^n)) \tag{3.6}$$

holds. This defines the duplex dueling structure of QPLEX. The constraint on the Q_a is transferred to the advantage functions A_i, leading to an unconstrained approximation of the state value function V.

3.4.4 *Other value-based methods*

In addition to QMIX [Rashid et al., 2018], QVMix [Leroy et al., 2021] and QPLEX [Wang et al., 2021], Qatten [Yang et al., 2020] is another example of Q_{tot} factorisation with transformers. Other, such as QTRAN [Son et al., 2019] and Weighted-QMIX [Rashid et al., 2020] factorise Q_{tot} differently from the QMIX and VDN approach to improve the representational capacity of the factorisationn. However, they end up not always satisfying IGM. Other methods learn to cooperate without factorising Q_{tot} to satisfy IGM. An example is local advantage network (LAN) [Avalos et al., 2023], which learns a central state value function of the joint policy $V^{\boldsymbol{\pi}}$ and individual advantages A^a to learn Q_a during training, keeping only A_a at execution. There are many other methods relying on the value function decomposition. Finally, Hong et al. [2022] claims that many have been trying to improve the Q_{tot} factorisation to satisfy IGM, but only a few examine IGM defects. Lossy decomposition occurs when the actions that maximise Q_{tot} are not the ones maximising Q_a. They demonstrate the existence of lossy decomposition when the observation of one agent is characterised as insufficient. An observation is insufficient when it does not change when the state changes. This claim is somehow similar to the one of Mahajan et al. [2019], who demonstrated the existence of games where QMIX cannot correctly approximate Q_{tot}, focusing on the reward structure. This highlights some remaining challenges in CTDE value-based methods.

3.5 POLICY-BASED METHODS

Agents trained with policy-based methods learn the policy with a neural network, as introduced in Chapter 2, and we hereafter describe actor-critic methods adapted to the Dec-POMDP. As demonstrated with DQN in the previous section, in the centralised mode, it is possible to train an agent with a SARL solution to select actions in the joint action space. In the decentralised mode, the equivalent to IQL for policy-based methods is named IAC [Foerster et al., 2018b]. The policies learned by the actor of each agent is $\pi^a(u_t^a|\tau_t^a, o_t^a; \theta)$. The policy parameters are updated by ascending the gradient

$$\nabla_\theta J(\pi_\theta^a) = \mathbb{E}_B \left[A_a(\tau_t^a, o_t^a, u_t^a; \phi) \nabla_\theta \log \pi^a(u_t^a|\tau_t^a, o_t^a; \theta) \right]. \tag{3.7}$$

As in SARL, there are two possible ways of estimating the advantage with a critic network. This leads to two IAC versions: IAC-V where a state value function is learned $A_a(\tau_t, u_t; \phi) = r + \gamma V_a(\tau_{t+1}; \phi) - V_a(\tau_t; \phi)$ and IAC-Q, where a state-action value function is learned $A_a(\tau_t, u_t; \phi) = Q_a(\tau_t, u_t; \phi) - \sum_{u^a} \pi_\theta(\tau_t, u^a) Q_a(\tau_t, u^a; \phi)$. The parameters of π^a and Q_a are denoted θ or ϕ and not θ_a or ϕ_a to improve readability, although they might be shared or not by agents.

In IAC, each agent independently learns an actor and a critic, but this solution does not benefit from any additional information, such as the state s. Since critics are used only during training, one straightforward solution is to exploit the state s to compute

a centralised critic. Moreover, the critic is trained to update the policy network and plays a crucial role in each agent's credit assignment. We hereafter detail two methods that consider a single centralised critic exploiting the state tested in the contributions presented in later chapters. Other methods exist, and we finish this section with a related work discussion.

3.5.1 *COMA*

COMA stands for counterfactual multi-agent policy gradient and is a solution proposed by Foerster et al. [2018b]. They motivate their approach with the constatation that a centralised critic computing the advantage $A_a(s_t, u_t^a) = r_t + \gamma V(s_{t+1}; \phi) - V(s_t; \phi)$ is based on the common reward r_t but does not consider how the action of an individual agent influences it. Such a centralised critic does not solve the credit assignment problem.

Foerster et al. [2018b] propose to use a counterfactual baseline inspired by difference rewards $D_t^a = R(s_{t+1}, s_t, \boldsymbol{u_t}) - R(s_{t+1}, s_t, (\boldsymbol{u_t^{-a}}, c_t^a))$ [Wolpert and Tumer, 2001]. The common reward $R(s_{t+1}, s_t, \boldsymbol{u_t})$ is compared to a reward obtained when agent a executes a default action c^a, while we preserve actions of other agents $\boldsymbol{u_t^{-a}}$ unchanged. Any action u^a that maximises $R(s_{t+1}, s_t, \boldsymbol{u_t})$ also maximises D_t^a. However, there are immediate limitations. First, the simulator needs to be run to obtain this "default reward" $R(s_{t+1}, s_t, (\boldsymbol{u_t^{-a}}, c_t^a))$. And this must be done for all agents. Second, if we consider it possible to approximate it, this would induce additional approximation error. Third, the choice of the default action is not trivial.

The COMA solution to these limitations is to build a centralised critic that computes difference rewards by learning $Q(s, \boldsymbol{u})$. For each agent a, the advantage updating actor networks in Equation 3.7 is

$$A_a(s_t, \boldsymbol{u_t}; \phi) = Q(s_t, \boldsymbol{u_t}; \phi) - \sum_{u'^a} \pi^a(u'^a | \tau_t^a, o_t^a; \theta) Q(s_t, (\boldsymbol{u_t^{-a}}, u'^a); \phi). \tag{3.8}$$

This solves the problems of simulating the default rewards and choosing default actions. However, $|\mathcal{U}^1| + ... + |\mathcal{U}^n|$ values must be computed to obtain the advantage of all agents. In practice, a Q network has one output for each possible action. Here, this leads to $\sum_{i=1}^{n} |\mathcal{U}^i|$ outputs which would become impractical whether n or $|\mathcal{U}^a|$ increases. To alleviate this, the critic can take as input $\boldsymbol{u^{-a}}$ and only outputs $|\mathcal{U}^a|$ outputs. To use the same critic for all agents implies that all action spaces are the same size.

A significant limitation is that this method is only possible for discrete action spaces, especially considering the sum over all actions in the advantage. Foerster et al. [2018b] argue that it would be possible to apply COMA with continuous action spaces easily by evaluating $\sum_{u'^a} \pi^a(u'^a | \tau^a) Q(s, (\boldsymbol{u^{-a}}, u'^a))$ with Monte Carlo estimations or by using policy construction that allows to compute it anatically. A final observation is that the critic is considered central because using the state s_t, but it should be called n times to update the n actor networks.

In COMA, policy networks θ usually follow the same architecture of IQL presented in Figure 3.5, where the outputs are not Q values for each action but the probability of taking each action, usually obtained by applying a softmax to the last layer. The critic is commonly composed of fully connected layers that output the $|\mathcal{U}^a|$ from the state s_t, the actions of others $\boldsymbol{u_t^{-a}}$ and sometimes additional information, such as o_t^a or the previous taken action $\boldsymbol{u_{t-1}}$.

3.5.2 *FACMAC*

FACMAC [Peng et al., 2021] stands for factored multi-agent centralised policy gradients. They propose to use a centralised but factored critic computing Q_{tot} as a factorisation of individual Q_a, like in QMIX. In COMA, the critic is centralised because it has access to the state, but all agents use it independently to update their decentralised policy. Peng et al. [2021] calls this approach a monolithic critic, and they argue that a non-monolithic factored critic is preferable to scale to many agents or actions. In addition, they show that estimating gradients independently for each agent can yield sub-optimal joint policies. This motivates the centralised and factored critic which computes a single gradient estimate for the joint policy, improving cooperation capabilities.

Conversely to COMA, FACMAC is designed for continuous action spaces and can be adapted to discrete ones. It is built on the same foundation of another actor-critic method for MARL developed to solve a stochastic game called multi-agent deep deterministic policy gradient (MADDPG) [Lowe et al., 2017]. The latter is a multi-agent version of DDPG [Lillicrap et al., 2015], an actor-critic method using deep neural networks to approximate a deterministic policy and a Q function in SARL. Since the action space is continuous, obtaining the action $\arg\max_u Q(s,u)$ can be difficult. Instead of computing the arg max, the solution in DDPG is to learn a parametrised deterministic policy $\mu_\theta : \mathcal{S} \rightarrow \mathcal{U}$ that solves $\max_\theta \mathbb{E}_B[Q(s, \mu_\theta(s); \phi)]$ by gradient ascent, while the critic estimates this Q [Silver et al., 2014]. MADDPG uses DDPG to learn decentralised policies using centralised critics with access to the state and actions of other agents. In MADDPG, each decentralised actor $\mu^a(\tau^a, o^a; \theta)$ is updated by ascending the gradient

$$\nabla_\theta J(\mu^a) = \mathbb{E}_B \left[\nabla_\theta \mu^a(\tau_t^a, o_t^a; \theta) \nabla_{u^a} Q_a^{\boldsymbol{\mu}}(s_t, \boldsymbol{u_t^{-a}}, ua; \phi) | u^a = \mu^a(\tau_t^a, o_t^a) \right], \tag{3.9}$$

where actions of other agents are taken from the replay buffer while the action of agent a is provided by its deterministic policy when computing the state-joint-action value function. The critic is updated following the same centralised loss adapted from DQN defined in Equation 3.1 where, in addition to a target network to compute Q, actions of agents to compute this target value are taken from target deterministic policies. While MADDPG addresses stochastic games with individual rewards, it can be applied easily in Dec-POMDP, with the drawbacks mentioned earlier. Note that the critic of MADDPG is monolithic and different for each agent, like COMA.

Using a single centralised and factored critic is FACMAC's change to MADDPG. All agents share this critic that computes $Q_{tot}^{\boldsymbol{\mu}}$ where $\boldsymbol{\mu}$ is the joint deterministic policy. Like the CTDE value-based methods, Q_{tot} is a function of individual utility functions Q_a. The factorisation can be done by a sum, such as in VDN, or by a hypernetwork, such as in QMIX in Equation 3.3. However, the monotonicity constraint to satisfy IGM is not required for a critic.

The factored critic does not replace how Q is computed in Equation 3.9. Indeed, in MADDPG, actors are updated based on $\boldsymbol{u_t^{-a}}$ stored in the replay buffer, which can lead to sub-optimal joint policies. Instead, FACMAC computes a single gradient based on the current joint policy $\boldsymbol{\mu}(\boldsymbol{\tau}, \boldsymbol{o}) = (\mu^{a_1}(\tau^{a_1}, o^{a_1}), .., \mu^{a_n}(\tau^{a_n}, o^{a_n}))$ given by

$$\nabla_\theta J(\mu^a) = \mathbb{E}_B \left[\nabla_\theta \boldsymbol{\mu} \nabla_{\boldsymbol{\mu}} Q_{tot}^{\boldsymbol{\mu}}(s_t, \boldsymbol{\tau_t}, \boldsymbol{\mu}(\boldsymbol{\tau_t}, \boldsymbol{o_t}); \phi) \right]. \tag{3.10}$$

With discrete action space, using a Straight-Through GumbelSoftmax [Jang et al., 2017] allows for a discrete but differentiable policy.

Finally, updating the joint policy instead of independently updating the policies is claimed to be required to optimally benefit from the centralised critic by Peng et al. [2021]. They confront the work by Lyu et al. [2021], who showcase the suboptimality of the independent updates of actor networks despite using a monolithic centralised critic.

3.5.3 *Other policy-based methods*

Multi-agent deep deterministic policy gradient (MADDPG) [Lowe et al., 2017] is a well-established method which does not learn a single centralised critic but one per agent. It is designed for continuous action spaces, has been exploited in stochastic games, and can easily be adapted to Dec-POMDP. Another method, LIIR [Du et al., 2019], aims to provide credit assignment by computing individual intrinsic rewards. Following the success of TRPO [Schulman et al., 2015] and PPO [Schulman et al., 2017] in SARL, IPPO [De Witt et al., 2020], MAPPO [Yu et al., 2022] alongside HATPRO and HAPPO [Kuba et al., 2021] demonstrate that the popular actor-critic methods from SARL can be extended to cooperative MARL tasks.

3.6 OTHER APPROACHES

This chapter defines several methods for the three modes of training and execution in a Dec-POMDP. While we defer the performance comparison to a later chapter, we mentioned some of their limitations and other existing methods. Hereafter, we discuss completely different approaches beyond the definition of a Dec-POMDP that tackle the MARL problem differently, in the cooperative setting or not.

Mean-field game is an additional way of modelling the multi-agent framework [Laurière et al., 2022]. By construction, mean-field games can consider an infinite number of agents because the reward is computed by considering the mean of the distribution of agents'

strategies or states rather than individual agents. This approach significantly reduces the computational complexity compared to traditional methods, making it suitable for analysing systems with a large population of interacting agents. It is an exciting approach to replace the CTDE methods considered in this manuscript that have difficulty scaling up with the number of agents, as shown later in Chapter5.

Another approach for dealing with cooperative multi-agent settings is to link the recent success of sequence models in language and RL by using a multi-agent transformer (MAT) that learns to transform a sequence of observations into a sequence of actions, one per agent [Wen et al., 2022]. Such approaches have become increasingly popular with the rise of foundation and large language models.

Finally, we do not address communication in the environment. Communication is also an additional topic mentioned by [Oliehoek and Amato, 2016]. Allowing agents to send messages can be modelled in different ways. Foerster et al. [2016] propose that messages are the parallel of actions. An agent decides on a message and an action based on observations and messages sent by other agents. Messages have no impact on the state transition or the reward. This may be one of the first works addressing communication in MARL, specifically in the cooperative setting. In its master thesis, Fombellida-Lopez [2020] studied how communication can affect performance in SMAC and also presented a survey of communication methods. The challenges include adapting to various numbers of agents, targeted communication, and limiting the number of messages.

4

THE DEEP QUALITY-VALUE FAMILY IN DEC-POMDP

Outline

This chapter presents four value-based methods issued from the Deep Quality-Value (DQV) family for the Dec-POMDP framework. We introduce the DQV family and detail the contribution of this chapter in Section 4.1. We then formally define DQV and the four methods in Section 4.2. The experimental setup to evaluate them is presented in Section 4.3, followed by the corresponding results in Section 4.4. This chapter ends with a conclusion in Section 4.5.
This chapter is an adapted version of the publication [Leroy et al., 2021] *QVMix and QVMix-Max: extending the deep quality-value family of algorithms to cooperative multi-agent reinforcement learning*, P. Leroy, D. Ernst, P. Geurts, G. Louppe, J. Pisane, and M. Sabatelli. AAAI-21 Workshop on Reinforcement Learning in Games, 2021.

4.1 INTRODUCTION

This chapter introduces four methods to learn to cooperate in a Dec-POMDP, defined in Section 3.2. They are adapted from the Deep Quality-Value (DQV) family of algorithm [Sabatelli et al., 2020]. DQV methods jointly learn an approximation of the state-value function V alongside an approximation of the state-action value function Q. They have proven to outperform popular algorithms in SARL, such as DQN and DDQN defined in Section 2.4.4. The four methods follow two modes of training and execution in a Dec-POMDP, introduced in Section 3.1. The methods designed for the decentralised mode are called IQV and IQV-Max. The other two dedicated to the CTDE mode are called QVMix and QVMix-Max. They are tested with the StarCraft Multi-Agent Challenge (SMAC) suite of environments, introduced in Section 3.3. They are compared with IQL, QMIX and MAVEN, three value-based methods defined in Section 3.4.

The contributions of [Leroy et al., 2021] presented in this chapter can be divided into three parts. The first is the generalisation of the DQV family of algorithms to cooperative MARL problems, their performance in the decentralised mode and their fundamental limitations. The second is the introduction of two methods, QVMix and QVMix-Max. Both combine the original benefits of the DQV algorithms with CTDE, resulting in a better performance than state-of-the-art techniques of their time. The third is to link

their better performance to the overestimation bias of the Q function that characterises model-free RL algorithms.

4.2 METHODS

As introduced, the work presented in this chapter revolves around the Deep Quality-Value (DQV) family of DRL algorithms [Sabatelli et al., 2018, 2020]. These SARL techniques learn an approximation of the state value function V alongside an approximation of the state-action value function Q. The DQV algorithms extend the tabular RL algorithms called QV(λ)-Learning [Wiering, 2005; Wiering and Van Hasselt, 2009] with neural networks as function approximations. One motivation of Wiering [2005] is that learning the state value function may converge faster than the state-action one. There are two possible ways of learning a joint approximation of the V function, $V(s;\phi) \approx V^{\pi^*}(s)$, and of the Q function, $Q(s,u;\theta) \approx Q^{\pi^*}(s,u)$.

DQV learns an approximation of the V function by minimising

$$\mathcal{L}(\phi) = \mathbb{E}_B\Big[\big(r_t + \gamma V(s_{t+1};\phi') - V(s_t;\phi)\big)^2\Big], \tag{4.1}$$

while the Q function is learned by minimising

$$\mathcal{L}(\theta) = \mathbb{E}_B\Big[\big(r_t + \gamma V(s_{t+1};\phi') - Q(s_t,u_t;\theta)\big)^2\Big]. \tag{4.2}$$

DQV-Max learns the Q function with the same loss defined in Equation (4.2) but learns the V function by minimising

$$\mathcal{L}(\phi) = \mathbb{E}_B\Big[\big(r_t + \gamma \max_{u\in\mathcal{U}} Q(s_{t+1},u;\theta') - V(s_t;\phi)\big)^2\Big]. \tag{4.3}$$

As in DQN, the replay buffer B stores transitions (s_t, u_t, r_t, s_{t+1}) from which batches of transitions are sampled to update the networks. The target network θ' and ϕ' are a copies of θ and ϕ updated periodically. This manuscript does not compare these SARL algorithms with DQN or DDQN, but we refer the reader to the PhD thesis of Sabatelli [2022]. This thesis provides further analysis of the quality of the approximations, showcasing DQN's overestimation bias while highlighting that DQV and DQV-MAX do not suffer from it.

The extension to the decentralised MARL methods is straightforward. Like IQL is the decentralised DQN, we define IQV and IQV-Max as the decentralised DQV and DQV-Max, using, respectively, the DQV and DQV-Max update rules to learn $Q_a(\tau^a, u^a;\theta)$ and $V_a(\tau^a;\phi)$ independently. For the CTDE mode, the methods are QVMix and QVMix-Max, and they follow the DQV and DQV-Max update rules to learn $Q_{tot}^{mix}(s,\boldsymbol{u})$. The architecture of the Q_a and the mixer network is the same as in QMIX, using the same monotonic decomposition. In both QVMix and QVMix-Max, the V network computes a

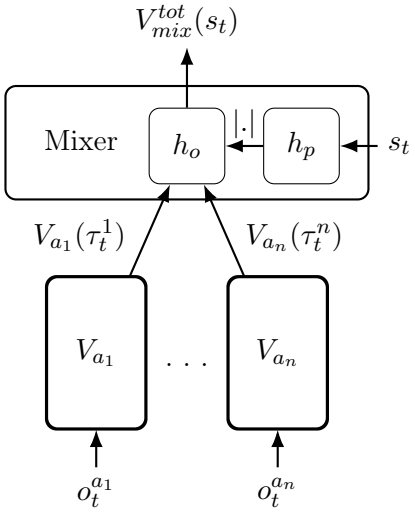


Figure 4.1: Architecture of the V_{mix} network.

central state value function $V_{tot}^{mix}(s)$ as a function of individual $V_a(\tau^a;\theta)$. It has the same architecture as the Q ones of QMIX, except they have a single output. The architecture is presented in Figure 4.1. In QVMix, V_{tot}^{mix} is updated following the loss defined in Equation 4.1 and with the loss defined in Equation 4.3 for QVMix-Max. Individual networks are GRU, and B stores sequences of contiguous transitions instead of single transitions to train recurrent neural networks. The following section described how these methods are evaluated in the SMAC suite of environments.

4.3 EXPERIMENTS

Our experiments evaluate seven methods in total. The four defined in this chapter, QVMix, QVMix-Max, IQV and IQV-Max and the three forming the state of the art at that time, QMIX, MAVEN and IQL, defined in Chapter 3. As a test-bed, we use SMAC and evaluate the methods on eight different maps: `3m`, `8m`, `so_many_baneling`, `2m_vs_1z`, `MMM`, `2s3z`, `3s5z` and `3s_vs_3z`. We refer the reader to Section 3.3.1 for the details on SMAC. It is worth noting that the maps chosen for our experiments differ in complexity. In SMAC, the goal is to maximise the sum of discounted rewards achieved by reducing each opponent team unit's health to zero, which is called a win.

Each method is executed on every map ten times, and neural networks are trained from scratch each time. Networks are trained for $5m$ time steps for the first four maps mentioned above and $10m$ time steps for the four others, chosen because of the time required to achieve convergence. Every 20.000 time step, the parameters of the networks are saved to perform 24 testing episodes.

Map	QMIX	MAVEN	QVMix	QVMix-Max	IQL	IQV	IQV-Max
3m	100	98.7	**100**	100	93.3	93.3	96.6
8m	96.6	98.3	**100**	96.6	83.3	93.3	90
so_many_baneling	100	97	**100**	100	50	40	40
2m_vs_1z	100	100	**100**	96.6	100	100	100
MMM	**100**	97.0	93.3	96.6	61.6	83.3	50
2s3z	96.6	97.5	96.6	**100**	59.9	56.6	40
3s5z	40	40.8	**86.6**	43.3	16.6	13.3	0
3s_vs_3z	100	97.9	**100**	100	83.3	76.6	63.3

Table 4.1: Means of win rates achieved in eight scenarios at the end of training by QMIX, MAVEN, QVMix, QVMix-Max, IQL, IQV, and IQVMax. In the first four scenarios, 3m, 8m, so_many_baneling and 2m_vs_1z, it is measured after 5 millions training time steps. In the last four, MMM, 2s3z, 3s5z and 3s_vs_3z it is measured after 10 millions training time steps. We report the best and second-best means by **bolding** and underlining them. When results are equivalent, the cells report the fastest and second-fastest method that reaches a win rate of 100% as shown in Figure 4.3.

Each algorithm uses the same hyperparameter values to compare the tested method fairly. Specifically, we refer to the authors of QMIX, MAVEN, and IQL to determine the hyper-parameters set and keep the same values for QVMix, QVMix-Max, IQV, and IQV-Max. For a more thorough presentation of all used hyper-parameters, we refer the reader to the open-sourced code[1]. As is common practice within the literature, the individual networks' parameters are shared among agents to improve the algorithms' learning speed. This means a single individual Q_a network is used for all agents. A one-hot encoding of the agent id ($\{1, .., n\}$) is added to the observation space to allow individual networks to produce different policies per agent.

4.4 RESULTS

We report the results of experiments in two different ways. We start by analysing each tested method's win rate before investigating the quality of the value functions learned by all algorithms.

The means of win rate for each map and algorithm are reported in Table 4.1. Suppose the algorithms perform equally in terms of overall performance, meaning the average win rate is the same. In that case, we consider the one which significantly converges the fastest to be the best-performing algorithm. Please note that reporting an episode's win

1 https://github.com/PaLeroy/QVMix

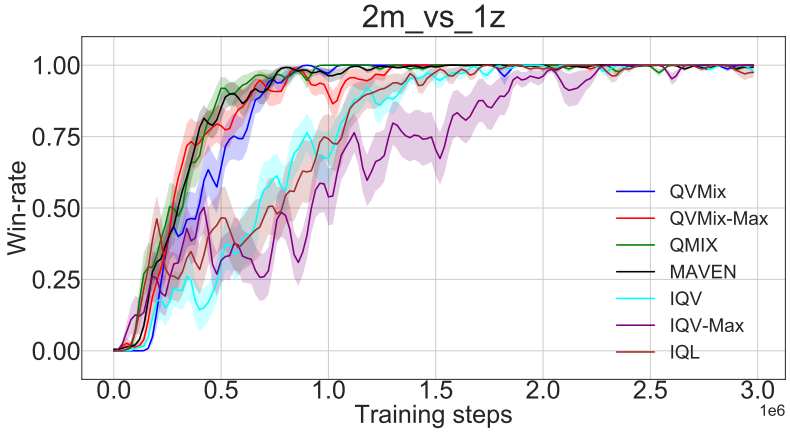


Figure 4.2: Mean of win rates achieved in the `2m_vs_1z` map by QVMix, QVMix-Max, QMIX, MAVEN, IQV, IQVMax and IQL. The error band is proportional to the variance of the measure. We observe that all CTDE methods result in faster training than decentralised ones. All four novel algorithms based on the DQV algorithms can be successfully used in cooperative MARL.

rate is a good indicator of the quality of an agent's learned policy since, as introduced in the previous section, a win directly corresponds to the best achievable sum of rewards an agent can receive.

We start by observing the differences in performance between the decentralised mode methods IQL, IQV, and IQV-Max and their respective CTDE extensions QMIX, QVMix, QVMix-Max, and MAVEN. As one might expect, we can see from the results reported in Figure 4.2 that methods for the decentralised mode converge slowly when compared to their CTDE counterparts on the considered `2_vs_1z` map. This is particularly interesting since it shows that the DQV family of algorithms can be successfully adapted to MARL, both in the decentralised mode and the CTDE one. However, these results are challenged once the number of agents in the maps increases: examples of such maps are `so_many_baneling`, `MMM` or `3s5z`. The performance of decentralised methods starts to drop, highlighting that CTDE methods learn faster once the complexity of the training scenario increases, as is reported both in Table 4.1 and Figure 4.3, where the evolution of the win rate of each algorithm on every tested map is presented. These win rate curves provide the variance between training executions. It is essential to highlight that all methods suffer from a significant variance. However, it is difficult to conclude which one suffers the most.

Therefore, directing attention to CTDE methods, we observe that only QVMix and QVMix-Max perform as well as QMIX and MAVEN in most of the eight maps. When we consider the `MMM`, `3m`, `2m_vs_1z` and the `so_many_baneling` maps, we observe that there is no significant difference between the performance that is obtained by our algorithms and that of QMIX and MAVEN. All methods converge towards the best possible winning rate and, in terms of convergence speed, perform closely.

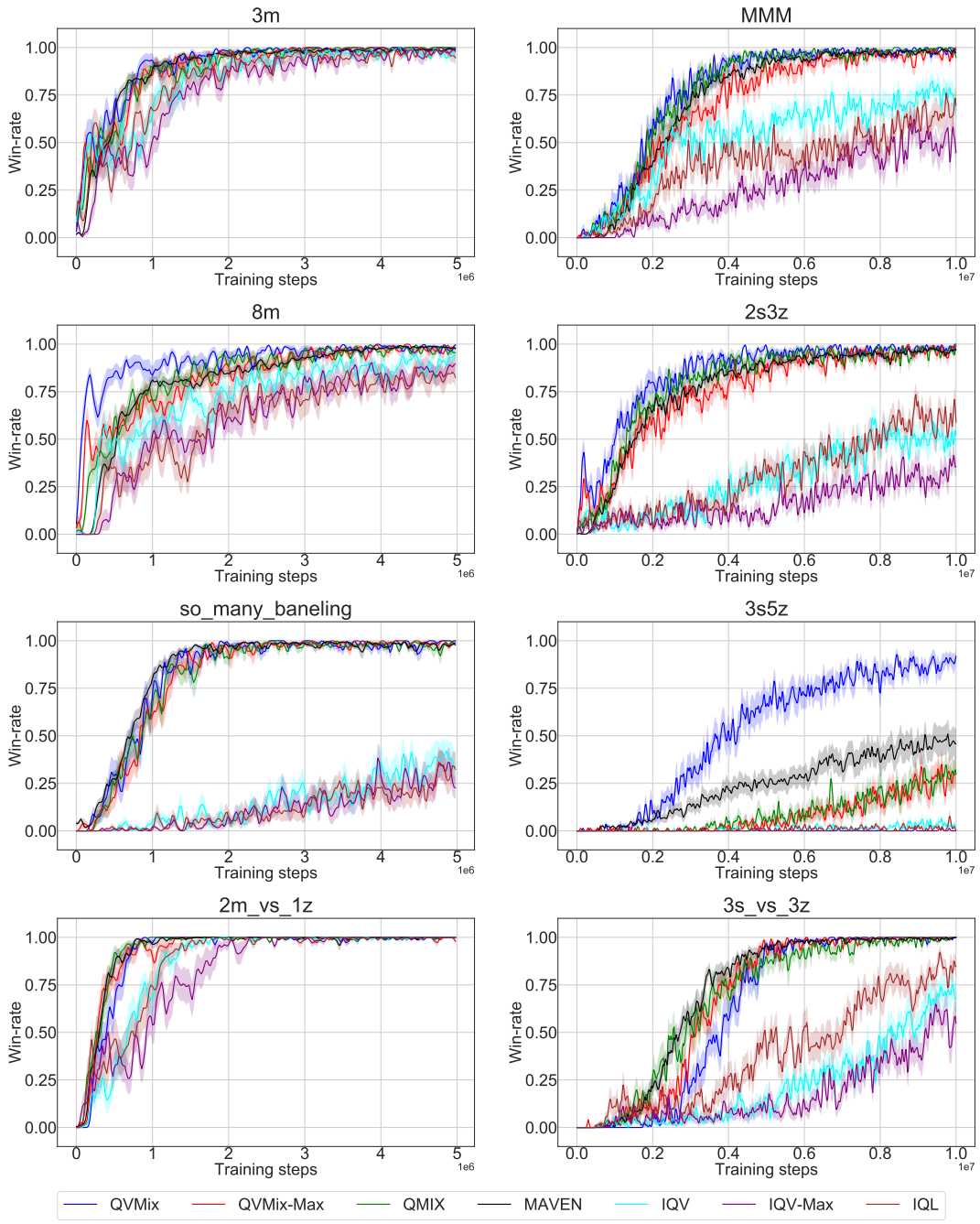


Figure 4.3: Means of win rates achieved by QVMix, QVMix-Max, QMIX, MAVEN, IQV, IQVMax and IQL in eight scenarios. Top to bottom, left to right, the scenarios are `3m`, `8m`, `so_many_baneling`, `2m_vs_1z`, `MMM`, `2s3z`, `3s5z` and `3s_vs_3z`. The error band is proportional to the variance of win rates.

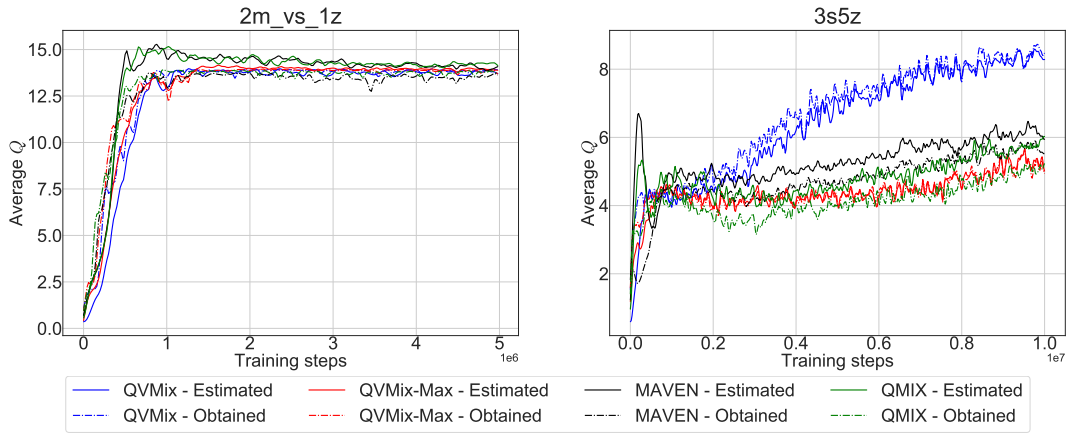


Figure 4.4: Q values obtained and estimated when training QVMix, QVMix-Max, MAVEN and QMIX. Dash-dotted lines represent the obtained Q values, while solid lines represent the estimated ones.

However, when considering the `2s3z`, `3s5z` and `8m` maps, we observe that the performance of QVMix results in even faster learning. Of greater interest, when looking at the results obtained on the `3s5z` map, QVMix is the only algorithm approaching the best possible win rate. It is also worth noting that QVMix-Max's performance is always competitive with that of QVMix, QMIX, and MAVEN. These results are unsurprising since a similar performance was observed when DQV-Max was tested in a SARL setup by Sabatelli et al. [2020].

To understand the reasons behind why QVMix is the best performing algorithm overall, we analyse how well each method estimates the state-joint-action value function $Q^{tot}_{mix}(s_t, \boldsymbol{u_t})$. Since, in most maps, decentralised methods do not perform as well as CTDE methods, we restrict our analysis to CTDE algorithms only where their respective mixer networks give the estimated $Q(s_t, \boldsymbol{u_t})$. Since the state space of the maps provided by the SMAC environment is not finite, it is not impossible to compute the exact $Q_{tot}(s_t, \boldsymbol{u_t})$ for all states to compare them with the estimations. To overcome this problem, we compute the discounted sum of rewards obtained with the current policy in each visited state during an episode and compare the results with the value function inferred from the Q_{tot} values estimated by the mixer network for these states. The closer the estimates are to the actual $Q(s_t, \arg\max_{\boldsymbol{u}}(Q(s_t, \boldsymbol{u})))$, the more accurate the learned value function is.

For this experiment, we selected two different maps: the `2m_vs_1z` map, which corresponds to the map on which the best results have been achieved by all methods at the same time, and the `3s5z` map, which on the other hand, is the map on which QVMix, and others, performed less well. In Figure 4.4, we report the averaged estimated Q values, represented by the solid lines, and the actual discounted sum of rewards, represented by the dash-dotted lines. All are computed for each visited state at testing time. In both scenarios, we can observe that the Q values estimated by QMIX and MAVEN suffer

from the overestimation bias of the Q function, while this is not the case for QVMix and QVMix-Max. Therefore, we justify the better quality of QVMix and QVMix-Max policies by better approximating the Q functions. However, further work is required to understand this phenomenon in more detail.

4.5 DISCUSSION AND FUTURE WORK

In this chapter, we introduced four new value-based methods for training a team of agents in a Dec-POMDP. Two of our methods, IQV and IQVMax, are designed for the decentralised mode, while the two dedicated to the CTDE mode are QVMix and QVMix-Max. We compared these algorithms with three methods from the literature, using the StarCraft Multi-Agent Challenge as a benchmark. We have shown that QVMix and QVMix-Max achieve the same results as popular techniques QMIX and MAVEN and that QVMix can result in faster and better learning in some maps. We suggest that this better performance can be related to the fact that QVMix seems to suffer less from the overestimation bias of the Q function.

In future work, it would be interesting to analyse each agent's behaviour and study the impact of the value function in the optimisation procedure. Furthermore, following the recommendations from [Gorsane et al., 2022], testing in different environment suites is crucial. The next chapter also tests QVMix, which provides performance similar to QMIX, as with SMAC. In this following study, QVMix is also compared to policy-based methods, which are missing in the analysis of this chapter and are justified by the performance of existing ones at the time of development. The number of agents is also limited in SMAC. The next chapter covers setting up to 100 agents, which allows testing the scalability of methods.

In Chapter 3, we defined the dueling structure, which is sometimes considered close to the DQV family because both approximate the state value function V to obtain the Q. We hereafter explain how different they are. In DQV methods, the V network approximates the expected return of the optimal policy $\approx V^{\pi^*}(s)$ to provide a target to train a Q network. Conversely, in the dueling structure $Q = V + A$, the V network does not play the target role in the loss but allows Q to be decomposed. Nevertheless, QPLEX and QVMix performance are compared in the next chapter. Finally, LAN proposes to learn a single central V and independent advantage A to compute independent Q_a using the dueling structure. It would be interesting to study how to relax the constraint of the Q mixer in QVMix and QVMix-Max to train only independent Q_a, using a single central V approximation as a target.

5

INFRASTRUCTURE MANAGEMENT PLANNING

Outline

This chapter presents IMP-MARL, an open-source suite of multi-agent reinforcement learning environments for large-scale infrastructure management planning. In section 5.1, we introduce the problem of infrastructure management planning (IMP) and the motivations of IMP-MARL. RL has not been the first solution to such problems, and we present related works in Section 5.2. We then define IMP as a cooperative MARL problem in Section 5.3, describing the different components of the Dec-POMDP and a high-level description of the environment. Following this, we provide the formal definition of the models of environments in Section 5.4. The experimental setup to demonstrate the interest of MARL for IMP is presented in Section 5.5, followed by the corresponding results in Section 5.6. Conclusions and discussions end the chapter in Section 5.7.
This chapter is an adapted version of the publication [Leroy et al., 2023] *IMP-MARL: a suite of environments for large-scale infrastructure management planning via MARL*, P. Leroy, P. G. Morato, J. Pisane, A. Kolios, and D. Ernst. Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2023.

5.1 INTRODUCTION

Multiple suites of environments based on games and simulators have served as benchmark testbeds to support the advancement of cooperative MARL methods and are presented in Section 3.3. Benchmarking environments based on games and simulators helps develop MARL methods in specific collaborative/competitive tasks. However, additional challenges may still be encountered when deploying MARL methods in real-world applications [Oroojlooy and Hajinezhad, 2023]. This work follows this direction to promote the interest of MARL to help solve real-world problems.

Infrastructure Management Planning (IMP) is a contemporary application that responds to current societal and environmental concerns. In IMP, inspections, repairs and/or retrofits should be timely planned to control the risk of potential system failures, e.g., bridge and wind turbine failures, among many others [Morato et al., 2022]. System failure risk is defined as the system failure probability multiplied by the consequences associated with a failure event, typically in monetary units. Due to model and

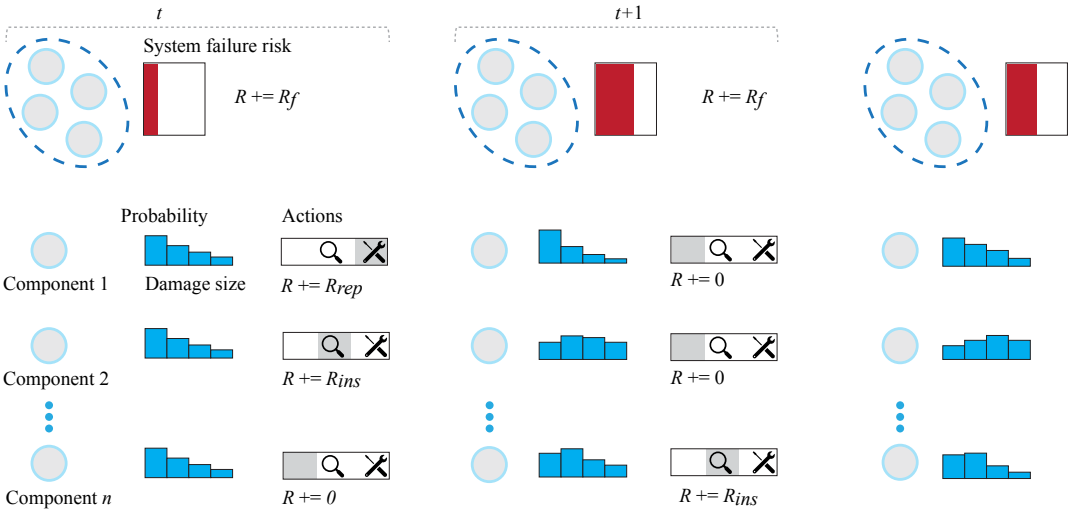


Figure 5.1: Overarching representation of an infrastructure management problem. The system failure risk is a function of the probability distribution over the components' damage condition. To control the system failure risk, components can be inspected or repaired at each time step t, and, typically, an agent controls one component. The objective of IMP's problem is to maximise the expected sum of discounted rewards by balancing the system failure risk R_f against inspections R_{ins} and repairs R_{rep}, all three being negative rewards. Here, we show three components with the same damage probability at time step t. When a component is not inspected nor repaired, its damage probability evolves according to a deterioration process. If a component is inspected, information from the inspection is also considered when updating the damage probability. If a component is repaired, the damage probability resets to its initial damage distribution.

measurement uncertainties, the components' damage is not perfectly known, and decisions are made based on a probability distribution over the damage condition, hereafter denoted as damage probability. The system failure probability is a function of components' damage probabilities. Starting from its initial damage distribution, each component's damage probability transitions according to a deterioration stochastic process and the decisions made [Morato et al., 2022]. Naturally, the damage probability transitions based on its deterioration model when the component is neither inspected nor repaired, i.e., do-nothing action. If a component is inspected, its damage probability is updated based on the inspection outcome. When a component is repaired, its damage condition is directly improved, and the damage probability resets to its initial damage distribution. A schematic of a typical IMP problem is shown in Figure 5.1.

IMP-MARL was introduced to generate more efficient strategies for managing engineering systems through cooperative MARL methods. In IMP-MARL, each agent is responsible for managing one constituent component in a system, making decisions based on the damage probability of the component. In addition to seeking to reduce component inspection and maintenance costs, agents should effectively cooperate to minimise the system failure risk. To assess the capability of cooperative MARL methods for gen-

erating effective policies for IMP problems involving many components, state-of-the-art cooperative MARL methods are benchmarked in terms of scalability and optimality. The benchmarked methods are presented in Chapter I and Chapter 3. Specifically, we benchmark five CTDE methods: QMIX, QVMix, QPLEX, COMA, and FACMAC, along with a decentralised method, i.e., IQL, and a centralised one, i.e., DQN. All tested MARL methods are compared against expert-based heuristic policies, which can be categorised as a state-of-the-art method to deal with IMP problems in the reliability engineering community [Luque and Straub, 2019; Morato et al., 2022]. In our study, three sets of IMP environments are investigated, including one related to offshore wind structural systems, where MARL methods are tested with up to 100 agents. These environments can be set up with two distinct reward models, one incorporating explicit cooperative objectives. Additionally, we ensure that the necessary code is publicly available so anyone can reproduce any published result[1]. This benefits an additional goal to facilitate the definition and implementation of new customisable environments.

From a societal perspective, more effective IMP policies contribute to a better allocation of resources. Additional societal impact is also made by controlling the risk of system failure events. For example, the failure of a wind turbine may affect the available electricity production. Beyond economic considerations, our proposed IMP-MARL framework can also be used to include sustainability and societal metrics within the objective function by accounting for those directly in the reward model.

Finally, the contributions of [Leroy et al., 2023] presented in this chapter can be outlined as follows:

- IMP-MARL is an open-source suite of environments, motivating the development of scalable MARL methods and the creation of new IMP environments, enabling the effective management of multi-component engineering systems and, as such, leading to a positive societal impact.

- In an extensive benchmark campaign, cooperative MARL methods are tested in high-dimensional IMP environments featuring up to 100 agents. The resulting management strategies are evaluated against expert-based heuristic policies. The source code is public so that we can reproduce our reported results and easily compare them with future developments.

- Based on the results, relevant insights for both machine learning and reliability engineering communities can be drawn, highlighting important challenges that must still be resolved. While cooperative MARL methods can learn superior strategies compared to expert-based heuristic policies, the relative performance benefit decreases in environments with over 50 agents. In specific environments, cooperative MARL policies are characterised by a high variance and sometimes underperform expert-based heuristic policies, suggesting the need for further research efforts.

1 https://github.com/moratodpg/imp_marl/

5.2 RELATED WORK

As introduced, RL is not the single approach when solving IMP problems, and we hereafter discuss how MARL is becoming popular in the field. Recent heuristic-based inspection and maintenance (I&M) planning methods generate IMP policies based on an optimised set of predefined decision rules [Luque and Straub, 2019; Bismut and Straub, 2021]. By evaluating only a set of decision rules out of the entire policy space, the previously mentioned approaches might yield suboptimal policies [Morato et al., 2022]. In the literature, one can also find POMDP-based methods applied to the I&M planning of engineering components, in most cases, relying on efficient point-based solvers [Papakonstantinou and Shinozuka, 2014a,b; Morato et al., 2022]. When dealing with multi-component engineering systems, solving point-based POMDPs becomes computationally complex. In that case, the policy and value function can be approximated by neural networks, enabling the treatment of high-dimensional engineering systems. Value-based and policy-based methods have been proposed in the literature for the management of engineering systems [Andriotis and Papakonstantinou, 2019, 2021; Morato et al., 2023], and some of them rely on CTDE methods [Nguyen et al., 2022; Saifullah et al., 2022]. Note that no open-source methods nor publicly available environments are provided in the abovementioned references. This emphasises the importance of our efforts to enhance comparison and reproducibility within the reliability engineering community.

5.3 IMP-MARL: A SUITE OF INFRASTRUCTURE MANAGEMENT PLANNING ENVIRONMENTS

In IMP, the damage condition of multiple components deteriorates stochastically over time, inducing a system failure risk that is penalised at each time step. Components can be inspected or repaired to control the system failure risk, yet incurring additional costs. The objective is to minimise the expected sum of discounted costs, including inspections, repairs, and system failure risk. This can be achieved through the agents' cooperative behaviour, assigning component inspections and repairs while jointly controlling the system failure risk. The introduced IMP decision-making problem can be modelled as a decentralised partially observable Markov decision process (Dec-POMDP). We hereafter define the components of this Dec-POMDP and formally define the deterioration, inspection, transition and reward models in Section 5.4.

5.3.1 *Environments formulation*

5.3.1.1 *States and observations*

As introduced, each agent in IMP perceives o_t^a, an observation corresponding to its respective component damage probability and the current time step. Each component damage

probability transitions based on a deterioration model, defined in Section 5.4. The damage probability is also updated based on maintenance decisions. Since the components' damage is not perfectly known, the state of the Dec-POMDP is defined as the collection of all components' damage probabilities along with the current time step: $s_t = (o_t^1, .., o_t^n, t)$. Following the discussion in Chapter 3, IMP is a jointly observable Dec-POMDP.

5.3.1.2 *Actions and rewards*

Each agent controls a component and collaborates with other agents to minimise the system failure risk while minimising local costs associated with individual repair and/or inspection actions. At each time step t, an agent decides u_t^a between (i) do-nothing, (ii) inspect, or (iii) repair actions. Both inspection and repair actions incur significant costs, formally included in the Dec-POMDP framework as negative rewards, R_{ins} and R_{rep}, respectively. Moreover, the system failure risk is defined as $R_f = c_F \cdot p_{F_{sys}}$ where $p_{F_{sys}}$ is the system failure probability, and c_F is the associated consequences of a failure event, encompassing economic, environmental, and societal losses. In IMP, we include two reward models. The first is a *campaign cost* model where a global cost, R_{camp}, is incurred if at least one component is inspected or repaired, plus a surplus, $R_{ins} + R_{rep}$, per inspected/repaired component. This campaign cost explicitly incentivises agents to cooperate. The second is a *no campaign cost* model, where the campaign cost $R_{camp} = 0$, and only component inspections and repairs costs are considered. Values of those costs are given in Section 5.4.4. Acting on finite-horizon episodes that span over T time steps, all agents aim at maximising the expected sum of discounted rewards

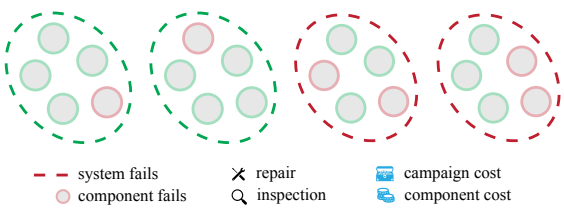
$$\mathbb{E}[R_0] = \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t \left[R_{t,f} + \sum_{a=1}^{n} \left(R_{t,ins}^a + R_{t,rep}^a\right) + R_{t,camp}\right]\right]. \tag{5.1}$$

5.3.1.3 *Real-world data*

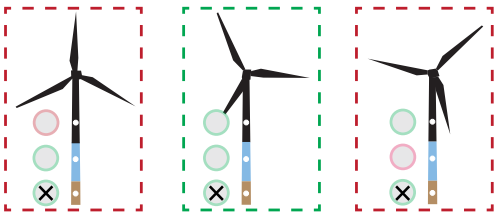
While IMP policies are trained based on simulated data, the policies can then be deployed to applications where real-world data streams are available. In that case, the damage condition of the components is updated based on collected real-world data, e.g., inspections.

5.3.2 *IMP-MARL environments*

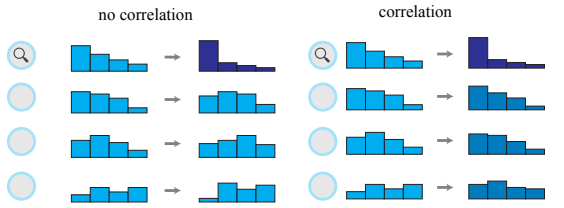
IMP-MARL provides three sets of environments to benchmark cooperative MARL methods. For all three, components are exposed to fatigue deterioration during a finite-horizon episode, inducing the growth of a crack over T time steps. The first set of environments is *k-out-of-n system* and refers to systems for which a system fails if (n-k+1) components fail. Those systems have been widely studied in the reliability engineering community [Barlow and Heidtmann, 1984]. The second type of environment is *correlated k-out-of-n system*



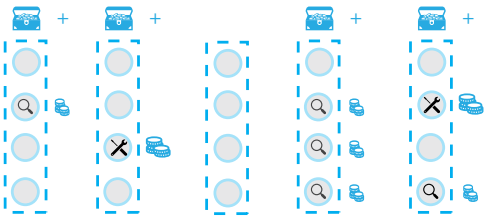
(a) A k-out-of-n system environment.



(b) An offshore wind farm environment.



(c) Uncorrelated and correlated initial damage distribution.



(d) A campaign cost environment.

Figure 5.2: Visual representation of available IMP-MARL environment sets and options. In 5.2a, a 4-out-of-5 system fails if two or more components fail. In 5.2b, a wind turbine fails if any constituent component fails. In 5.2c, when the environment is under deterioration correlation, the information collected by inspecting one component also influences uninspected components. In 5.2d campaign cost environments, a global cost is incurred if any component is inspected and/or repaired, plus a surplus per inspected/repaired component.

and is a variation of the first one for which the initial components' damage distributions are correlated. The last one is *offshore wind farm* and allows the definition of environments for which a group of offshore wind turbines must be maintained. They are graphically illustrated in Figure 5.2, and we hereafter provide details about these sets of environments. The implementation details are provided in Appendix A.1.

5.3.2.1 *k-out-of-n system*

In this set of environments, the components' damage probability distribution, $p(d_t^a)$, is defined as a vector of 30 bins, each representing a crack size interval. The failure probability of one component is defined as the probability indicated in the last bin. The specificity of a k-out-of-n system is that it fails if (n-k+1) components fail, establishing a direct link between the system failure probability and the component failure probabilities. The initial damage distribution among components is statistically independent for this first system, and the time horizon is $T = 30$ time steps. Since it is finite, we normalise each time step input and define $s_t = (p(d_t^1), ..., p(d_t^n), t/T)$ and $o_t^a = (p(d_t^a), t/T)$. The

interest of this system is that, in many practical scenarios, the reliability of an engineering system can be modelled as a *k-out-of-n system*.

5.3.2.2 *Correlated k-out-of-n system*

The second set of environments is the same as the previously defined one, with the difference that the initial damage distribution is correlated among all components. Therefore, inspecting one component also provides information about other uninspected components, depending on the specified degree of correlation. This setting is particularly challenging when approached in a decentralised mode without providing individual agents with component correlation information. To address this issue, in addition to their 30-bin local damage probability, the agents perceive correlation information α_t common to all, updated based on inspection outcomes collected from all components. We thus have: $s_t = (p(d_t^1), ..., p(d_t^n), \alpha_t, t/T)$ and $o_t^a = (p(d_t^a), \alpha_t, t/T)$. This damage correlation structure is inspired by practical engineering applications where initial defects among components are statistically correlated because components undergo similar manufacturing processes [Morato et al., 2022].

5.3.2.3 *Offshore wind farm*

The third set of environments differs from previous ones as it considers a system with wind turbines. Specifically, each wind turbine contains three representative components: (i) the top component located in the atmospheric zone, (ii) the middle component in the underwater zone, and (iii) the mudline component submerged under the seabed. In this case, the mudline component is considered impossible to inspect or repair, as it is installed under the seabed in an inaccessible region. Since only the top and middle components can be inspected or repaired, two agents are assigned for each wind turbine. Furthermore, the damage probability, $p(d_t^a)$, is a vector with 60 bins and transitions differently depending on the component location in the wind turbine, as corrosion-induced effects accelerate deterioration in certain areas. Besides individual component damage models, inspection techniques and their associated costs depend on the component location: inspecting or repairing the top components is cheaper than the middle one [Giro et al., 2022]. Moreover, while the mudline component cannot be directly maintained, its damage probability also impacts the failure risk of a wind turbine. In offshore wind farm environments, a wind turbine fails if one of its constituent components fails, and the overall system failure risk is defined as the sum of all individual wind turbine failure risks. In this case, $p(d_t^a)$ is modelled as a 60-bin vector, and the time horizon is $T = 20$. In this set of environments $s_t = (p(d_t^1), ..., p(d_t^n), t/T)$ and $o_t^a = (p(d_t^a), t/T)$.

5.4 MODELLING INFRASTRUCTURE MANAGEMENT IN IMP-MARL

This section formally defines the deterioration, inspection, transition and reward models implemented in IMP-MARL. These models drive the dynamics of the IMP-MARL environments.

5.4.1 *Deterioration models*

The deterioration processes introduced here correspond to fatigue deterioration mechanisms, yet corrosion, erosion, and many other practical infrastructure management problems can be similarly modelled.

5.4.1.1 *Correlated and uncorrelated k-out-of-n systems*

Throughout the following, the set of environments related to uncorrelated and correlated k-out-of-n systems are abbreviated as struct when referring to both. The structural components are exposed to fatigue deterioration in both k-out-of-n environments. Unless a repair is undertaken, the crack size d_t (i.e., damage condition) evolves over time t following

$$d_{t+1} = \left[\left(1 - \frac{m}{2}\right) C_{FM} S_R^m \pi^{m/2} n_S + d_t^{1-m/2}\right]^{2/(2-m)}, \tag{5.2}$$

where $\ln(C_{FM}) \sim \mathcal{N}(\mu = -35.2, \sigma = 0.5)$ and $m = 3.5$ stand for material variables, which directly influence the crack growth [Ditlevsen and Madsen, 1996]. Due to environmental and operational conditions, the components are subject to a dynamic load characterised by the stress range, $S_R \sim \mathcal{N}(\mu = 70, \sigma = 10\ \mathrm{N/mm^2})$, over $n_S = 10^6$ annual stress cycles, i.e., the number of load cycles experienced by the structural component in one year. At the initial step or after a component is repaired, the initial crack size is at its intact condition, defined by its initial distribution $d_0 \sim \mathrm{Exp}(\mu = 1\ \mathrm{mm})$, and a component level failure occurs when the crack size exceeds a critical length of $d_c = 20$ mm. The component failure probability p_F, defined as $p_F = P[g \leq 0]$, can be computed following a through-thickness failure criterion Hlaing et al. [2022], where the failure limit at time step t is formulated as $g_t = d_c - d_t$. At the system level, a failure event occurs if $n - k + 1$ components fail, and its corresponding system failure probability, $p_{F_{sys}}$, can be efficiently computed as a function of all components failure probabilities, as proposed in [Barlow and Heidtmann, 1984].

The continuous crack size is discretised into discrete bins to enable efficient Bayesian inference when inspection indications are available. Further details can be found in [Morato et al., 2022]. In a correlated k-out-of-n system, the initial crack size among components is correlated. In that case, the damage condition of each component is defined conditional on a common correlation factor, α, via a Gaussian hierarchical structure [Morato et al.,

Environment	Interval boundaries	Bins
struct	$[0, \exp\{\ln(10^{-4}) : (\ln(d_c) - \ln(10^{-4}))/28 : \ln(d_c)\}, \infty]$	30
owf	$[0, d_0 : (d_c - d_0)/(60 - 2) : d_c, \infty]$	60

Table 5.1: Description of the discretisation scheme implemented to discretise d_t.

2023]. In that case, the discretised damage bins should be defined conditionally based on the correlation factor. Table 5.1 defines the specific discretisation implemented in our environments.

5.4.1.2 *Offshore wind farm*

In this set of environments, abbreviated as owf in the following, a group of offshore wind substructures is considered, in which three representative structural components are modelled at different locations of the wind turbine: (i) at the atmospheric zone - upper level, (ii) at the splash zone - middle level, (iii) below the seabed - mudline. The deterioration, inspection, and cost models differ for each component. While the fatigue deterioration is calculated according to Equation 5.2, the expected dynamic load is defined based on industrial standards $S_r = q\Gamma(1 + 1/\lambda)Y$ [Lotsberg et al., 2016], corresponding to the expected value of a Weibull distribution defined by the scale parameters listed in Table 5.2, $q \sim \mathcal{N}$, and shape factor, $\lambda = 0.8$, weighted by a geometric parameter, $Y \sim \mathcal{LN}(\mu = 0.1, \sigma = 0.1)$. The initial crack size distribution is specified for all wind turbine components as $d_0 \sim \text{Exp}(\mu = 0.11)$ and the remaining specific fatigue variables associated with each wind turbine component are listed in Table 5.2. At the wind turbine level, a failure event occurs if one component of the wind turbine fails. The wind turbine failure risk is thus defined as the probability of failure multiplied by the consequences of a failure event. At the wind farm level, a wind turbine's damage condition does not influence the condition of the other wind turbines, and the wind farm system failure risk is defined as the sum of all turbines' failure risks.

5.4.2 *Inspection models*

The inspection models implemented in IMP-MARL are hereafter described. They define the likelihood of retrieving a specific inspection outcome as a function of the damage size.

5.4.2.1 *Correlated and uncorrelated k-out-of-n systems*

The inspection model is normally characterised depending on the accuracy of the measurement instrument, formally specified through probability of detection (PoD) curves, in

	Upper component	Middle component	Mudline component
$ln(C_{FM})$	$\mu = -26.45$ $\sigma = 0.12$	$\mu = -26.04$ $\sigma = 0.4$	$\mu = -26.12$ $\sigma = 0.39$
m	3	3	3
q	$\mu = 10.21$ $CoV = 25\%$	$\mu = 7.40$ $CoV = 25\%$	$\mu = 6.74$ $CoV = 25\%$
d_c	20	60	60
n_S	5,049,216	5,049,216	5,049,216

Table 5.2: Variables specified in the offshore wind farm deterioration models.

which the probability of observing a crack is defined as a function of the crack size [Morato et al., 2022]. In this case, the inspection model is described by an exponential distribution $p(i_{d_t}|d_t) \sim \text{Exp}(\mu = 8)$, defining the probability of observing a crack during an inspection.

5.4.2.2 *Offshore wind farm*

In this more practical set of environments, an eddy current inspection technique is considered, whose PoD is modelled by

$$p(i_{d_t}|d_t) = 1 - \frac{1}{1 + (d_t/\chi)^b}, \tag{5.3}$$

with $\chi = 0.4$ and $b = 1.43$ for the upper component and $\chi = 1.16$ and $b = 0.90$ for the middle component, according to industrial standards [Lotsberg et al., 2016]. The middle component, located below the water level, can naturally expect less accurate inspection outcomes.

5.4.3 *Transition models*

An overview of the transition model is explained hereafter. We refer the reader to [Morato et al., 2023] for a more detailed description. Since the crack size is discretised, the transition and inspection models can be stored in tables. This allows our IMP-MARL environments to be efficiently simulated. Alternatively, the crack size evolution could be directly computed at execution time, but this would incur an additional computational expense.

The transition model can be defined based on previously described deterioration and inspection models. If no inspection and maintenance are performed, i.e. do-nothing action, the damage condition progresses each time step according to the fatigue deterioration model formulated in Equation 5.2. Note that a time step represents a year in our environments. Considering that the damage follows a non-stationary deterioration process,

the crack size distribution d_{t+1} can be efficiently encoded as a function of the annual deterioration rate, τ_{t+1}, and the crack size at the previous time step d_t as $p(d_{t+1}|d_t, \tau_{t+1})$. Starting from $\tau_0 = 0$, the deterioration rate increases by one unit every year unless a component is repaired, in which case the deterioration rate returns to the initial value. The deterioration evolution over one time step is

$$p(d_{t+1}) = \sum_{\tau_{t+1}} \sum_{d_t} p(d_{t+1}|d_t, \tau_{t+1}) p(d_t) p(\tau_{t+1}). \tag{5.4}$$

If an inspection action is planned, a damage indication $i_{d_{t+1}}$ is collected, and the crack size distribution can be updated via Bayes' rule

$$p(d_{t+1}|i_{d_{t+1}}) \propto p(i_{d_{t+1}}|d_{t+1}) p(d_{t+1}), \tag{5.5}$$

where the likelihood corresponds to the specific inspection model, described by a probability of detection curve, as mentioned before. Since the damage probabilities are discrete, the normalisation constant can be straightforwardly computed by simply summing the unnormalised bins [Morato et al., 2022].

To enable efficient computation of the deterioration evolution under correlation, a Gaussian hierarchical structure is adopted, in which the crack size probability $p(d_t|\alpha)$ is defined conditional on a common factor α[Morato et al., 2023]. This work considers that the initial damage probabilities are equally correlated among components with a Pearson coefficient of 0.8. The damage transition, in this case, is

$$p(d_{t+1}|\alpha) = \sum_{\tau_{t+1}} \sum_{d_t} p(d_{t+1}|d_t, \tau_{t+1}) p(d_t|\alpha) p(\tau_{t+1}). \tag{5.6}$$

Once an inspection outcome is available, the common correlation factor is updated based on the new information, thus influencing all components. The likelihood of collecting one inspection indication given α is

$$p(i_{d_{t+1}}|\alpha) = \sum_{d_{t+1}} \Big[p(d_{t+1}|\alpha) p(i_{d_{t+1}}|d_{t+1}) \Big], \tag{5.7}$$

and the correlation factor can then be updated

$$p(\alpha|i_{d_{t+1}}) \propto p(\alpha) p(i_{d_{t+1}}|\alpha). \tag{5.8}$$

Finally, the marginal damage probabilities are computed as:

$$p(d_{t+1}) = \sum_{\alpha} \Big[p(d_{t+1}|\alpha) p(\alpha) \Big]. \tag{5.9}$$

Component	Campaign cost	R_{ins}	R_{rep}	c_f	R_{camp}
struct	False	-1	-20	-10,000	0
	True	-0.2	-20	-10,000	-5
owf upper level	False	-1	-10	-1,000	0
	True	-0.2	-10	-1,000	-5
owf middle level	False	-4	-30	-1,000	0
	True	-1	-30	-1,000	-5

Table 5.3: Rewards specified in our experiments.

5.4.4 *Reward model*

The goal of the agents is to maximise the expected sum of discounted rewards, $\mathbb{E}[R_0] = \mathbb{E}\left[\sum_{t=0}^{T-1}\gamma^t\left[R_{t,f}+\sum_{a=1}^{n}\left(R^a_{t,ins}+R^a_{t,rep}\right)+R_{t,camp}\right]\right]$. At each time step, the reward may include inspection R_{ins} and repair R_{rep} costs for all considered components, along with the system failure risk, which is defined as the system failure probability $p_{f_{sys}}$ multiplied by the associated consequences of a failure event c_f, formulated as $R_f = p_{f_{sys}} \cdot c_f$. A campaign cost R_{camp} may also be included if that option is active. The discount factor is defined as $\gamma = 0.95$ in our experiments, and the specific rewards are listed in Table 5.3.

5.5 EXPERIMENTS

5.5.1 *Tested methods*

In an extensive benchmark campaign, we test seven RL methods. The centralised controller, which has an action space that scales exponentially with the number of agents, is trained with the centralised method DQN and is the only method taking s_t as input. Furthermore, for the decentralised method tested is IQL, in which all agents are independently trained. Regarding the five CTDE methods, we investigate three value-based methods, QMIX, QVMix, and QPLEX, as well as two actor-critic methods, COMA and FACMAC. We selected these methods for our benchmark study because they are well established, and their implementations are open-sourced and available within the PyMarl framework [Samvelyan et al., 2019].

All investigated RL methods are compared against a representative baseline in the reliability engineering community [Luque and Straub, 2019; Morato et al., 2023]. This baseline, referred to as expert-based heuristic policy, consists of a set of heuristic decision rules defined based on expert knowledge. The heuristic policy includes both parametric

IMP environments	Number of agents				
k-out-of-n system	3	5	10	50	100
Correlated k-out-of-n system	3	5	10	50	100
Offshore wind farm	2	4	10	50	100

Table 5.4: Number of agents specified in all investigated IMP environments.

and non-parametric rules. Parametric decision rules depend on two parameters: (i) the inspection interval and (ii) the number of inspected components. Non-parametric rules involve taking a repair action after detecting a crack and prioritising component inspections with higher failure probability. To determine the best heuristic policy for each environment, all parametric rule combinations are evaluated over 500 policy realisations, thereby identifying the heuristic policy that maximises the expected sum of discounted rewards among all policies evaluated.

5.5.2 *Experimental setup*

The abovementioned seven MARL methods are tested in the three previously defined sets of IMP environments. The environments differ by the number of agents and whether they include a campaign cost model. The numbers of agents tested in the six types of environments are presented in Table 5.4. To objectively interpret the variance associated with the examined MARL methods. As explained in Section 5.3, an agent makes decisions based on its local damage probability, the current normalised time step, and sometimes correlation information is additionally provided, while the state, used by DQN and CTDE methods, encompasses all of the information combined. In all cases, the action space features three possible discrete actions per agent, except for DQN, where the centralised controller selects an action among the 3^n possible combinations. For complexity reasons, we only test DQN in k-out-of-n environments featuring 3 and 5 components and in environments with 1 and 2 wind turbines.

Given the importance of hyperparameters on the performance of RL methods [Gorsane et al., 2022], we initially selected their values reported by the original authors. In an attempt to objectively compare the examined methods, parameters that play the same role across methods are equal. Notably, the learning rate and gamma, among others, are identical in all experiments. The controller agent network features the same architecture in all methods, consisting of a single GRU layer with a hidden state composed of 64 features encapsulated between fully connected layers and three outputs, one per action, except for DQN, where the network output includes 3^n actions. In our case, DQN's architecture includes additional fully connected layers and a larger size of hidden GRU states. Moreover, following common practice, agent networks are shared among agents,

and thus, a single agent network is trained. Specifically, we train only one network for all agents instead of training n distinct agent networks. The training process with a single agent network improves data efficiency because the same episode can be used to perform n backpropagations through the same agent network, using n different observations. In contrast, if training is performed with n different agent networks, only one backpropagation per agent network would be possible with a single episode. To allow diversity in agents' behaviour, a one-hot encoded vector is also added to the input of this shared network to indicate which one of the n agents is making the decision. In CTDE methods, critics or mixers are also incorporated at the training stage with specific architectures according to each method and environment configuration. In most cases, the neural networks are updated after each played episode based on 64 episodes sampled from the replay buffer containing the latest 2,000 episodes. The only exception is COMA, which follows an on-policy approach, updating the network parameters every four episodes. For value-based methods, the training episodes are played following an epsilon greedy policy, whereas test episodes are executed with a greedy policy. The epsilon value is initially specified as 1 and linearly decreases to 0.05 after 5,000 time steps. This is different for COMA and FACMAC. Appendix A.3.1 and the source code list more details and all parameters.

The number of time steps allocated for one training realisation is 2 million time steps for all methods. These 2 million training time steps are executed with training policies, e.g. ϵ-greedy policy, saving the networks every 20,000 training time steps. To evaluate them, we execute 10,000 test episodes and obtain the average sum of discounted rewards per episode per saved network. These test episodes are executed with testing policies, e.g., the greedy policy. We show in Appendix A.3.2 that 10,000 test episodes are needed due to the variance induced in the implemented environments. We emphasise that ten training realisations are executed with different seeds for the same parameter values. Finally, hardware and experiment duration are provided in Appendix A.3.3.

5.6 RESULTS

The benchmark campaign results are presented in a boxplot showcasing the relative performance of MARL methods with respect to expert-based heuristic policies in terms of their expected sum of discounted rewards. Each boxplot represents each of the ten seeds by its best policy, which achieved the highest average sum of discounted rewards during evaluation. The construction of such a boxplot is presented in Figure 5.3, and all results are presented in Figure 5.4. Our analysis relies on relative performance metrics because the optimal policies are unavailable. Finally, the corresponding learning curves and the best-performing policy realisations can be found in Appendix A.4.

MARL-based strategies outperform expert-based heuristic policies. While heuristic policies provide reasonable IMP policies, most tested MARL methods yield a substantially higher expected sum of discounted rewards. Yet, the variance over identical

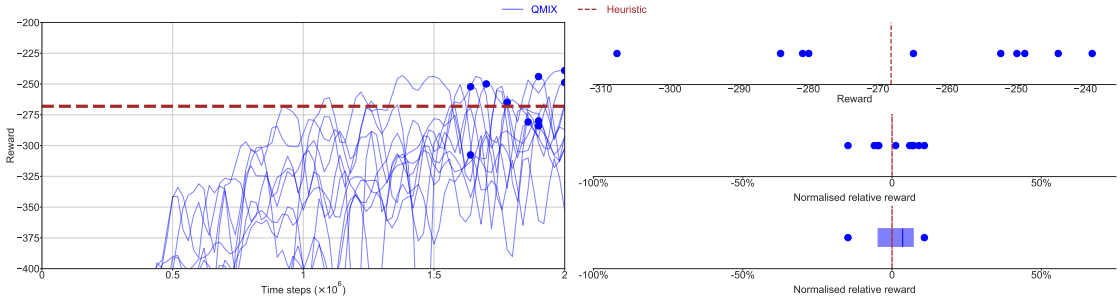


Figure 5.3: Visual description of the iterative process followed to generate the boxplots showcased in Figure 5.4. [Left] Learning curves corresponding to 10 QMIX training seeds in a k-out-of-n system with 50 agents. The markers highlight the policies that result in the highest expected sum of discounted rewards during evaluation, i.e., one policy per seed. [Right] The ten policies and the heuristic are displayed at the top as a function of the expected sum of discounted rewards score obtained. The middle plot presents them as a function of normalised relative rewards with respect to the heuristic, i.e., (x - h) / h. Finally, a boxplot is constructed at the bottom based on the previously calculated ten normalised relative rewards.

MARL experiments is still sometimes significant. In environments with no campaign cost, the performance achieved by MARL methods with respect to the baseline differs in configurations with a high number of agents, as shown at the top of Figure 5.4. In contrast, MARL methods reach better relative results in environments with many agents when the campaign cost model is adopted, as illustrated at the bottom of Figure 5.4. In general, the superiority of MARL methods with respect to expert-based heuristic policies is justified by the complexity of defining decision rules in high-dimensional multi-component engineering systems, where the sequence of optimal actions is challenging to predict based on engineering judgment [Morato et al., 2023].

IMP challenges. In correlated k-out-of-n IMP environments, the variance over identical MARL experiments is higher than in the uncorrelated ones, emphasising a specific IMP challenge. Under correlation, inspecting one component also provides information to uninspected components, impacting their damage probability and thus hindering cooperation between MARL agents. Another challenge is imposed in offshore wind farm environments, where the benefits achieved by MARL methods with respect to the baseline are also reduced in environments with a high number of agents. This can be explained by the fact that each wind turbine is controlled by two agents and is independent of other turbines in terms of rewards. Each agent must then cooperate closely with only one of all agents, hence complicating global cooperation in environments featuring an increasing number of agents.

Campaign cost environments. Yet another challenge can be observed in campaign cost environments under 50 agents, where MARL methods' superior performance with respect to heuristic policies is more limited. The aforementioned environments are challenging for MARL methods because agents should cooperate to group component in-

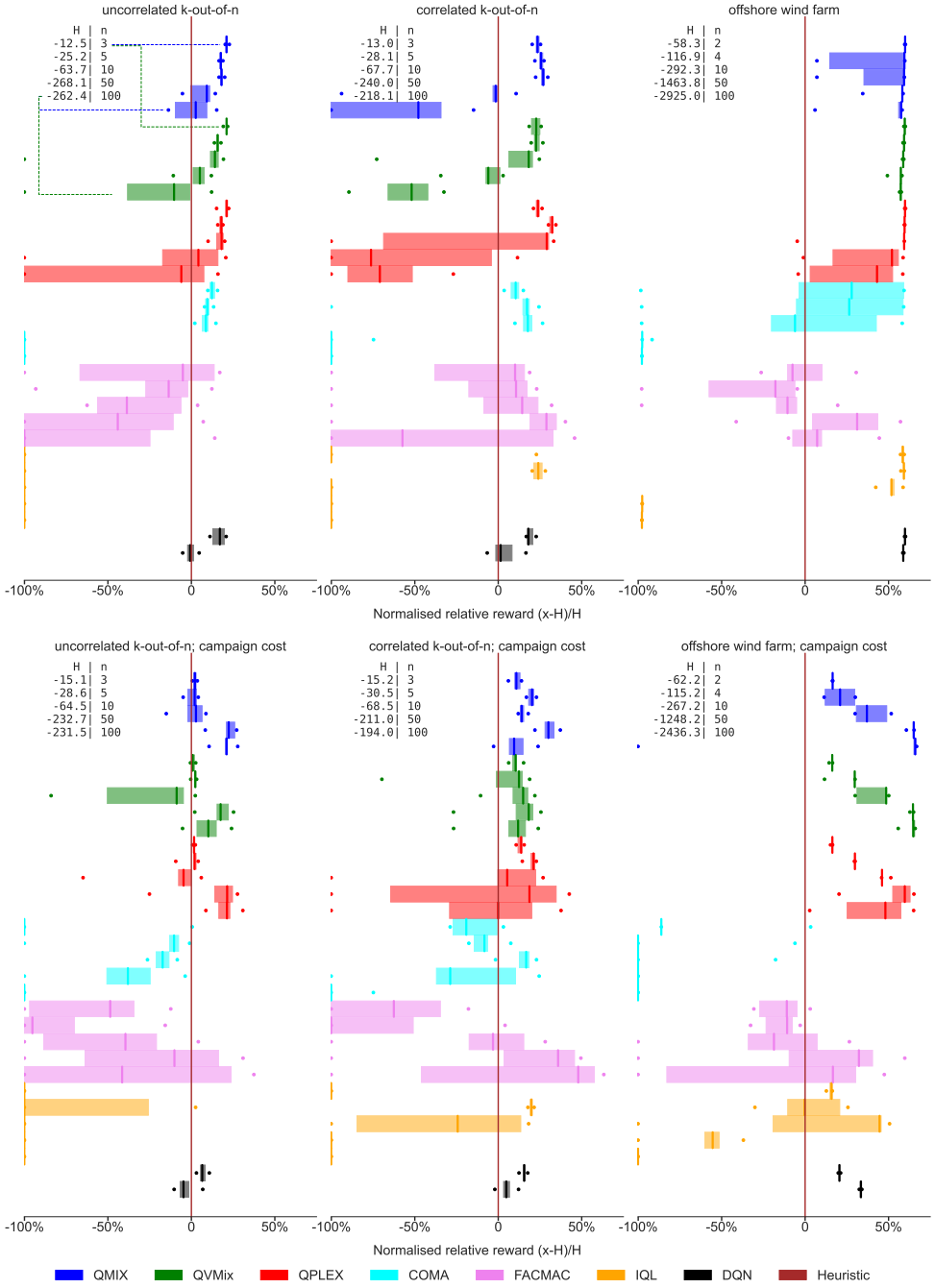


Figure 5.4: Performance reached by MARL methods in terms of normalised discounted rewards with respect to expert-based heuristic policies in all IMP environments, H referring to the heuristics result. Every boxplot gathers the best policies from each of 10 executed training realisations, indicating the 25th-75th percentile range, median, minimum, and maximum obtained results. The coloured boxplots are grouped per method, vertically arranging environments with increasing n agents, as indicated in the top-left legend boxes. Note that the results are clipped at -100%.

spection/repair actions together, saving global campaign costs. In addition, the heuristic policies are designed to schedule group inspections, being favourable in this case automatically. This is confirmed by the learning curves presented in Figures A.7 and A.8 in Appendix A.4. On the other hand, in environments with more than 50 agents, MARL methods substantially outperform heuristic policies. At least one component is inspected or repaired at each time step and the results reflect that avoiding global annual campaign costs becomes less crucial.

Centralised RL methods do not scale with the number of agents. DQN reaches better results than heuristic policies, though it achieves lower rewards than CTDE methods in most environments despite benefiting from larger networks during execution. This highlights the scalability limitations of such centralised methods, mainly because they select one action from each possible combination of component actions.

IMP demands cooperation among agents. The results reveal that CTDE methods outperform IQL in all tested environments, especially those with many agents. This confirms that realistic IMP problems demand coordination among component agents. Providing only independent local feedback to each IQL agent during training leads to a lack of coordination in cooperative environments, also shown by Rashid et al. [2018]. However, the performance may be improved by enhancing networks' representation capabilities by including more neurons, yet this is true for all investigated methods.

Infrastructure management planning via CTDE methods. Overall, CTDE methods generate more effective IMP policies than the other investigated methods, demonstrating their capabilities for supporting decisions in real-world engineering scenarios. While Figure 5.4 presents the variance of the best results across runs, the learning curves further confirm this finding in Appendix A.4. In particular, QMIX and QVMIX generally learn effective policies with low variability over their runs. Slightly more unstable, QPLEX also yields results similar to those of QMIX and QVMIX. While being able to outperform heuristic policies in almost every environment, FACMAC exhibits a high variance among runs. However, FACMAC effectively scales up with the number of agents and environment complexity (as reported by their authors [Peng et al., 2021]), achieving some of the best results in IMP environments with over 50 agents as well as in correlated IMP environments. The results also suggest that COMA is our benchmark's least scalable MARL method. This can be attributed to the fact that the computation of the critic's counterfactual becomes challenging with an increasing number of agents. Additional results are presented in Appendix A.4, where many tables and figures can be found.

5.7 DISCUSSION AND FUTURE WORK

This work offers an open-source suite of environments for testing the scalability of cooperative MARL methods for efficiently generating IMP policies. Through our publicly available code repository, we also encourage the implementation of additional IMP en-

vironments, such as bridges, transportation networks, pipelines, and other relevant engineering systems. This allows specific disciplinary challenges to be identified in a common simulation framework. Based on the reported benchmark results, we can conclude that CTDE methods generate effective infrastructure management policies in real-world engineering scenarios. While the results reveal that MARL methods outperform expert-based heuristic policies, additional research efforts should still be devoted to developing scalable cooperative MARL methods.

While we model the IMP decision-making problem as a Dec-POMDP, modelling IMP problems as mean-field games [Laurière et al., 2022] is a promising direction to be considered in environments with an increasing number of agents. Moreover, specific improvements are still required in environments where a global cost is triggered from the actions taken by any local agent, e.g., global campaign cost. Besides, more stable training is still needed in environments where local information perceived by one agent can influence the damage condition probabilities of others, as in the correlated IMP environments. More realistic and challenging environments for cooperative MARL methods could be investigated. One example is assigning campaign costs to specific groups of components instead of specifying only one global campaign cost. Another example is to address systems composed of more heterogeneous components.

Part III

COOPERATE AGAINST AN OPPOSING TEAM

6

COMPETITION

Outline

This chapter introduces basic concepts and literature stories of competitive and general-sum settings of MARL. This is the third background chapter of this manuscript. Section 6.1 introduces this third part of the manuscript, wrapping up the two MARL settings introduced earlier. The solution of each agent maximising its return can suffer from different problems, and we present other types of solutions in Section 6.2. Section 6.3 presents historical methods to learn in non-cooperative environments, from game theory to adversarial search and MARL. We conclude this chapter in Section 6.4 by defining methods to solve competition and general-sum settings.

6.1 INTRODUCTION

In Part I of this manuscript, we present the general framework for MARL and its different settings. In Part II, we focused on the cooperative setting, where all agents receive the same reward. In this third part of the manuscript, we focus on a different setting where two teams of agents compete against each other. This chapter provides an overview of MARL in the competition and general-sum setting. The next one then presents the two-team competition setting and conducted experiments to highlight how to train teams by combining the methods defined in Part II and the methods to train agents to compete presented in this chapter.

One problem of MARL is that agents may never stop adapting their policies to the new policies of other agents who are also adapting to changes since they are also learning. This never-ending adaptation cycle led many researchers to define equilibria, achieved when agents do not want to change their policy. Moreover, when agents receive a common reward, like in Dec-POMDP, deciding which of two joint policies is the best is straightforward. However, this becomes less obvious when agents receive different rewards, like in competitive and general-sum settings, where the change in the return of agents may not be fair between two joint policies, equilibria or not. As introduced in previous chapters, finding and selecting an equilibrium represents one of the main challenges of MARL. Section 6.2 discusses different solutions in MARL to achieve an equilibrium and rank policies.

Once these solutions are defined, Section 6.3 presents an overview of the history of acting in multi-agent systems. As will be discussed, game theory defines solution concepts,

and we cover some of its foundations, including historical RL and planning methods. Finally, Section 6.4 discusses self-play and population-based training, two learning scenarios allowing agents to learn to act in complex games.

6.2 SOLUTIONS

The solution to an MDP is the policy that maximises the agent's expected return. While finding the optimal policy may be challenging, deciding which of two policies is better, given the expected return, is simple. In a POSG, the return of each agent is $G_0^{a_i} = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{T-1} \gamma^t r_t^{a_i}\right]$, also shortened $G^i(\boldsymbol{\pi})$ in the following. Choosing between two joint policies is as straightforward in a Dec-POMDP as in SARL because agents all receive the same reward, and their return changes identically. However, in other settings, when agents receive different rewards, it can become challenging to determine which joint policy is a better solution to the POSG. This is because the change in the return of each agent will likely be different. Moreover, maximising each agent's return may lead to never-ending learning, where each agent adapts to the others. Therefore, alternative solutions to games have been defined, and this section provides some. Albrecht et al. [2023] defines MARL as a pair composed of an environment and a solution concept. While this section defines some, a thorough list can be found in [Albrecht et al., 2023].

One of the solutions is the best response. It evaluates the policy of an agent a_i by considering that the policies of all other agents $\boldsymbol{\pi}^{-\boldsymbol{i}}$ are fixed. The best response policy is in the set of $\arg\max_{\pi^i} G^i(\pi^i, \boldsymbol{\pi}^{-\boldsymbol{i}})$ and may not be unique. It is indeed the solution that provides the optimal return when other agents' policies are fixed. It can be used to iteratively compute a solution for all agents by computing the best response policy of one agent after the others, as will be shown later.

Another well-known solution is the minimax solution, which we define in a two-player zero-sum game. In such a game, the agent's reward can be modelled as the negative of the other's, so $r_t^{a_i} = -r_t^{a_{-i}}$ and $G^i(\boldsymbol{\pi}) = -G^{-i}(\boldsymbol{\pi})$. A joint policy $\boldsymbol{\pi} = (\pi^i, \pi^{-i})$ is a minimax solution if $G^i(\boldsymbol{\pi}) = \max_{\pi^i} \min_{\pi^j} G^i(\pi^i, \pi^j)$. Again, several minimax solutions can exist, but their returns are equal, called the game's minimax value. In other words, the minimax value can be achieved from a given state, typically the initial one, given that both players play optimally [Russell and Norvig, 2010]. Regarding best responses defined previously, one can interpret the minimax joint policy as the best response of one agent to the best response policy of the other [Albrecht et al., 2023].

Extending this concept of mutual best responses to general-sum games with two or more agents can be the definition of the Nash equilibrium [Albrecht et al., 2023]. This equilibrium is achieved when no agent benefits from changing its current policy. A joint policy $\boldsymbol{\pi}$ is a Nash equilibrium if $\forall a_i$, any π_i' is such that $G^i(\pi_i', \boldsymbol{\pi}^{-\boldsymbol{i}}) \leq G^i(\boldsymbol{\pi})$ [Nash Jr, 1950]. For some environments, only stochastic policies can achieve a Nash equilibrium. For the rock-paper-scissor game, one example of stochastic policies achieving a Nash equilibrium is to take each action with a uniform probability. Moreover, a Nash equilibrium

is not necessarily unique and has different expected return values for each agent. This highlights the challenge of selecting the optimal equilibrium because the difference in returns between equilibria is not always the same between agents.

These three solutions have been demonstrated in an SG, and we refer the reader to the book of Albrecht et al. [2023]. They also discuss the complexity of computing an equilibrium and explain that Nash equilibrium cannot be computed in polynomial time in the general case. Variants and extensions of the Nash equilibrium also exist. The ϵ-Nash equilibrium allows a surrounding region to improve flexibility and reduce the strict Nash equilibrium. Until now, we have silently considered that agents take actions independently. However, this is not always the case. Indeed, agents may have correlated policies leading to the correlated equilibrium, also called coarse equilibrium. This equilibrium generalises the Nash equilibrium to correlated policies. Both are fully covered by Albrecht et al. [2023], who also conclude the definitions of equilibrium by highlighting their three main limitations: the possible sub-optimality, the non-uniqueness, and the incompleteness of these equilibrium solutions. The latter means that the equilibrium solution does not provide specific actions to guide players back to the equilibrium path once they deviate from it.

A non-equilibrium solution is a Pareto optimal joint policy. It comes from the multi-objective optimisation literature [Ehrgott, 2012]. A joint-policy $\boldsymbol{\pi}$ Pareto dominates an other one $\boldsymbol{\pi'}$ if $G^i(\boldsymbol{\pi}) \geq G^i(\boldsymbol{\pi'}) \forall i$ but for one agent a_i, $G^i(\boldsymbol{\pi}) > G^i(\boldsymbol{\pi'})$. In other words, a joint policy is Pareto-dominated if one agent can achieve better returns without reducing the returns of others. A Pareto optimal joint policy is easily defined as not Pareto-dominated by any other. The difference between Nash equilibrium is that here, no agent can improve its return without worsening one of the others. Meanwhile, the Nash equilibrium considers a single agent to change without becoming worse. This Pareto optimality combined with equilibrium allows the solution to be improved. Specifically, it reduces the number of equilibriums considered because it seems evident that a Pareto optimal equilibrium is better than a non-Parto one. However, there is often no unique Pareto-optimal joint policy. More importantly, a joint policy can be Pareto-optimal without being an equilibrium one. Finally, Pareto optimality does not consider how the return is distributed amongst agents, leading to the notion of social welfare and fairness detailed in [Albrecht et al., 2023].

6.3 HISTORY

Game theory (GT) [Von Neumann and Morgenstern, 1944] is as much the foundation of MARL as is RL [Nowé et al., 2012; Albrecht et al., 2023]. The solution concepts presented in Section 6.2 and the classification in three types of multi-agent settings are results from game theory. GT with RL has a long history, ranging from normal-form games to repeated normal-form games, and POSG is the complex extension of these settings. In the following, we present the premise of binding GT and RL and cover

methods that have led to what exists now to try to solve POSG. We finish with planning methods dedicated to the extensive-form games that contributed to POSG's progress. This section presents some methods at the foundation of MARL to highlight foundations from dynamic programming. These methods are described in [Albrecht et al., 2023] and [Russell and Norvig, 2010], already exploited in this manuscript, in addition to the chapter "Game theory and Multi-agent RL" [Nowé et al., 2012] presenting others.

In non-repeated normal-form games, finding the minimax solution with linear programming [Albrecht et al., 2023] is possible. Game theory with RL also started in normal-form games. For another example, we can cite the study on independent Q-Learning in cooperative normal-form games by Claus and Boutilier [1998]. Many methods also have been proposed for SG. Learning the optimal value function for all states in an SG is possible using dynamic programming. Value iteration [Shapley, 1953] is an example of such a method, the same name as its homologue in MDP [Sutton and Barto, 2018] despite being slightly different in the concept. As in SARL, this method requires knowledge of the environment model and has the same limitations.

A class of methods involves modelling other agents to predict their policy. This is called agent modelling, which means that their policy is approximated from the observed past actions. Once other agents' policies are modelled, the best response against their approximated policy can be computed. This is done in fictitious play [Brown, 1951] for the normal-form game or more recently in Poker [Heinrich et al., 2015] Extensions consider additionally learning how the opponent will update its strategy [He et al., 2016; Foerster et al., 2018a].

There is also a different class of method called joint action learning. These methods directly learn n state-joint-actions value functions $Q_i(s, \boldsymbol{u})$ to learn equilibrium Q value iteratively with a temporal difference. However, these methods require assumptions on the type of equilibrium agent aim. This leads to different algorithms, such as Minimax Q-learning [Littman, 1994], Nash Q-learning [Hu and Wellman, 2003] and Correlated Q-learning [Greenwald et al., 2003]. These algorithms have some limitations in the type of equilibrium they can achieve [Albrecht et al., 2023]. The joint action learning class of the method forces all agents to follow the same equilibrium desire, or at least learn, considering that they follow this same desire of equilibrium.

Many more details and methods can be found in [Albrecht et al., 2023], [Nowé et al., 2012] and [Russell and Norvig, 2010]. We presented an overview of existing methods to provide a bit of history. In the next section, we focus on self-play and population-based methods that motivated the work presented in Chapter 7. However, we finish this section with planning methods in multi-agent systems.

Section 2.4.2 defines planning as computing a local solution using the environment model. Planning has a long history in multi-agent systems, especially in fully observable two-player zero-sum games with turn-taking actions. These games are called extensive-form games, different from the repeated normal-form ones because agents do not act simultaneously [Nowé et al., 2012]. Famous examples include Chess and Go. In such

board games, agents take action, modifying the state of the board that is observable by both. Both obtain zero rewards except at the end of the game. If one wins, it obtains a reward equal to 1; if it is a draw, both receive a reward equal to 1/2. The sum of all rewards is thus always 1. One family of planning methods to play this game is adversarial search [Russell and Norvig, 2010].

Adversarial search is an extension of classical tree search, where a tree representing all possible games is built by considering all possible successive actions of the agent. In such a tree, each node represents a state, and each edge represents an action. A path from the root to a leaf thus represents one possible game. The best solution is, therefore, the path toward the leaf providing the best outcomes. Growing such a tree can become intractable when the number of possible games increases. This leads to methods that build only a subpart rather than the entire tree using information during the search. This information is typically given by an evaluation function approximating the best achievable outcomes from a given node. A well-known informed tree search method is A*.

Back to adversarial search, growing the tree of all possibilities is possible. However, one needs to consider the presence of other agents, especially the uncertainty regarding their future actions. This motivates the need for a search algorithm to be executed for every action. In Section 6.2, we presented the minimax solution providing the minimax value if both players play optimally. The minimax algorithm is an adversarial search algorithm that computes this minimax value from each tree node. Therefore, the optimal action is to select the action with the maximum minimax value or the minimum for the other player. However, despite being able to prune such a tree and not compute all values with the alpha-beta minimax algorithm, problems still arise as the tree's size increases. A solution is thus to define a criterion to stop expanding the tree at a given node and approximate the minimax value at this node. However, this approximation cannot always be perfect and could lead to suboptimal choices in complex games. The reader should now understand why a game's complexity is linked to the number of possible games.

This leads to Monte Carlo tree search (MCTS), in which states are evaluated by playing games with random actions instead of an evaluation function. This allows the agent to estimate the win rate achievable from a state without knowledge. In MCTS, the tree is expanded iteratively by selecting nodes to evaluate by random games. The underlying idea is not to explore the tree arbitrarily but to focus on nodes that frequently provide good results. Simple rules define the selection and expansion process of nodes. Random games are played from intermediate nodes, not always the root, whose direct parent have already been used to start a random game. For each expanded node, the win rate associated with the action taken to reach it and the number of times this node has been selected are stored. These two quantities drive the selection process, highlighting the exploration/exploitation dilemma to play games from unexplored nodes while maintaining certainty of good ones.

AlphaGo [Silver et al., 2016] allowed an agent to beat the best humans by modifying the traditional MCTS to benefit from the learning capabilities of neural networks. Instead of

playing random games to approximate win rates and probabilities of taking actions, two neural networks approximate them. The MCTS searches are guided by neural networks, which play games that are then used to train the neural network. These networks are initially trained with a dataset of human games before using games generated by letting the agent play against itself. AlphaGo Zero [Silver et al., 2017] is the extension without pre-training the networks with human data and other minor changes. They have been generalised to AlphaZero [Silver et al., 2018], which also plays Go and Shogi. Later, such tree search method has been extended to MuZero, which learns the model of any game to play it [Schrittwieser et al., 2020]. MCTS and these two extensions are part of the self-play class of algorithms, training the same strategy for all agents in the environment. The following section presents works that combine model-free RL and self-play. The extension considers population-based training, where multiple agents train by playing against each other.

6.4 SELF-PLAY AND POPULATION-BASED TRAINING

Its name implicitly defines self-play as methods that train an agent by making it play against itself. In the previous section, we mentioned the model-based methods that have been generalised to AlphaZero [Silver et al., 2018]. However, self-play can also be exploited with model-free methods. One of the first methods combining model-free RL with neural network and self-play is TD-Gammon, which plays backgammon, starting from random network parameters to achieve a master level [Tesauro, 1994]. Later, the success of modern RL methods allowed the achievement of human-level performance in other games, such as in Stratego [Perolat et al., 2022]. Self-play has also been used with more than two competing agents in mixed cooperative-competitive environments, like the framework introduced in Chapter 7, such as Hide and Seek [Baker et al., 2019] or open AI five [OpenAI et al., 2019]. In Hide and Seek, they trained agents in self-play with PPO and a centralised critic with access to the state, a CTDE method. They demonstrated that the auto-curriculum provided by self-play, in addition to learning successive strategies, allows agents to understand the environment better and perform better on new tasks than agents trained with the classical unsupervised RL method, where agents learn to explore the environment. Self-play also shown benefits for autonomous driving [Cornelisse and Vinitsky, 2024].

In some, the self-play agent sometimes plays against an old version of itself. This allows it to be resilient against diverse strategies, not generalise, and not forget how to play them. An essential requirement of self-play is the policy that the agent learns should be able to act in place of other agents. When agents are heterogenous, typically in their uncertainty about the state or in their action space, self-play may not be possible. A solution is to train a population of policies for each agent [Jaderberg et al., 2017]. Moreover, in self-play, when agents play against a past version of themselves, it is considered a particular case of population-based training because several policies are kept to play against.

There are several examples of games where human-level performance has been achieved with population-based training. We can think of Quake III Arena in Capture the Flag mode [Jaderberg et al., 2019], StarCraft [Vinyals et al., 2019]. In these two, self-play is still possible because agents are the same. Still, population-based training allows for solving complex games because, on the one hand, it updates all policies from the population instead of just one. On the other hand, there are many different strategies in such games. Indeed, the classical process of population-based training starts with several policies, which are evaluated and updated as they play games against each other. The population may be modified by increasing its size, duplicating some of its individuals, or freezing others. Maybe the most successful method when training a population is to use policy space response oracles (PSRO) [Lanctot et al., 2017; Muller et al., 2020]. In short, PSRO reframes population-based training as a normal-form metagame in which actions are policies. It computes the probabilities of selecting each policy in each population of agents to achieve an equilibrium. From these computed distributions, it is possible to create new best-response policies to be added to the population. The complexity of such a method is immense because, on the one hand, the distributions of actual strategies to achieve equilibrium should be established. On the other hand, the best responses to such distribution should be computed by training new agents.

7

TWO-TEAM MARKOV GAME

Outline

This chapter presents how to combine cooperative and competitive methods to train two teams to compete. Section 7.1 introduces the motivations behind this study. Following this, we define the two-team Markov game in Section 7.2 and the adaptation of SMAC to this mixed cooperative-competitive framework in Section 7.3. Section 7.4 describes the learning scenarios used to train and how to evaluate teams, followed by the experimental setup in Section 7.5. The corresponding results are presented in Section 7.6, and we conclude this chapter with discussions in Section 7.7. This chapter is an adapted version of the publication [Leroy et al., 2022] *Value-based CTDE methods in symmetric two-team Markov game: from cooperation to team competition*, P. Leroy, J. Pisane, and D. Ernst. Deep Reinforcement Learning Workshop NeurIPS, 2022

7.1 INTRODUCTION

Many applications exist where two teams of multiple agents compete, such as in games (Pommerman [Resnick et al., 2018]), cyber security (CyberBattleSim [Kunz et al., 2022]), or in robotics (RoboCup [Kitano et al., 1997]). In certain use cases, agents are not fully aware of the entire environment, such as in Cyber-Physical Production Systems [Phan et al., 2020], Hide and Seek [Baker et al., 2019] or Capture the Flag [Jaderberg et al., 2019]. In this chapter, we combine the cooperative methods from Part II with the competitive ones defined in Chapter 6 to train teams to compete in a mixed cooperative-competitive POSG. The setting is a symmetric two-team Markov game where two teams of the same agents compete. The goal is identifying how to train a team to be resilient to different adversarial strategies.

Specifically, we study the difference between three learning scenarios: learning against a stationary team policy, against a single evolving team strategy (self-play), and against multiple evolving team strategies within a population of learning teams. To evaluate the performance of each learning scenario, we created a new competitive environment suite by modifying SMAC, initially designed for cooperation and defined in Section 3.3.1. We chose symmetrical competition to ensure fair and balanced competition and the possibility of controlling either team with the same agents. In two different SMAC environments, teams are trained with the three value-based CTDE methods, QMIX, MAVEN, and

QVMix, each with three learning scenarios. These methods are defined in Section 3.4. We then analyse how they perform when faced with multiple opposing strategies by forming test populations of trained teams. Our results suggest that when competing against several possible strategies, teams trained in a population achieve the best performance, but a selection process is required to select the best team. We reached this conclusion irrespective of whether or not the stationary strategy was better than all trained teams.

7.2 TWO-TEAM MARKOV GAME

Our framework is a particular case of a POSG defined in Chapter 2 with a reward function that provides two rewards, one per team. Specifically, we are in a symmetric, mixed cooperative-competitive, partially observable two-team Markov game defined by a tuple $[\mathcal{S}, \mathcal{Z}, \mathcal{U}, n, O, R, P, \gamma, T, p]$. The components of this tuple are similar to the ones defined in the stochastic game in Section 2.2 or in the Dec-PODMP in Section 3.2, and we hereafter define only the different ones. The main one is that $2n$ agents learn in this framework, divided into two teams of n agents. Let $\mathcal{I} = \{1, .., n\}$ and $\mathcal{J} = \{1, -1\}$, the i^{th} agent of the j^{th} team is denoted by $a_{i,j}$ with $i \in \mathcal{I}$ and $j \in \mathcal{J}$. Since we assume team symmetry, agents $a_{i,1}$ and $a_{i,-1}$ share the same action space $\forall i$, leading to only one team joint action space $\mathcal{U} = \bigtimes_{i \in \mathcal{I}} \mathcal{U}_{a_{i,1}} = \bigtimes_{i \in \mathcal{I}} \mathcal{U}_{a_{i,-1}}$. This leads the transition function to be defined by $P : \mathcal{S} \times \mathcal{U}^2 \to \Delta(\mathcal{S})$. The symmetry also induces that agents $a_{i,1}$ and $a_{i,-1}$ have a similar uncertainty about the state $\forall i$. Still, the observation function $O : \mathcal{S} \times I \times J \to \Delta(\mathcal{Z})$ maps a state to different observations for the same i. Common team rewards $\{r_t^1, r_t^{-1}\} = R(s_{t+1}, s_t, \boldsymbol{u_t}) : \mathcal{S}^2 \times \mathcal{U}^2 \to \mathbb{R}^2$ are assigned to agents after each time step. The goal of each agent $a_{i,j}$ to maximise its expected return $\mathbb{E}_{\boldsymbol{\pi},p,P}\left[\sum_{t=0}^{T-1} \gamma^t r_t^j\right]$ where the joint policy $\pi \in \mathcal{U}^2$. Finally, the symmetry allows the policy of agent $a_{i,1}$ to select the agent's actions $_{i,-1}$ $\forall i$.

If one team is constrained to have a stationary policy, the other team would be the only one learning, and the two-team Markov game can be considered a Dec-POMDP. In our experiments, teams are trained with value-based CTDE methods designed for Dec-POMDP. We chose QMIX because of its popularity, MAVEN because of its improved exploration capability, which outperforms QMIX in complex scenarios, and QVMix because it has proven competitive with these two as illustrated in Chapter 4.

7.3 COMPETITIVE STARCRAFT MULTI-AGENT CHALLENGE

To perform our experiments, we created a new environment by modifying SMAC to create a competitive environment. We adapted[1] both SMAC and the associated learning framework PyMARL [Samvelyan et al., 2019]. It is now possible to control both opposing teams in SMAC and train them simultaneously. In competitive SMAC, the goal remains

1 github.com/PaLeroy/competSmac - github.com/PaLeroy/competPymarl

unchanged, and it is to defeat the opposing team by inflicting sufficient damage to reduce the hit points of all opponents to zero. However, unlike in Chapter 4 experiments, the winning team is the one that ends up with the highest sum of remaining hit points at the end. It is a draw if both teams end up with equal hit points. We have chosen the $3m$ and $3s5z$ maps for our experiments but increased the time horizon. In the $3m$ map, two teams of three marines compete for a maximum of 100 time steps. In the $3s5z$ map, two teams of three stalkers and five zealots compete for a maximum of 150 time steps. Additionally, agents observe their relative position to the centre of the map. Section 3.3.1 provides more details about these two maps and SMAC.

7.4 LEARNING SCENARIOS AND PERFORMANCES CRITERIA

We aim to train a team to be resilient to different adversarial strategies. We test three different learning scenarios differentiated by the diversity of opponents' strategies encountered during training. The trained teams can act as any of the two teams in our environments thanks to the environment symmetry. In the first learning scenario, the team is trained against a stationary strategy, which we refer to as a heuristic. As explained in Section 7.2, such configuration is a Dec-POMDP and is the framework for which CTDE methods are designed. The second learning scenario is self-play, in which the team is trained by playing against itself, thus facing a strategy that continuously improves at the same learning speed. In the third scenario, a population of several teams is trained using the same method. Teams either play against themselves or against other learning teams. Self-play is the particular case of a training population composed of a single team. In this study, the size of the training population is fixed to five to reduce computational complexity.

After training, we evaluate teams with the Elo rating system [Elo, 1978]. The Elo rating system aims to assign each population player a rating of R to rank them. From these ratings, one can compute the probability that a player will win when facing another. Let R_A and R_B be the ELO scores of players A and B, respectively. The probability that player A wins over player B is $E_A = \frac{10^{R_A/400}}{10^{R_A/400}+10^{R_B/400}}$ and the probability that B wins over A is $E_B = \frac{10^{R_B/400}}{10^{R_A/400}+10^{R_B/400}}$. One can see that $E_A + E_B = 1$. The number 400 can be considered as a parameter. It determines that if player A's Elo score is 400 points above that of B, it has a ten-times greater chance of defeating B. After a game, player A's rating is updated based on its score S_A, equal to 1 for a win, 0 for a loss, and 0.5 for a draw. The updated score is $R'_A = R_A + cst * (S_A - E_A)$ where cst is a constant that defines the maximum possible update of the Elo score. Typically, cst is 32, but for our experiments, we set it to 10 to decrease the amplitude of oscillations in the Elo score during tests.

We form test populations to compute Elo scores in different configurations. To evaluate only the learning scenarios, we formed one test population for each CTDE method, with teams trained with all learning scenarios. To evaluate the heuristic's performance, we add

it to the previously defined test populations to form new ones. To find the best learning scenario/training method pair, we group all trained teams in two test populations, with and without the heuristic. To evaluate training efficiency and heuristic performances, each trained team is tested along training against the heuristic and against teams trained with other learning scenarios but using the same CTDE method.

7.5 EXPERIMENTS

For our experiments, teams were trained in two different environments using three learning scenarios and three CTDE methods. We conducted the same experimental process for both environments. Each method's learning scenario was executed ten times and stopped after each team had exploited 10^7 samples. For each method, this resulted in ten teams trained against the heuristic requiring 10^7 environment time steps, ten teams trained in self-play requiring 5×10^6 environment time steps and ten training populations of five teams leading to 50 teams trained within populations requiring at least 5×10^7 environment time steps. Therefore, 210 teams of nine types were trained in both environments.

To train teams against the heuristic, it was ensured that they play the same number of episodes, acting as team $j = 1$ and team $j = -1$. Concerning training within a population, each team had an equal chance of playing as either team and an equal chance of playing against any team in the population, including itself. Network architectures and parameters are identical for all learning scenarios and methods to ensure fair comparison. The default configurations provided by their different authors [Rashid et al., 2018; Mahajan et al., 2019; Leroy et al., 2021] determined learning parameters. However, the epsilon anneal time is set to 2 million instead of 0.5 million, and the networks are updated every eight episodes in $3m$ and every episode in $3s5z$. As in the literature, individual networks of each team share the same parameters to improve learning speed. Appendix B.1 provides more details about the training parameters, and we present training times in Appendix B.2.

The heuristic policy is based on two rules. It moves toward the starting point of the opponent's team until it reaches the opposite side of the map and stops. If enemies are within shooting range, the agent attacks the nearest one. This heuristic slightly differs from StarCraft's built-in AI, which is being used to train teams to cooperate in SMAC. The built-in AI agents also move toward the other side but select targets based on a priority score. They will choose to attack the closest unit with the highest priority, which will remain the target until its priority drops or it can no longer be attacked. A unit's priority score is based on its type and current action. For example, if two of the same units attack and the targeted unit stops attacking, its priority score will drop, and a built-in AI agent will select the other unit to attack. This is the main difference from our heuristic because agents will attack the nearest unit regardless of its action and priority. Results show that our heuristic is more complex to beat than the one provided in SMAC. Finally, while both maps are denoted as easy in [Samvelyan et al., 2019], the task of

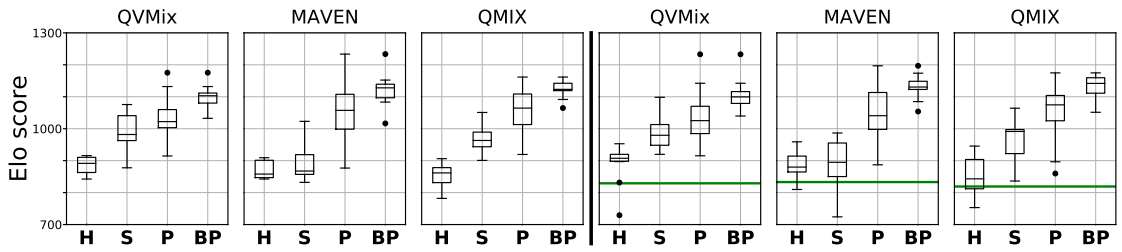
learning everything from scratch is not. When learning against the heuristic, teams do not need to learn how to find their opponents because they automatically move towards them. When they learn through self-play or within a population, they first need to learn where to find opponents before they can face them. This increased complexity compared to the original SMAC led us to choose these maps, motivated by a compromise between computational complexity and learning complexity.

As described in Section 7.4, we form test populations to evaluate teams with the Elo rating system. For both environments, there are three test populations of 70 teams trained with the same method but the three learning scenarios and one test population with all 210 trained teams. Adding the heuristic creates four additional test populations from the previously defined ones. In practice, to compute the Elo scores in these 16 test populations, each team plays 20 games against all the other teams in a randomised order. Every team starts with an Elo score of 1000, and we set the maximum Elo score update to 10, which is sufficiently small for our population sizes. In Section 7.6, we analyse the distribution of Elo scores after every team had finished its testing games.

During training, team neural network parameters are recorded every 20000 time step until the 10 millionth played time step. The test populations are composed of agents whose network parameters correspond to those recorded at the 10 millionth time step. The other saved networks allow one to evaluate teams' performances during training. We evaluate teams trained with a method and a learning scenario against the heuristic and against all teams trained with the same method but a different learning scenario. We analysed the win rates along training time steps of these different matchups when teams play 24 games against each other. For example, for the same method, each of the ten teams trained in self-play played 24 games against all the 50 teams trained within a population and against the 10 teams trained against the heuristic.

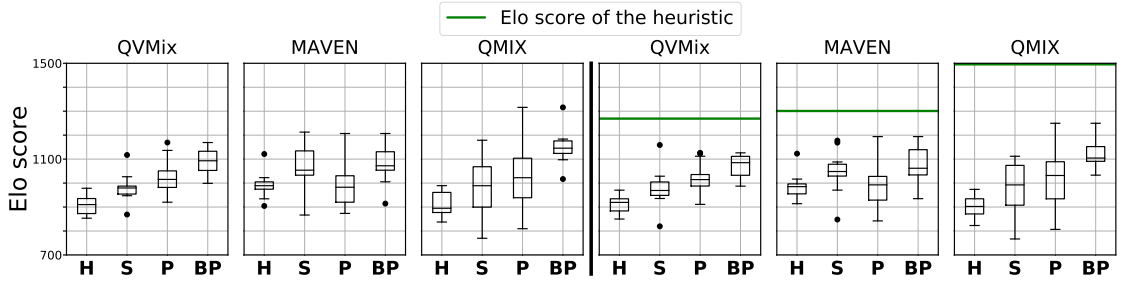
7.6 RESULTS

We present the Elo scores of teams with box plots in Figure 7.1 when the test population contains teams trained with the same method. We first focus on performances when the heuristic is not in the test population (Fig. 7.1a, 7.1c). The first observation is that the best teams are the ones trained within a population, except with MAVEN in $3s5z$ (Fig. 7.1c.2) for which the differences between learning scenarios are smaller. We discuss these differences later. The second observation is that the population scenario has the highest variance. To understand this, we also plot the box plots corresponding to the ten teams that achieved the highest Elo score of each training population, denoted by **BP**. These box plots confirm a difference between teams of the same training population, and a selection must be made to find the best one to optimise the performance of this learning scenario. Training against the heuristic is the worst scenario, arguably because agents do not generalise to other strategies than the heuristic. However, the heuristic is not in these test populations, and its impact is a concern for later analysis. The performance



(a.1) (a.2) (a.3) (b.1) (b.2) (b.3)

(a) $3m$ map **without** heuristic. (b) $3m$ map **with** heuristic.



(c.1) (c.2) (c.3) (d.1) (d.2) (d.3)

(c) $3s5z$ map **without** heuristic. (d) $3s5z$ map **with** heuristic.

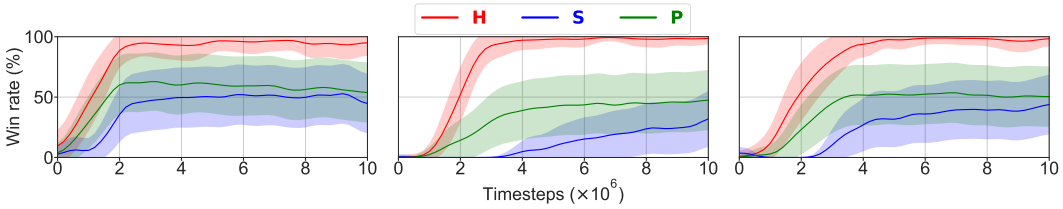
Figure 7.1: Elo score box plots of 12 test populations. Half of the experiments were performed in the $3m$ map, shown at the top (7.1a, 7.1b) and the other half in the $3s5z$ map, shown at the bottom (7.1c, 7.1c). In each test population, teams are trained with the same method, which is either QVMix (7.1a.1, 7.1b.1,7.1c.1, 7.1d.1), MAVEN (7.1a.2, 7.1b.2,7.1c.2, 7.1d.2) or QMIX (7.1a.3, 7.1b.3,7.1c.3, 7.1d.3). In 7.1b and 7.1d, the heuristic is present in the test population and a green line represents its Elo score. Box plots represent the distribution of the ELO scores of the teams trained either against the heuristic (**H**), in self-play (**S**), within a population (**P**) or the best of each training population (**BP**). For most methods, teams trained within a population achieved the highest Elo scores. Box plots present the median, the first quantile ($Q1$) and the third quantile ($Q3$). The reach of whiskers is defined by $1.7 * (Q3 - Q1)$.

of teams trained in self-play lies between the two other learning scenarios. While some achieve Elo scores close to those of the best teams in each training population, the scores of others are lower than the lowest scores of teams trained within the population.

The same experiment is performed by adding the heuristic in the three test populations, and corresponding box plots are presented in Figures 7.1b and 7.1d. The heuristic scores are different depending on the map. In the $3m$ map, most teams achieve a higher Elo score than the heuristic, whereas in $3s5z$, the heuristic dominates all teams. Adding the heuristic in the test populations slightly decreased Elo scores in the $3s5z$ map for all three learning scenarios. The conclusion is straightforward and the heuristic is better than all teams. However, one should note that the score ordering between the teams remained the same between Figure 7.1c and 7.1d. In $3m$, one can see that the Elo score of teams trained against the heuristic (**H**) is higher in Fig. 7.1b than the ones in Figure 7.1a, as a direct consequence of the introduction into the test populations of a team against which they win. When compared with the previous test populations (Fig. 7.1a), teams trained with QVMix achieved higher Elo scores than without the heuristic in the test population (Fig. 7.1b.1). In contrast, when trained with MAVEN and QMIX, they achieved lower Elo scores (Fig. 7.1b.2, 7.1b.3). In all cases, the higher values of box plots are not significantly different, but lower values are, meaning that some teams performed poorly against the heuristic. The conclusion is that teams trained within a population remain the most successful in most of our experiments, regardless of whether the heuristic is the best or almost the worst team.

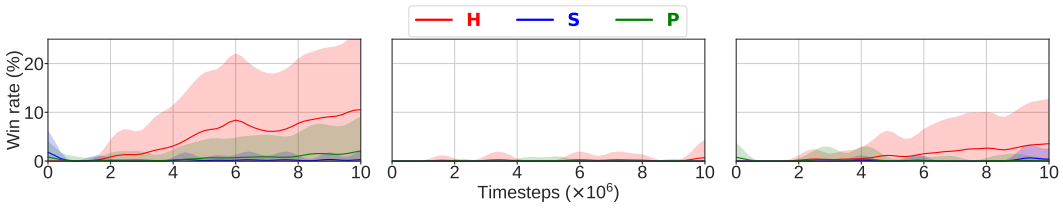
In Figure 7.2a, we present the evolution of win rates against the heuristic along training time steps by each team in the $3m$ map. For all methods, teams trained against the heuristic are the best against it on average. This explains why their Elo scores improve when the heuristic is included in the test population. This is also the case for QVMix teams trained in self-play and within a population that performed better and learned faster than teams trained with MAVEN and QMIX with these two learning scenarios. However, in the $3s5z$ map, it can be seen in Figure 7.2b that the win rates against the heuristic are very low, not to say equal to zero. Only the win rates of teams trained against the heuristic, especially with QVMix and QMIX, increase at the end of the training but with a high variance in comparison to the $3m$ map (Fig. 7.2a). For QVMix, the win rates of teams trained with a population also increase at the end of the training phase. The time we have budgeted for training in the $3s5z$ map may be insufficient to achieve a high win rate. However, we find this beneficial because it shows that, even when the heuristic is better than all teams, training against it, with the same training time steps allowance, is not the best learning scenario when teams have to be good against several strategies. This also shows that our heuristic is more complex to defeat when comparing the results of [Rashid et al., 2018; Mahajan et al., 2019; Leroy et al., 2021] where better results are observed in these maps with the former SMAC heuristic.

In the $3m$ map, the standard deviation of green and blue win rates, representing population and self-play learning scenarios, respectively, is high. This confirms the results



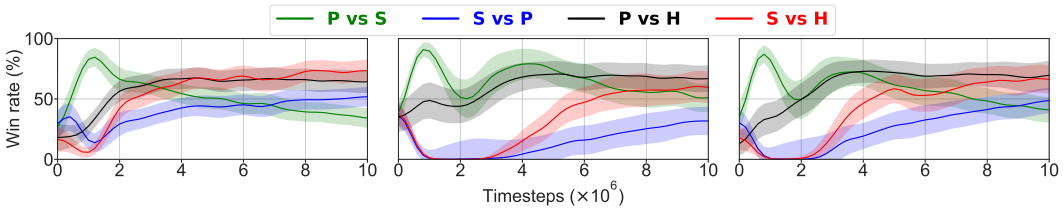
(a.1) QVMix (a.2) MAVEN (a.3) QMIX

(a) Win rates achieved against the heuristic in the $3m$ map.



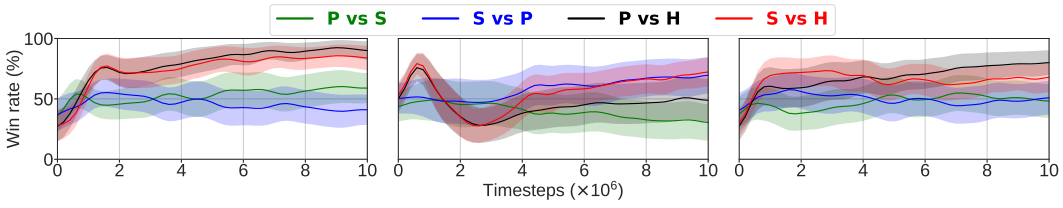
(b.1) QVMix (b.2) MAVEN (b.3) QMIX

(b) Win rates achieved against the heuristic in the $3s5z$ map. Note the change in scale of the y-axis.



(c.1) QVMix (c.2) MAVEN (c.3) QMIX

(c) Win rates achieved by teams trained in the $3m$ map against themselves.



(d.1) QVMix (d.2) MAVEN (d.3) QMIX

(d) Win rates achieved by teams trained in the $3s5z$ map against themselves.

Figure 7.2: Means of win rate achieved along training time steps by confronting teams trained with the same method against the heuristic or against other teams trained with a different learning scenario. Teams are trained either with QVMix, MAVEN, or QMIX. Tests were performed in the $3m$ map, shown at the top, and in the $3s5z$ maps, shown at the bottom. Win rates against the heuristic are presented in red, blue and green for teams trained against the heuristic, in self-play and within a population, respectively. Win rates of teams trained within a population against teams trained in self-play are presented in green and against teams trained against the heuristic in black. Win rates of teams trained in self-play against teams trained within a population are presented in blue and against teams trained against the heuristic in red. The error band is half the standard deviation.

of Elo score box plots, which show performance gaps between teams trained within the same learning scenario. Observations also suggest that if trained for longer, teams trained in self-play would achieve the same win rates as teams trained within a population, suggesting a difference in training sample efficiency. As teams first need to learn to cross the map to meet opponents and fight them, this difference is maybe because training within a population against several strategies increases the probability of creating episodes where two opposing agents meet and fight at the beginning of training.

Figures 7.2c and 7.2d show win rates from the confrontations between trained teams. The draw rates can be obtained by subtracting from 1 the sum of these two curves. This confirms the previous results with box plots that training against the heuristic is the worst scenario, as an average win rate above 60% is achieved against them, as shown in the black and red curves. Green and blue curves enable one to analyse self-play against population-learning scenarios. At the beginning of the training phase, teams trained within a population are better than self-play in the $3m$ map. This high win rate decreases with training in favour of the win rate of teams trained in self-play for QVMix and QMIX until it becomes higher than the latter. Again, this suggests a difference in training sample efficiency. Moreover, although the training sample efficiency is lower for teams trained in self-play, the number of environment time steps required to train them is five times lower in our setting. However, it is on average that the self-play teams become better. In the $3s5z$ map, this overlap phenomenon does not occur, and the average win rates fluctuate around 50% with the green curves remaining just above the blue ones at the end. The proximity of performances between teams trained in self-play and within a population is arguably due to the environment and the $3m$ map, which does not offer the possibility of winning with very different strategies. As the $3s5z$ map appears to be more complex, with the lack of performances against the heuristic as evidence, teams trained within a population remain better on average.

On average, the results of the teams trained within a population with MAVEN in the $3s5z$ map differ from the other experiments. Some results are slightly worse than those of the other experiments. The reason for this is unsure, as we executed the experiments several times, but this does not affect the conclusions of our experiments that remain clear. Finally, it would be necessary to repeat these experiments more times or to analyse the agents' behaviour in depth to find the problem, which is beyond the scope of the study.

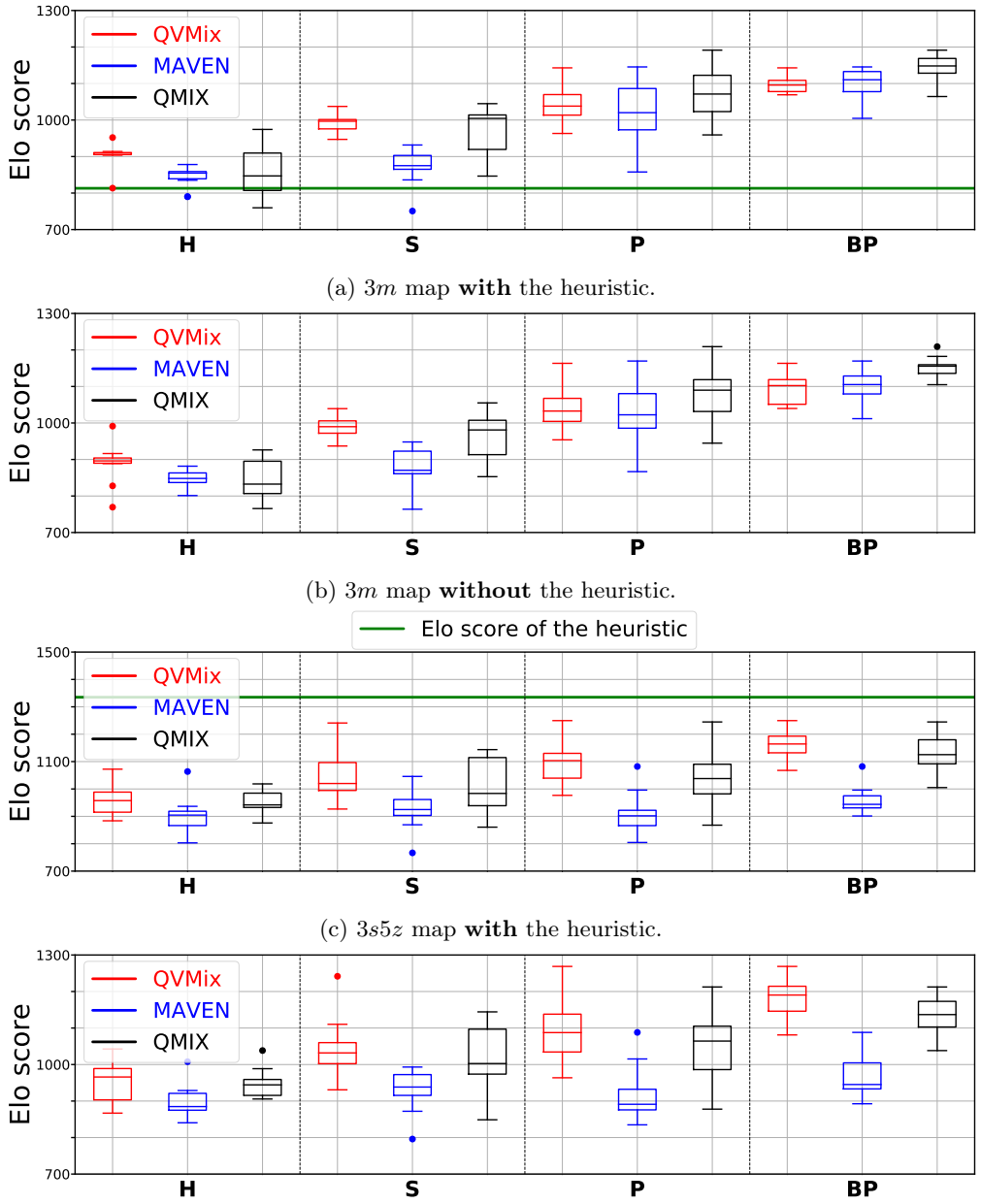
In Figure 7.3, we present the box plots of Elo scores obtained when grouping all trained teams in a single test population for both maps, with and without the heuristic. The learning scenario ranking remains the same as in other experiments. In the $3m$ map, QMIX achieves the highest Elo scores, while the lowest MAVEN Elo scores are worse than the ones of QMIX and QVMix when teams are trained in self-play or within a population. QVMix produces results with a lower variance than QMIX. In the $3s5z$ map, QVMix achieves the highest Elo scores, and MAVEN achieves the lowest ones. The same experiment without the heuristic led to the same conclusion. Despite its exploration mechanism, MAVEN cannot outperform QMIX and QVMix. This is also the case in

[Mahajan et al., 2019] and [Leroy et al., 2021], where they show that MAVEN outperforms QMIX but not QVMix in more complex Dec-POMDP environments.

7.7 DISCUSSION AND FUTURE WORK

This study evaluated learning scenarios to train teams to face multiple strategies in a symmetric two-team Markov game. Teams are trained with three value-based CTDE methods, QMIX, MAVEN, and QVMix, and with three learning scenarios differentiated by the variety of strategies they will encounter during their training. Specifically, they are trained by playing against a stationary strategy, against themselves, or within a population of teams trained using the same method. To perform our experiments, we modified SMAC's cooperative environment to allow teams to compete and train simultaneously and trained teams in two different SMAC environments. The Elo rating system evaluates these nine types of trained teams at the end of their training. Different groups are formed to identify the best learning scenario and the best learning scenario/training method pair. We also analysed the win rates of several matchups during training to support the results provided by the Elo scores. Our results showed that the best learning scenario is to train teams within a population of learning teams when each team plays the same number of time steps for training purposes. We reached this conclusion irrespective of whether or not the stationary strategy was better than all trained teams. Finally, a selection procedure is required because teams from the same training population do not perform equally.

This work is one of the first investigations of two-team competition with CTDE methods, and we hereafter suggest several future research directions. First, we suggest performing the same experiments on more complex environments and tackling the challenges of asymmetric settings. In this paper, we selected value-based methods because of their performances in SMAC [Samvelyan et al., 2019] at the time of our experiments. Recent CTDE methods overcome these performances and may lead to interesting new results. Typically, stochastic policies trained with policy-based methods might provide better results in our setting. Another research direction would be to study how diversity in the training population impacts performance. Diversity could be increased by adding the heuristic in the population, confronting agents against older learned policies, varying the size of the population or training teams with different methods in the same population. In addition, as described in Chapter 6, methods that compute the best responses, exploit PSRO or model the opponent would be interesting to test in such an environment. We also propose performing behavioural and policy analyses to understand why some teams achieve a better Elo score and how strategies differ in a single training population.



(a) $3m$ map **with** the heuristic.

(b) $3m$ map **without** the heuristic.

(c) $3s5z$ map **with** the heuristic.

(d) $3s5z$ map **without** the heuristic.

Figure 7.3: Elo score box plots of four test populations, in $3m$ at the top and $3s5z$ at the bottom, composed of teams trained using three methods and three learning scenarios, with and without the heuristic. The training method is either QVMix (red), MAVEN (blue) or QMIX (black). Box plots represent the distribution of the ELO scores of teams trained either against the heuristic (**H**), in self-play (**S**), within a population (**P**) or the best of each population (**BP**). Box plots present the median, the first quantile ($Q1$) and the third quantile ($Q3$). The reach of whiskers is defined by $1.7 * (Q3 - Q1)$.

Part IV

CONCLUSION

8

CONCLUSION

This thesis showcases the usefulness of considering other learning agents when training one with reinforcement learning in the general case of MARL through three specific contributions. The first contribution extends the Deep-Quality-Value single-agent family of algorithms to the cooperative multi-agent setting. The second contribution demonstrates the interest of these cooperative methods in the context of a real-world application: the infrastructure management planning problem. The third contribution analyses how to train a team in a mixed cooperative-competitive setting where two teams compete. We hereafter summarise the manuscript's content, divided into three parts, and discuss the scientific findings before finally discussing the potential societal impact of MARL.

Part I provides the required background. Specifically, Chapter 2 presents the stochastic game, a general framework for multi-agent reinforcement learning. This framework can be adapted to specific settings distinguished by the agent's objectives: cooperation, competition, or general-sum. In addition, multi-agent reinforcement learning has its foundation in single-agent reinforcement learning, and we offer a concise introduction to its basics. Specifically, we discuss model-based and dynamic programming. We detail value-based and policy-based methods for model-free RL. We conclude by addressing the partial observability of agents and the consequences in MARL.

Part II tackles the cooperative setting. When agents cooperate, it can be framed as a decentralised partially observable Markov decision process where agents share the same reward. This part comprises three chapters. One is a background chapter, and the others address two contributions.

Chapter 3 defines the cooperative framework and details certain applications. Several cooperative MARL methods to solve Dec-POMDP are defined following its definition, emphasising how algorithms have been designed to allow centralised training with decentralised execution. While the performance of these methods is compared in later chapters, this third chapter concludes with a literature discussion, highlighting approaches not considered in this manuscript but still of interest.

Chapter 4 presents QVMix and other value-based methods for Dec-POMDP. It highlights that learning the state value function to update the state-action value function as done in SARL by the DQV algorithms can be extended to cooperative CTDE methods in MARL. Some of the introduced methods outperform the ones compared against, previously defined in Chapter 3. Moreover, they allow one to not overestimate the state-action value function, unlike these compared value-based methods relying on the classical Q-Learning update.

Chapter 5 presents infrastructure management planning, a real-world application that can be framed as a Dec-POMDP. Managing infrastructure by timely inspecting or repairing system components to minimise failure risk and maintenance cost becomes challenging as the system grows. Decentralising the decision, making each component an independent agent, and training them with CTDE methods allows for scaling to large systems. Framing it as a multi-agent problem allows it to scale but also to outperform rule-based approaches, considered the state-of-the-art for solving these problems. However, this work also highlights potential limitations when the number of agents increases, especially regarding the variance each method provides.

Part III tackles a setting where two teams compete. This mixed cooperative-competitive setting benefits from the cooperative methods and the training scenarios required to train an agent in a competitive environment. This part is divided into two chapters. The former provides background on the competition setting, while the latter presents a specific contribution in this two-team framework.

Chapter 6 addresses challenges when agents do not purely cooperate. This is the case not only in the competitive setting but also in the general-sum one. This chapter presents different types of solutions provided by game theory and discusses historical methods for computing some of these solutions, from dynamic programming to planning methods. The chapter concludes with the presentation of self-play and population training, two popular training scenarios for training agents in competitive or general-sum environments.

Chapter 7 presents an empirical study of learning scenarios to train one team of agents to compete against another. The setting is a symmetric two-team Markov game where two teams of the same agents compete. We compared three learning scenarios in this setting. We trained teams against a stationary policy, in self-play or within a population of learning teams. The performance of these learning scenarios is evaluated with the Elo score computed within different subsets of trained teams. This allows one to evaluate the resilience level against various policies. This leads to the conclusion that the population-based scenario is the best in this setting.

MARL is still a novel field, although it resides on strong and long-established foundations based on game theory and reinforcement learning. The precise moment when advances in neural networks significantly influenced MARL's progress is debatable, given that groundbreaking work in this area has only emerged recently. For the cooperative setting, it is difficult to provide one, but VDN [Sunehag et al., 2018], MADDPG Lowe et al. [2017] and COMA [Foerster et al., 2018b] can be considered as such works, along with SMAC [Samvelyan et al., 2019] which is the starting point of many experiments. We can also cite AlphaGo Zero [Silver et al., 2017], one of the popular breakthroughs in the competitive setting, despite being considered as planning more than RL by some.

However, despite these advancements, many challenges remain, particularly regarding the application of MARL in real-world scenarios. The community is bridging the gap with environments closer to reality. One notable example of bridging this gap is IMP-MARL. However, convincing applications of MARL that entirely replace classical planning ap-

proaches have yet to emerge. While the classical problems of deploying RL in real life, such as safety, robustness, or explainability of the trained agents, explain this, some effort is still required to bring MARL into real life by demonstrating their potential.

The number of agents studied in the literature is often very limited, and increasing this number should be considered in the future, in addition to getting closer to real-world applications. Mean-field games, which aim to study systems with an infinite number of agents, may be such candidates. Moreover, it is worth noting that the general-sum setting is underrepresented in this manuscript while being the most challenging. Despite existing work in this area, presented in Chapter 6, real-life general-sum settings, such as autonomous driving, remain particularly challenging due to the complex interactions between agents. We believe the reader can now understand why.

Finally, MARL has the potential to significantly impact society in various ways. Decision-making is a fundamental aspect of human cognition, and throughout history, scientific advancements have played a crucial role in aiding humans in making decisions. In this manuscript, we discuss numerous impactful applications of MARL. Consider, for instance, the profound impact of improving infrastructure management planning strategies. In anticipation of environmental changes stemming from climate change, it is not merely about saving costs in monetary terms but also minimising environmental damage, thereby highlighting the broader implications of decision-making. Moreover, the challenges of autonomous driving also highlight the complexities that MARL can address. From navigating crowded streets to ensuring passenger safety, MARL may offer solutions in this domain. Note that traffic management, controlling traffic lights, is also a well-studied problem framed as a MARL one. Additionally, some contributions leading to the development of this manuscript originated from a project involving defence use cases. While remaining neutral on value judgments, it's evident that the future of defence systems will likely incorporate autonomous decision-making processes facilitated by MARL techniques. However, a disclaimer regarding potential ethical concerns or unintended consequences of these advancements is essential.

We denoted three applications where machine learning, whether as MARL or not, will profoundly impact humanity's future. And there are many more. As machine learning agents are increasingly deployed, studying how they interact is of greater interest. Whether we like it or not, AI is changing the landscape of our lives. Nowadays, the world's concerns are divided between ecological sustainability and international armed conflicts. In such times, efforts should be made to measure the direction of AI improvements and, maybe, to revolutionise the world in the right direction. We leave it to the reader to imagine their preferred direction.

Part V

APPENDIX

A

IMP-MARL SUPPLEMENTARY MATERIALS

A.1 REPOSITORY, LICENSE, DATA, AND DOCUMENTATION

All defined IMP environments are integrated with well-known MARL ecosystems, i.e., Gym [Brockman et al., 2016], Gymnasium [Towers et al., 2023], PettingZoo [Terry et al., 2021] and PyMarl [Samvelyan et al., 2019], through wrappers. The tested MARL methods are adopted from PyMarl's library, but other libraries are also compatible with our wrappers, e.g., RLlib [Liang et al., 2018], CleanRL [Huang et al., 2022], MARLlib [Hu et al., 2022], or TorchRL [Bou et al., 2024]. All developments are available on a public GitHub repository, `https://github.com/moratodpg/imp_marl`, featuring an open-source Apache v2 license.

To reproduce the work reported in this paper, the following process can be executed: (i) cloning the repository, (ii) installing a virtual environment with the package requirements, and (iii) executing the script instructions of the corresponding method and IMP-MARL environments. We provide instructions for reproducing our results and tutorials for adding new environments or wrappers. Our code encodes models in Numpy files, stored in the repository folder `pomdp_models`.

We also provide the data files resulting from our experiments, enabling the reproduction of any reported result without re-running the experiments and the corresponding implementation code, hence facilitating future cross-comparisons. The readily available results include Figures 5.4, A.7, and A.8, along with Tables A.5, A.6, A.7. Configuration, execution, and results files are permanently stored at Zenodo, accessible via `https://zenodo.org/record/8032339`. Additionally, the controller networks' weights of the best policies presented in Figure 5.4 are also stored there, thus fostering further interpretability studies of MARL-based strategies. The dataset is open-access and registered with the Digital Object Identifier (DOI) 10.5281/zenodo.8032339. More information and dedicated tutorials can be found on the repository.

A.2 IMPLEMENTED OPTIONS

In IMP-MARL, the environments can be easily set up with specific options, from the definition of the number of agents to the observation information perceived by the agents and the state information received by mixers/critics. This can be straightforwardly specified through the configuration files provided on IMP-MARL's GitHub repository. These options are parameters included in IMP-MARL's classes. We refer the reader to the Appendix of the original paper from Leroy et al. [2023] to know more.

Parameter name	Parameters value	Parameter name	Parameters value
γ	0.95	Time max	2,050,000
Target update	200	RMS epsilon	10^{-5}
RMS alpha	0.99	Grad norm clip	10
Learning rate	0.0005	Obs last action	True
Agent network	[] - 64 GRU - []	Save model interval	20,000

Table A.1: Parameters set in our experiments.

A.3 EXPERIMENTAL DETAILS

A.3.1 *Description of the parameters set up in the experiments*

This section provides the information required to reproduce the results reported in this paper. Since the neural networks are trained via MARL using PyMarl's [Samvelyan et al., 2019] library, the parameters follow PyMarl's convention. However, their purpose can be easily deduced from the names themselves. The experimental parameters set up equal across all experiments are presented in Table A.1, while the parameters specific to each method are hereafter detailed. Note that in Table A.1, some parameters are not used by all methods, e.g., RMS parameters. Besides, target update intervals, buffer size, and batch size are specified based on the number of episodes. When the number of agents increases, we only augment the number of trainable parameters of the mixer/critic networks, while the actor networks and other parameters are not modified. The configuration files, available on the GitHub repository, contain all the parameters and can be used to launch any of the experiments conducted in this paper.

The agent network representation for CTDE methods and IQL consists of one GRU layer with a hidden state with 64 features. This means that the input is fully connected to the 64 hidden states, which are then fully connected to the outputs, one per action. We represent it as "[] - 64 GRU - []". Since the number of actions is $3n$, and hence the number of outputs of DQN network, a network with more representation capacity is needed. In that case, linear layers, whose number of output features are specified between brackets, are also included in the agent network surrounding the GRU layer. With $n = 2$ or $n = 3$ agents, the network is set as "[128] - 128 GRU - [128,64]" while for $n = 4$ or $n = 5$ agents, it is set as "[256] - 256 GRU - [256,256]". Note that the linear layers before the GRUs include a Relu activation function, and the last action taken is also added to the agents' observation as an additional input.

As mentioned previously, some parameters are the same for almost all methods. For instance, the optimiser selected is RMSProp for all methods except for FACMAC, which is trained with ADAM. In nearly all methods, the buffer stores the latest $2,000$ episodes,

	Epsilon start	Epsilon finish	Epsilon anneal time
Value-base methods	0.5	.01	5000
FACMAC	0.3	0.005	50000

Table A.2: Exploration parameters.

and 64 episodes are sampled at each episode to update the network. However, since COMA is an on-policy method, the networks are updated with the episodes just played. Therefore, in our experiments, COMA's networks are updated every four episodes based on these last experienced ones. This update is performed four times to ensure a fair number of network updates compared to the other methods, which are updated every episode.

During training, the value-based methods rely on an epsilon-greedy policy, whose parameters are specified in Table A.2, while they act following a greedy policy at testing. Note, however, that COMA and FACMAC utilise different training policies. FACMAC samples discrete actions through a Gumbel softmax for its actor, whereas exploration is performed via an epsilon, whose values are specified in Table A.2. On the other hand, COMA follows a classic stochastic policy during training. At the testing stage, FACMAC and COMA select actions adhering to a greedy approach, selecting the action associated with the maximum probability.

In the conducted experiments, the parameters set up in the environments differ with the number of agents, and we distinguish three cases: (i) $n <= 10$, (ii) $n = 50$, and (iii) $n = 100$. Starting with QMIX, the parameters are all the same when increasing n, except for the architecture of the mixing network, whose embedding size in the middle of the mixer is (i) 32, (ii) 64, and (iii) 128. QMIX relies on a double-Q feature, i.e., the loss computed to update θ differs from the original. While in the original loss function, the target Q value used for the update is selected with the action that maximises the target Q value parameterised by θ', in double-Q, the action is the one that maximises the Q value parameterised by θ. Therefore, we have: $\mathcal{L}(\theta) = \mathbb{E}_{\langle . \rangle \sim B}\big[\big(r_t + \gamma Q(s_{t+1}, u*; \theta') - Q(s_t, u_t; \theta)\big)^2\big]$ where $u* = \arg\max_u Q(s_{t+1}, u; \theta)$. This target network is updated every 200 episode. QVMix is a variant of QMIX, and the parameter values are similarly specified. In particular, QVMix contains two networks: (i) a Q network with the same architecture as QMIX and (ii) a V network with a copy of the Q network but with only one output. As for QPLEX, we try to be close to the parameters selected for the SMAC experiments they conduct in their paper. When n increases, we change only the size of the attention layer, which is composed of a layers and b heads. In particular, we set up (i) $1L4H$, (ii) $2L4H$, and (iii) $2L10H$. The mixer embedding is 64 for all experiments.

Concerning COMA, an actor-critic method, a few specific parameters should be additionally set up. The agent network, denoted as the actor, is the same as the other previously described. However, the critic varies with n because the input of the critic

Method	Network	$n = 3$	$n = 5$	$n = 10$	$n = 50$	$n = 100$
QMIX	Agent	27,587	27,715	28,035	30,595	33,795
	Mixer	18,657	41,249	133,569	5,430,657	42,202,113
QVMix	Agent	27,587	27,715	28,035	30,595	33,795
	Mixer	64,771	110,083	295,043	10,891,779	84,437,891
QPLEX	Agent	27,587	27,715	28,035	30,595	33,795
	Mixer	58,249	83,289	155,269	1,830,933	4,901,797
COMA	Agent	27,587	27,715	28,035	30,595	33,795
	Critic	35,971	45,955	70,915	1,195,907	10,840,323
FACMAC	Agent	27,587	27,715	28,035	30,595	33,795
	Critic	48,002	72,706	134,466	1,042,370	3,313,218
IQL	Agent	27,587	27,715	28,035	30,595	33,795
	/	0	0	0	0	0
DQN	Agent	157,979	758,003	/	/	/
	/	0	0	/	/	/

Table A.3: Number of trainable parameters in uncorrelated k-out-of-n systems.

becomes relatively large, and its architecture is entirely linear: (i) [128, 128], (ii) [512, 256, 128, 128], and (iii) [2048, 1024, 512, 256, 128]. A TD-λ used to update the critic is set to 0.8, and the learning rate to train the critic is the same as that of the actor, specified in Table A.1. Regarding FACMAC, the parameters of interest are those at the critic, as its size increases with the number of agents n. FACMAC features a 2-layer-mixing network and, therefore, we specify the size for both: (i) 64-64, (ii) 128-64, and (iii) 128-128. As in COMA, the TD-λ parameter is set to 0.8. The critic has a target network, similar to value-based methods, updated every 200 episodes with a soft target update with $\tau = 0.001$. Moreover, we followed the optimiser selection made in FACMAC's original paper and used ADAM, with an epsilon equal to 10^{-8}.

Regarding IQL, a decentralised method, the parameters are identical in all tested environments. IQL also has the double-Q feature activated and learns individual Q values independently via DQN's algorithm. For DQN, we only run experiments with less than five agents, i.e., $n <= 5$. The main difference between tested environments is the size of their networks. Since DQN features 3^n outputs, only 64 GRU cells are insufficient in network capacity. In our experiments, we increase the network size and confirm that DQN achieves similar results as the other MARL methods. In DQN, the network architecture is the only parameter adjusted with the number of agents n.

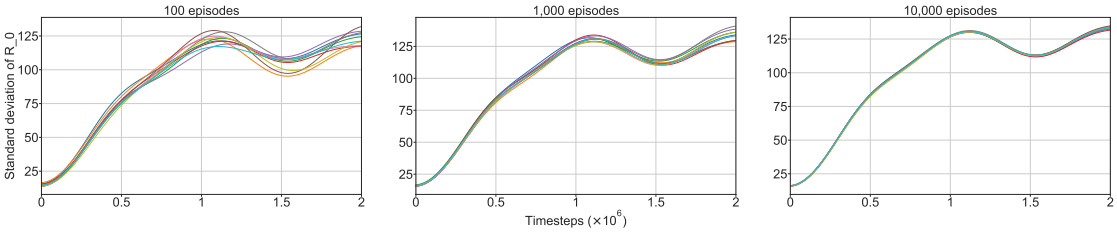


Figure A.1: Variance analysis of a given set of neural networks. We report the standard deviation of the sum of discounted rewards obtained during the test phase. Each curve represents an entire test experiment when executing 100, 1,000, or 10,000 test episodes.

Finally, we list in Table A.3 the number of trainable parameters of the networks. The input is slightly different between the tested sets of IMP-MARL environments, so we show only the parameters for one of them in the table. Note that there is an agent network taking action for all agents, and we purposely duplicate the agent network row to emphasise that all agent networks are identical across experiments.

A.3.2 *Statistical analysis of the variance associated with the number of test episodes*

As previously explained, we conduct 10,000 test episodes to reduce the variance related to the expected sum of discounted rewards within a given environment. The choice is motivated by the direct relationship between the variance associated with $\mathbb{E}[R_0]$ and the number of test episodes. In our experiments, we average over 10,000 policy realisations. Still, here, we show that the standard deviation associated with $\mathbb{E}[R_0]$ for each training time step can vary significantly if an insufficient number of test episodes is simulated. Figure A.1 illustrates the standard deviations observed when executing with 100, 1,000, and 10,000 test episodes. We produce this figure with the neural networks obtained with one FACMAC training run. These networks are trained over time in the offshore wind farm environment with 100 components. We execute ten times the 100, 1,000, and 10,000 test episodes from this single set of networks to observe how the variance evolves with this number of test episodes. We observe that the standard deviations of $\mathbb{E}[R_0]$ obtained with 100 test episodes significantly vary over the investigated test runs. Naturally, the standard deviation variation is reduced with an increasing number of test episodes, obtaining very similar standard deviations when testing with 10,000 episodes.

Moreover, one can also compute the largest difference that is observed between these 10 test runs at a specific time step, i.e., the absolute difference of the estimated $\mathbb{E}[R_0]$ at a given time step. When testing 100 episodes, the absolute difference is 231.89 at a time step where the maximum of the ten provided $\mathbb{E}[R_0]$ is -2786.1. If 1000 episodes are tested, the absolute difference is 99.8, where the maximum is -2757.2 at this time step, whereas by testing 10,000 episodes, the difference equals 24.8 with a maximum of -2622.8. These numbers represent only a trained set of networks over 1920 training runs,

Parameters	Train only on CPUs	Train on GPUs	Test only on CPUs	Test on GPUs
Number of CPU	4	2	8	5
RAM	5 Gb	6 Gb	5Gb	10 Gb

Table A.4: Hardware configurations for training and testing experiments.

and thus, a final conclusion cannot be claimed. Yet, it motivates the need to simulate 10,000 test episodes, as with only 100 test episodes, the absolute difference can reach 10 %.

A.3.3 *Hardware and experiments duration*

Our experiments run on different clusters managed by SLURM [Yoo et al., 2003]. They are executed with specific hardware requirements based on the number of agents: experiments with up to 10 agents are run on only CPUs, and we perform experiments on GPUs with 50 and 100 agents. The efficiency does not substantially improve when running experiments with less than 10 agents on GPUs because training a GRU layer requires forwarding the whole episode sequentially. In contrast, the computational time can be reduced when running experiments with 50 and 100 agents on GPUs because we train all agents as a single network, and the batch size increases with n. We categorise the computational time required for the reported experiments according to whether (i) the experiment is (or is not) run only on CPUs and (ii) the value reported corresponds to the training or the testing stage. In Table A.4, we additionally provide the hardware requirements demanded during the training and testing phases. Note that we benefit from more resources during testing because ten parallel environments run. Moreover, we intentionally demand more RAM to avoid problems. These reported RAM configurations are indicative and can be seen as requirements, yet not as exact memory usage numbers.

We represent the computational time required for the experiments during training in Figures A.2 and A.3 as well as during testing in Figures A.4 and A.5. To avoid overloading the figures, the markers do not explicitly indicate which experiment they correspond to, yet the experiments are all vertically grouped based on the method and the environment. For each abscissa, 20 experiments are represented with and without campaign cost. The first three sets of experiments represent QMIX in the uncorrelated k-out-of-n setting, followed by QMIX in the correlated k-out-of-n environment and QMIX in the offshore wind farm one. The methods are ordered as QMIX, QVMIX, QPLEX, COMA, FACMAC, IQL, and DQN, while the environments are ordered as k-out-of-n setting, correlated k-out-of-n, and offshore wind farm. It can be seen in Figures A.2 and A.4 that the two plots with $n < 10$ additionally represent the three additional experiments related to DQN.

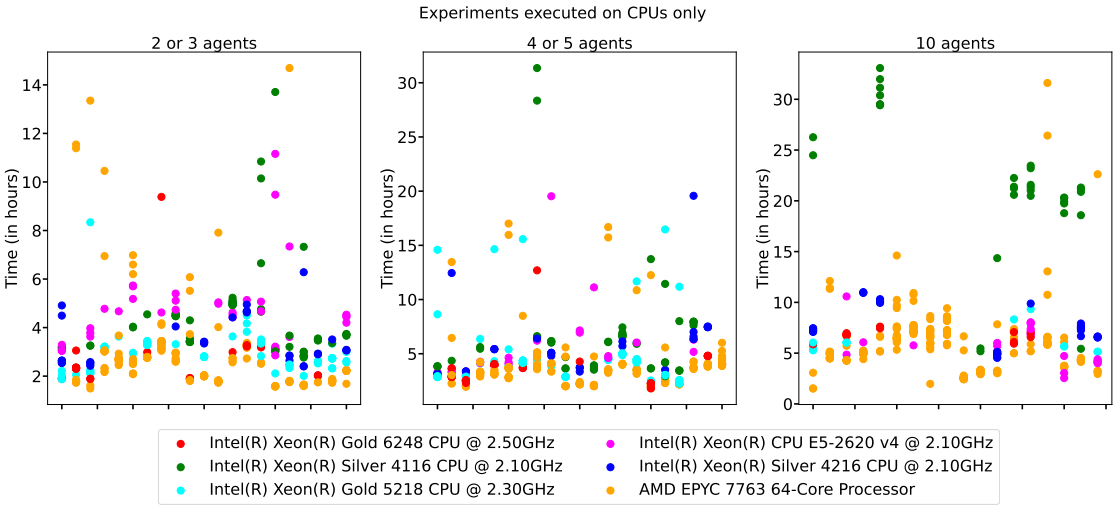


Figure A.2: Training duration for experiments with $n <= 10$ performed on only CPUs.

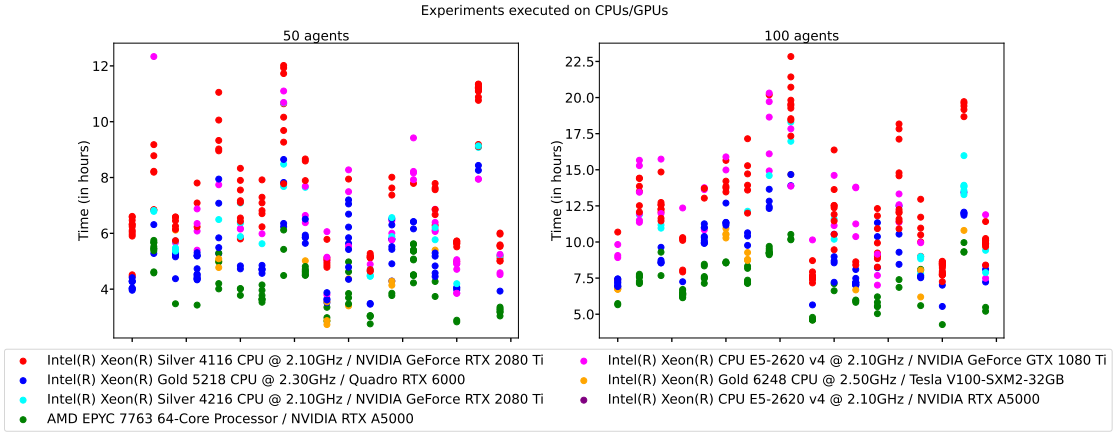


Figure A.3: Training duration for experiments with $n >= 50$ performed on CPUs and GPUs.

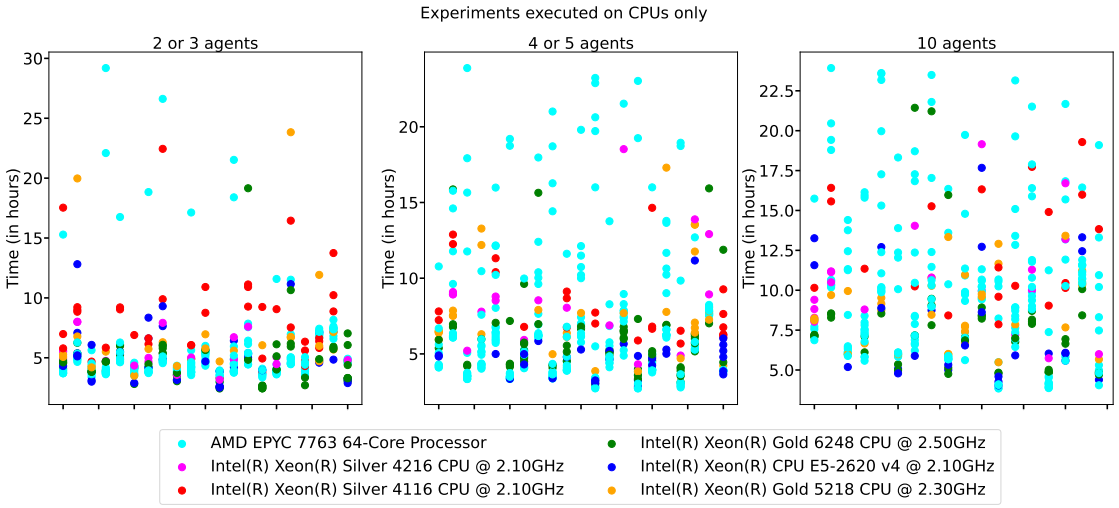


Figure A.4: Testing duration for experiments with $n <= 10$ performed on only CPUs.

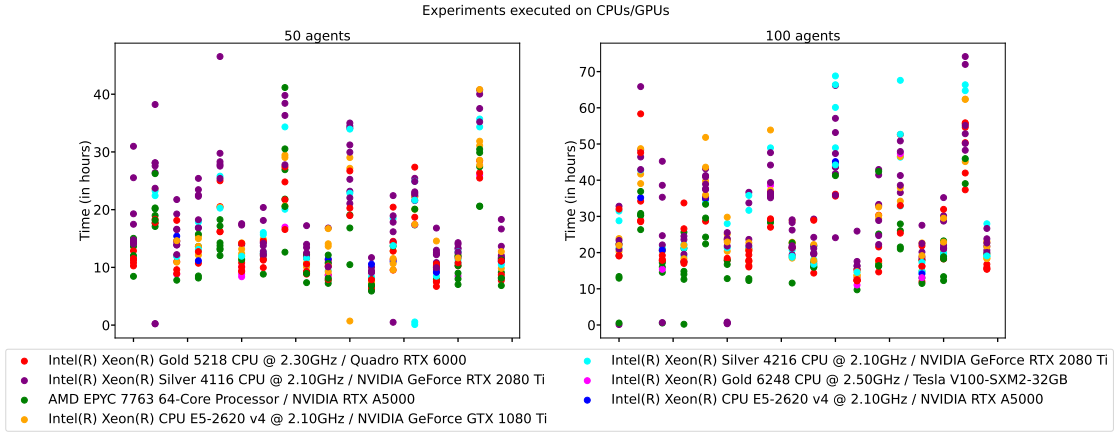


Figure A.5: Testing duration for experiments with $n >= 50$ performed on CPUs and GPUs.

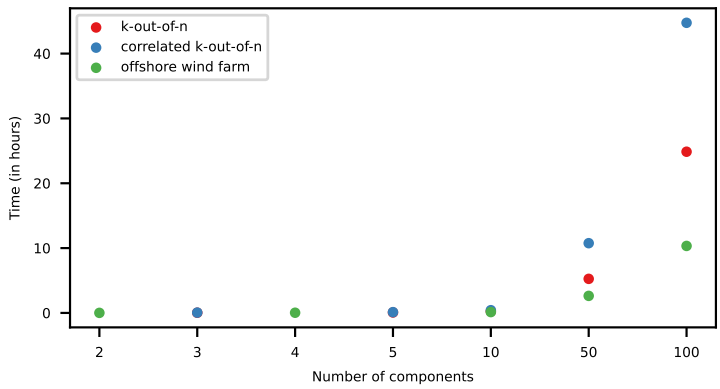


Figure A.6: Computational time required for executing expert-based heuristic policies as a function of the number of components. The experiments are run on 2 AMD EPYC Rome 7542 CPUs @ 2.9GHz.

The first observation that may be addressed is the resulting high variance. The variation across runs is logical because of the specific performance of the CPU/GPU models employed and the additional activity of clusters at the time of our experiments. Concerning the computational time required for training, testing episodes are not executed. We can see that, by running the experiments on only CPUs, we manage to train the agents in less than 10 hours, except for occasional outliers. For experiments with 50 agents relying on GPUs, the computational time is overall very similar to those previously mentioned, requiring less than 10 hours. Yet, a longer time is needed for those with 100 agents. The fastest training results correspond to COMA because we run four environments in parallel instead of one during training. IQL and QMIX follow closely, but due to their complex architectures, QVMix, QPLEX, and FACMAC require additional computational time. Naturally, the testing stage requires more time than training for all experiments because it executes $10,000$ test episodes per stored network. The correlated k-out-of-n environment requires slightly more time than the others because the correlation information is updated at every inspection step.

Furthermore, we represent in Figure A.6 the time required to compute expert-based heuristic policies. The experiments are plotted as a function of the number of components and coloured based on their corresponding environment. In this case, all experiments are run on CPUs. We can see that heuristic policies can be efficiently computed for environments with less than 50 components, yet the computational time significantly increases for experiments with 50 or 100 components. This result is logical since the combination of evaluated parameters includes the number of components to be inspected at each inspection interval. Besides, the overall computation time is directly influenced by the time needed to run an episode, with the k-out-of-n environments taking longer compared to the offshore wind farm ones because the episode's finite horizon spans over 10 additional time steps.

A.4 ADDITIONAL BENCHMARK RESULTS

This section presents additional results and remarks beyond those reported in the main text. First, we provide the values of the expected sum of discounted rewards achieved by the best runs over all experiments. We list the best policy for each conducted experiment in Tables A.5, A.6, and A.7. Note that the maximum values represented with markers at the right of each box in Figure 5.4 can be retrieved from these values by applying the normalisation (x-H)/H, with H being the value achieved by the heuristic policies.

Additionally, we represent the learning curves corresponding to all our experiments in Figures A.7 and A.8. The learning curves showcase the evolution of the expected sum of discounted rewards every 20,000 training time steps, computed at the testing stage with 10,000 test episodes. Since the training is conducted with 10 different seeds for each environment and method, we plot the corresponding 25th-75th percentiles around the median. These results confirm the variance observed between the best results and presented in Figure 5.4.

Based on Tables A.5 and A.6, one may additionally infer that correlated environments result in lower costs with respect to those uncorrelated. This is especially true for environments with n>=10 agents, specified without campaign costs, and in all environments with campaign costs. While MARL methods profit from the additionally provided correlation information, this is not always the case for the heuristic policies.

One final remark is that the discrepancy between the expert-based heuristic policy and MARL methods is more pronounced in offshore wind farm environments. This could be attributed to the shorter decision horizon or, the higher inspection cost in this case (see Table A.7).

n	QMIX	QVMix	QPLEX	COMA	FACMAC	IQL	DQN	Heuristics
3	-9.7	-9.8	-9.7	-10.6	-10.4	-35.3	-9.9	-12.5
5	-20.4	-20.7	-20.4	-21.8	-22.1	-108.7	-24.0	-25.2
10	-51.0	-51.5	-51.0	-54.3	-61.3	-404.5	/	-63.7
50	-229.7	-236.0	-212.8	-1190.6	-249.0	-1991.1	/	-268.1
100	-222.6	-230.7	-220.6	-1770.1	-225.7	-1770.1	/	-262.4
*3	-14.6	-14.7	-14.7	-15.0	-17.0	-35.3	-13.5	-15.1
*5	-27.4	-27.7	-27.4	-28.9	-33.0	-27.8	-26.6	-28.6
*10	-58.9	-63.0	-60.7	-70.0	-61.9	-404.5	/	-64.5
*50	-169.5	-173.9	-168.4	-241.4	-160.7	-623.3	/	-232.7
*100	-167.2	-175.8	-160.2	-1770.1	-144.8	-1770.1	/	-231.5

Table A.5: k-out-of-n system best policies (* = campaign cost).

n	QMIX	QVMix	QPLEX	COMA	FACMAC	IQL	DQN	Heuristics
3	-9.7	-9.7	-9.6	-11.0	-10.6	-10.0	-10.0	-13.0
5	-20.4	-20.6	-18.4	-21.2	-21.6	-20.2	-23.4	-28.1
10	-47.6	-51.0	-45.2	-49.7	-46.1	-374.5	/	-67.7
50	-214.3	-233.0	-212.3	-419.3	-143.4	-1339.9	/	-240.0
100	-250.3	-289.0	-276.8	-486.9	-118.3	-1744.0	/	-218.1
*3	-13.1	-12.9	-12.9	-14.8	-18.0	-34.7	-12.6	-15.2
*5	-23.5	-24.7	-23.5	-28.2	-29.2	-23.9	-26.8	-30.5
*10	-56.2	-53.4	-50.1	-52.8	-49.2	-56.0	/	-68.5
*50	-132.6	-157.1	-121.2	-159.3	-106.6	-814.9	/	-211.0
*100	-147.7	-147.5	-121.0	-339.1	-71.3	-723.8	/	-194.0

Table A.6: Correlated k-out-of-n system best policies (* = campaign cost).

n	QMIX	QVMix	QPLEX	COMA	FACMAC	IQL	DQN	Heuristics
2	-23.3	-23.3	-23.2	-23.7	-40.5	-23.7	-23.2	-58.3
4	-47.1	-47.4	-47.1	-47.9	-122.4	-47.4	-47.7	-116.9
10	-118.4	-119.4	-118.5	-122.2	-235.2	-120.8	/	-292.3
50	-604.4	-613.9	-604.6	-2805.8	-627.3	-2892.5	/	-1463.8
100	-1224.1	-1238.8	-1213.2	-5785.1	-1625.2	-5785.1	/	-2925.0
*2	-51.8	-52.0	-51.9	-60.1	-60.3	-52.0	-48.9	-62.2
*4	-80.5	-80.7	-80.7	-122.2	-118.6	-85.6	-76.0	-115.2
*10	-129.3	-133.3	-130.0	-314.5	-196.4	-132.0	/	-267.2
*50	-432.9	-436.9	-434.5	-2892.5	-502.8	-1709.7	/	-1248.2
*100	-808.1	-829.0	-852.3	-5785.1	-1280.5	-5785.1	/	-2436.3

Table A.7: Offshore wind farm best policies (* = campaign cost).

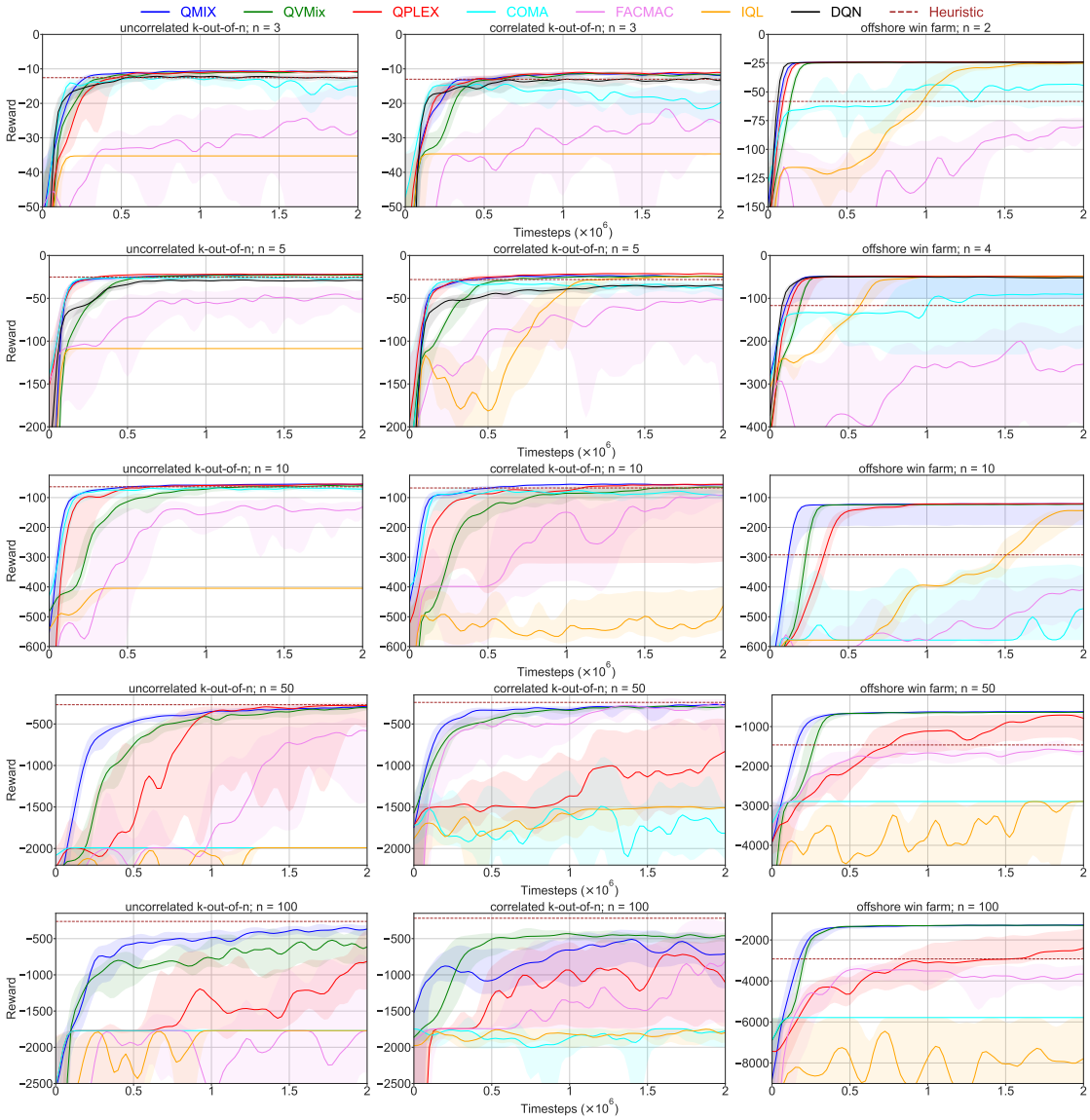


Figure A.7: Learning curves in all environments with no campaign cost. Curves represent the sum of discounted rewards obtained during test time. The bold line is the median, while the 25th and 75th percentiles delimit the error bands. Colours represent the different methods, and each environment's parameters can be inferred from the title above its corresponding graph.

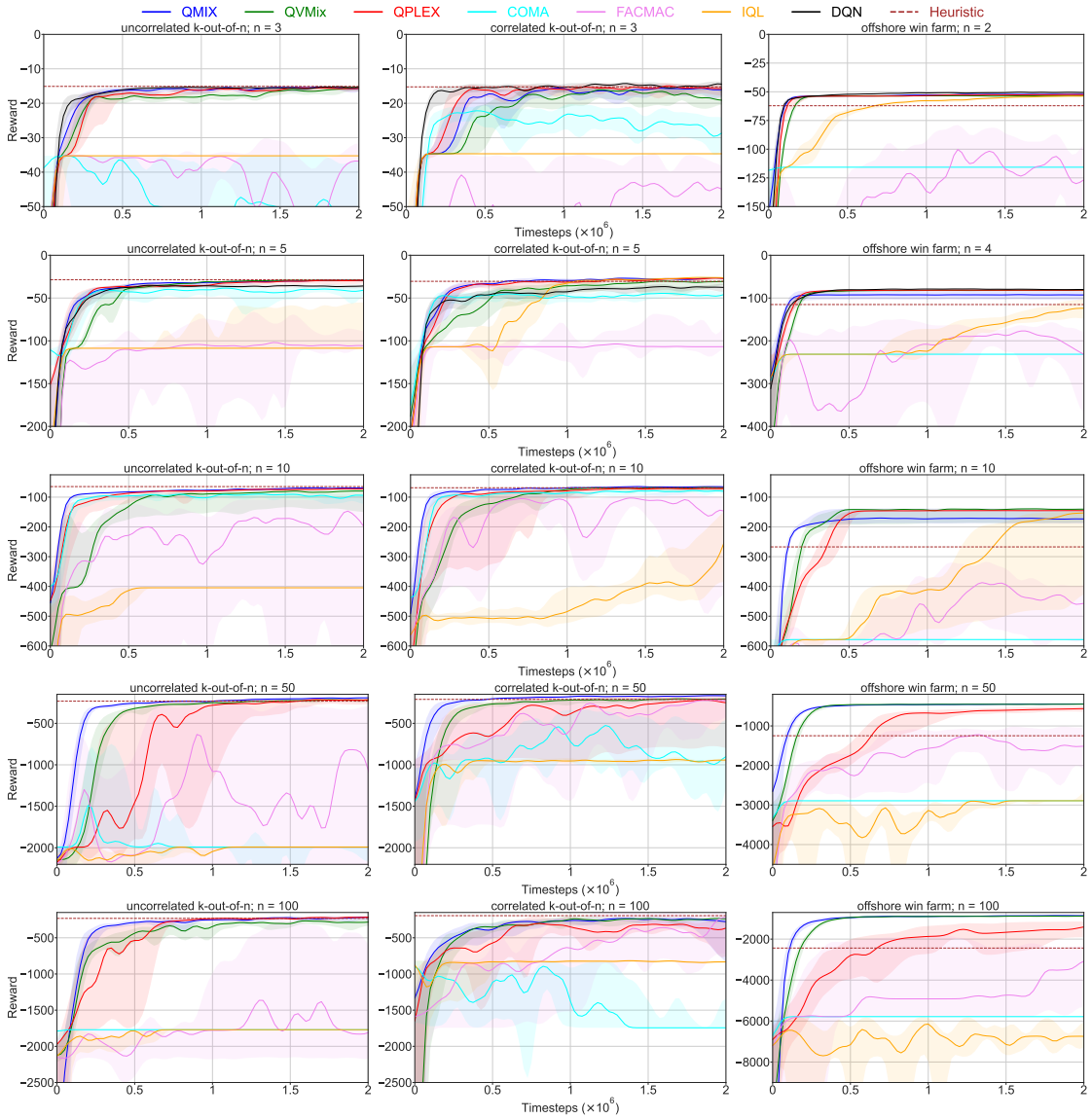


Figure A.8: Learning curves in all environments with campaign cost. Curves represent the sum of discounted rewards obtained during test time. The bold line is the median, while the 25th and 75th percentiles delimit the error bands. Colours represent the different methods, and each environment's parameters can be inferred from the title above its corresponding graph.

B

TWO-TEAM MARKOV GAME SUPPLEMENTARY MATERIALS

B.1 TRAINING PARAMETERS

The learning parameters of the three methods were determined by default configurations provided by their different authors. They are the same as the ones used for experiments in Chapter 4. Individual networks are 64 cells GRU enclosed with fully connected layers (see Chapter 3). The mixer network is the same as in [Rashid et al., 2018] with an embedded size of 32. The individual and mixer networks are the same for the three methods. We used the default parameters of MAVEN policy networks provided by [Mahajan et al., 2019]. For QVMix, the V network is a copy of the QMIX network with only one output for each V network.

For each learning scenario, networks are updated regardless of how episodes have been generated. Networks are updated from a replay buffer that collects the 5000 latest played episodes, and 32 of them are sampled to update the network. The network update is performed every eight episodes in the $3m$ map and every episode in the $3s5z$ map. The difference is justified by the desire to increase the number of network updates for $3s5$ to improve performances, especially against the heuristic. The epsilon greedy exploration starts with an epsilon equal to 1, decreasing linearly to 0.05 during 2 million timesteps. This is perhaps the main difference between the provided parameters, which decrease the epsilon only during 0.5 million timesteps. The discount factor is $\gamma = 0.99$, and the learning rate is 0.0005. Target networks are updated every 200 episode. We refer the reader to [Mahajan et al., 2019] for further parameter definitions for MAVEN optimisation.

B.2 TRAINING TIME

Experiments were performed with CPUs only because small recurrent neural networks do not arguably benefit from GPU. We had access to several types of computers, with different numbers of CPUs accessible at the same time. Training times for each experiment performed are presented in Figure B.1. With all these different hardware configurations, it is not possible to rigorously compare the times of the experiments. However, it is possible to present the time complexity. As explained in Section 7.5, training in self-play requires five times fewer environment timesteps than training within a population but also five times fewer network updates. Furthermore, networks are updated sequentially when training a population, increasing the time. Finally, SC2 processes are prone to errors and sometimes need to be restarted. As the actions of all agents in the different

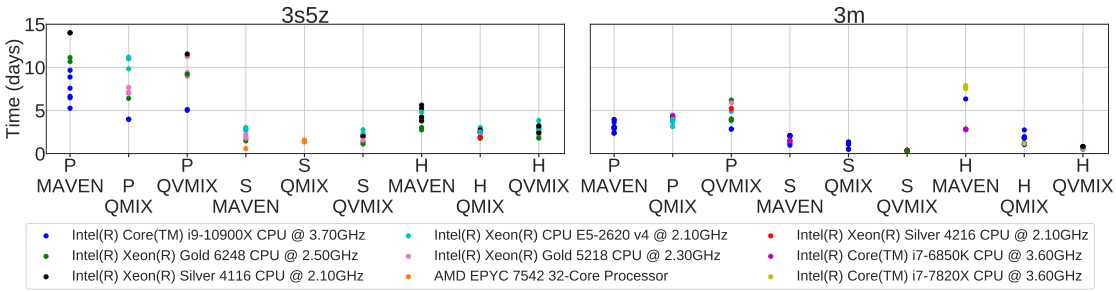


Figure B.1: Training duration, in days, for the different learning scenarios tested in this paper. The different colours represent the different CPUs used to perform the experiments.

running environments are performed simultaneously, these restarts are time-consuming operations as the processes have to wait for the faulty one.

C

REFERENCES

I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv:1910.07113*, 2019.

S. V. Albrecht, F. Christianos, and L. Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2023.

C. P. Andriotis and K. G. Papakonstantinou. Managing engineering systems with large state and action spaces through deep reinforcement learning. *Reliability Engineering & System Safety*, 2019.

C. P. Andriotis and K. G. Papakonstantinou. Deep reinforcement learning driven inspection and maintenance planning under incomplete information and constraints. *Reliability Engineering & System Safety*, 2021.

R. Avalos, M. Reymond, A. Nowé, and D. M. Roijers. Local advantage networks for multi-agent reinforcement learning in Dec-POMDPs. *Transactions on Machine Learning Research*, 2023.

B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch. Emergent tool use from multi-agent autocurricula. In *International Conference on Learning Representations*, 2019.

N. Bard, J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. F. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, I. Dunning, S. Mourad, H. Larochelle, M. G. Bellemare, and M. Bowling. The Hanabi challenge: A new frontier for AI research. *Artificial Intelligence*, 2020.

R. E. Barlow and K. D. Heidtmann. Computing k-out-of-n system reliability. *IEEE Transactions on Reliability*, 1984.

M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*. PMLR, 2017.

R. Bellman. Dynamic programming. *Science*, 1966.

D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 2002.

E. Bismut and D. Straub. Optimal adaptive inspection and maintenance planning for deteriorating structural systems. *Reliability Engineering & System Safety*, 2021.

A. Bolland, G. Lambrechts, and D. Ernst. Behind the myth of exploration in policy gradients. *arXiv:2402.00162*, 2024.

A. Bou, M. Bettini, S. Dittert, V. Kumar, S. Sodhani, X. Yang, G. D. Fabritiis, and V. Moens. TorchRL: A data-driven decision-making library for pytorch. In *The Twelfth International Conference on Learning Representations*, 2024.

C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, 1996.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv:1606.01540*, 2016.

G. W. Brown. Iterative solution of games by fictitious play. *Act. Anal. Prod Allocation*, 1951.

N. Brown and T. Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 2018.

L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC Press, 2010.

M. Campbell, A. J. Hoane Jr, and F.-h. Hsu. Deep blue. *Artificial intelligence*, 2002.

F. Christianos, L. Schäfer, and S. Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS Workshop on Deep Learning*, 2014.

C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multi-agent systems. *AAAI*, 1998.

D. Cornelisse and E. Vinitsky. Human-compatible driving partners through data-regularized self-play reinforcement learning. *arXiv:2403.19648*, 2024.

C. S. De Witt, T. Gupta, D. Makoviichuk, V. Makoviychuk, P. H. Torr, M. Sun, and S. Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv:2011.09533*, 2020.

J. Dinneweth, A. Boubezoul, R. Mandiau, and S. Espié. Multi-agent reinforcement learning for autonomous vehicles: A survey. *Autonomous Intelligent Systems*, 2022.

O. Ditlevsen and H. O. Madsen. *Structural reliability methods*. Wiley New York, 1996.

Y. Du, L. Han, M. Fang, J. Liu, T. Dai, and D. Tao. LIIR: Learning individual intrinsic reward in multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 2019.

M. Ehrgott. Vilfredo pareto and multi-objective optimization. *Doc. math*, 2012.

B. Ellis, J. Cook, S. Moalla, M. Samvelyan, M. Sun, A. Mahajan, J. N. Foerster, and S. Whiteson. SMACv2: An improved benchmark for cooperative multi-agent reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

A. E. Elo. *The rating of chessplayers, past and present*. Ishi press international, 1978.

J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 2016.

J. Foerster, R. Y. Chen, M. Al-Shedivat, S. Whiteson, P. Abbeel, and I. Mordatch. Learning with opponent-learning awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018a.

J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018b.

A. Fombellida-Lopez. Master's thesis: Coordination on the battlefield by multi-agent reinforcement learning., 2020.

V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 2018.

F. Giro, J. Mishael, P. G. Morato, and P. Rigo. Inspection and maintenance planning for offshore wind support structures: Modelling reliability and inspection costs at the system level. In *International Conference on Offshore Mechanics and Arctic Engineering*, 2022.

R. Gorsane, O. Mahjoub, R. J. de Kock, R. Dubb, S. Singh, and A. Pretorius. Towards a standardised performance evaluation protocol for cooperative marl. *Advances in Neural Information Processing Systems*, 2022.

A. Greenwald, K. Hall, R. Serrano, et al. Correlated Q-learning. In *ICML*, 2003.

D. Ha, A. Dai, and Q. V. Le. HyperNetworks. *5th International Conference on Learning Representations*, 2016.

E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, 2004.

M. Hausknecht and P. Stone. Deep recurrent Q-learning for partially observable MDPs. *AAAI Fall Symposium Series*, 2015.

H. He, J. Boyd-Graber, K. Kwok, and H. Daumé III. Opponent modeling in deep reinforcement learning. In *International conference on machine learning*. PMLR, 2016.

J. Heinrich, M. Lanctot, and D. Silver. Fictitious self-play in extensive-form games. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2017.

N. Hlaing, P. G. Morato, J. S. Nielsen, P. Amirafshari, A. Kolios, and P. Rigo. Inspection and maintenance planning for offshore wind structural components: integrating fatigue failure criteria with Bayesian networks and Markov decision processes. *Structure and Infrastructure Engineering*, 2022.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Y. Hong, Y. Jin, and Y. Tang. Rethinking individual global max in cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2022.

J. Hu and M. P. Wellman. Nash Q-learning for general-sum stochastic games. *Journal of machine learning research*, 2003.

S. Hu, Y. Zhong, M. Gao, W. Wang, H. Dong, Z. Li, X. Liang, X. Chang, and Y. Yang. MARLlib: A scalable multi-agent reinforcement learning library. *arXiv:2210.13708*, 2022.

S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 2022.

M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, et al. Population based training of neural networks. *arXiv:1711.09846*, 2017.

M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 2019.

E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 1998.

H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. Robocup: A challenge problem for AI. *AI magazine*, 1997.

V. Konda and J. Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 1999.

J. G. Kuba, R. Chen, M. Wen, Y. Wen, F. Sun, J. Wang, and Y. Yang. Trust region policy optimisation in multi-agent reinforcement learning. *International Conference on Learning Representations*, 2021.

T. Kunz, C. Fisher, J. La Novara-Gsell, C. Nguyen, and L. Li. A multiagent cyberbattlesim for rl cyber operation agents. In *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2022.

K. Kurach, A. Raichuk, P. Stańczyk, M. Zając, O. Bachem, L. Espeholt, C. Riquelme, D. Vincent, M. Michalski, O. Bousquet, et al. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

G. Lambrechts, A. Bolland, and D. Ernst. Recurrent networks, hidden states and beliefs in partially observable environments. *Transactions on Machine Learning Research*, 2022.

M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Perolat, D. Silver, and T. Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

M. Laurière, S. Perrin, M. Geist, and O. Pietquin. Learning mean field games: A survey. *arXiv:2205.12944*, 2022.

P. Leroy, D. Ernst, P. Geurts, G. Louppe, J. Pisane, and M. Sabatelli. QVMix and QVMix-Max: extending the deep quality-value family of algorithms to cooperative multi-agent reinforcement learning. *AAAI-21 Workshop on Reinforcement Learning in Games*, 2021.

P. Leroy, J. Pisane, and D. Ernst. Value-based CTDE methods in symmetric two-team markov game: from cooperation to team competition. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

P. Leroy, P. G. Morato, J. Pisane, A. Kolios, and D. Ernst. IMP-MARL: a suite of environments for large-scale infrastructure management planning via MARL. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. RLlib: Abstractions for distributed reinforcement learning. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

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. In *International conference on learning representations (ICLR)*, 2015.

L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 1992.

M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings*. Elsevier, 1994.

I. Lotsberg, G. Sigurdsson, A. Fjeldstad, and T. Moan. Probabilistic methods for planning of inspection for fatigue cracks in offshore structures. *Marine Structures*, 2016.

R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in Neural Information Processing Systems (NIPS)*, 2017.

J. Luque and D. Straub. Risk-based optimal inspection strategies for structural systems using dynamic Bayesian networks. *Structural Safety*, 2019.

X. Lyu, Y. Xiao, B. Daley, and C. Amato. Contrasting centralized and decentralized critics in multi-agent reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 2021.

A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson. MAVEN: Multi-agent variational exploration. *Advances in Neural Information Processing Systems*, 2019.

J. McInerney, B. Lacker, S. Hansen, K. Higley, H. Bouchard, A. Gruson, and R. Mehrotra. Explore, exploit, and explain: personalizing explainable recommendations with bandits. In *Proceedings of the 12th ACM conference on recommender systems*, 2018.

R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, 2018.

A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 2021.

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*, 2015.

T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 2023.

S. Mohanty, E. Nygren, F. Laurent, M. Schneider, C. Scheller, N. Bhattacharya, J. Watson, A. Egli, C. Eichenberger, C. Baumberger, et al. Flatland-RL : Multi-agent reinforcement learning on trains. *arXiv:2012.05893*, 2020.

P. G. Morato, K. G. Papakonstantinou, C. P. Andriotis, J. S. Nielsen, and P. Rigo. Optimal inspection and maintenance planning for deteriorating structural components through dynamic Bayesian networks and Markov decision processes. *Structural Safety*, 2022.

P. G. Morato, C. P. Andriotis, K. G. Papakonstantinou, and P. Rigo. Inference and dynamic decision-making for deteriorating systems with probabilistic dependencies through bayesian networks and deep reinforcement learning. *Reliability Engineering & System Safety*, 2023.

P. Muller, S. Omidshafiei, M. Rowland, K. Tuyls, J. Perolat, S. Liu, D. Hennes, L. Marris, M. Lanctot, E. Hughes, Z. Wang, G. Lever, N. Heess, T. Graepel, and R. Munos. A generalized training approach for multiagent learning. In *International Conference on Learning Representations*, 2020.

J. F. Nash Jr. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 1950.

V.-T. Nguyen, P. Do, A. Voisin, and B. Iung. Weighted-QMIX-based optimization for maintenance decision-making of multi-component systems. *PHM Society European Conference*, 2022.

A. Nowé, P. Vrancx, and Y.-M. De Hauwere. Game theory and multi-agent reinforcement learning. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning: State-of-the-Art*. Springer Berlin Heidelberg, 2012.

F. A. Oliehoek and C. Amato. *A concise introduction to decentralized POMDPs*. Springer, 2016.

F. A. Oliehoek, M. T. Spaan, and N. Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 2008.

OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv:1912.06680*, 2019.

A. Oroojlooy and D. Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, 2023.

X. Pan, M. Liu, F. Zhong, Y. Yang, S.-C. Zhu, and Y. Wang. MATE: Benchmarking multi-agent reinforcement learning in distributed target coverage control. In *Advances in Neural Information Processing Systems*, 2022.

K. G. Papakonstantinou and M. Shinozuka. Planning structural inspection and maintenance policies via dynamic programming and Markov processes. Part I: Theory. *Reliability Engineering & System Safety*, 2014a.

K. G. Papakonstantinou and M. Shinozuka. Planning structural inspection and maintenance policies via dynamic programming and Markov processes. Part II: POMDP implementation. *Reliability Engineering & System Safety*, 2014b.

G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*, 2021.

B. Peng, T. Rashid, C. Schroeder de Witt, P.-A. Kamienny, P. Torr, W. Boehmer, and S. Whiteson. FACMAC: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

J. Perolat, B. D. Vylder, D. Hennes, E. Tarassov, F. Strub, V. de Boer, P. Muller, J. T. Connor, N. Burch, T. Anthony, S. McAleer, R. Elie, S. H. Cen, Z. Wang, A. Gruslys, A. Malysheva, M. Khan, S. Ozair, F. Timbers, T. Pohlen, T. Eccles, M. Rowland, M. Lanctot, J.-B. Lespiau, B. Piot, S. Omidshafiei, E. Lockhart, L. Sifre, N. Beauguerlange, R. Munos, D. Silver, S. Singh, D. Hassabis, and K. Tuyls. Mastering the game of stratego with model-free multiagent reinforcement learning. *Science*, 2022.

T. Phan, T. Gabor, A. Sedlmeier, F. Ritz, B. Kempter, C. Klein, H. Sauer, R. Schmid, J. Wieghardt, M. Zeller, et al. Learning and testing resilience in cooperative multi-agent systems. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020.

S. J. Prince. *Understanding Deep Learning*. MIT Press, 2023.

T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

T. Rashid, G. Farquhar, B. Peng, and S. Whiteson. Weighted QMIX: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 2020.

C. Resnick, W. Eldridge, D. Ha, D. Britz, J. Foerster, J. Togelius, K. Cho, and J. Bruna. Pommerman: A multi-agent playground. In *CEUR Workshop Proceedings*, 2018.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

M. Sabatelli. *Contributions to Deep Transfer Learning: from Supervised to Reinforcement Learning*. PhD thesis, Université de Liège, Liège, Belgique, 2022.

M. Sabatelli, G. Louppe, P. Geurts, and M. A. Wiering. Deep quality-value (DQV) learning. *Advances in Neural Information Processing Systems, Deep Reinforcement Learning Workshop*, 2018.

M. Sabatelli, G. Louppe, P. Geurts, and M. A. Wiering. The deep quality-value family of deep reinforcement learning algorithms. In *International Joint Conference on Neural Networks (IJCNN)*, 2020.

M. Saifullah, C. Andriotis, K. Papakonstantinou, and S. Stoffels. Deep reinforcement learning-based life-cycle management of deteriorating transportation systems. In *Bridge Safety, Maintenance, Management, Life-Cycle, Resilience and Sustainability*, 2022.

M. Samvelyan, T. Rashid, C. Schroeder de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019.

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 2020.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*. PMLR, 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

C. E. Shannon. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1950.

L. S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 1953.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*. Pmlr, 2014.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 2017.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 2018.

K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *Proceedings of the 36th International Conference on Machine Learning*, 2019.

P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018.

R. S. Sutton and A. G. Barto. *Reinforcement learning, second edition: An introduction*. MIT press, 2018.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 1999.

A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PLOS ONE*, 2017.

M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on machine learning*, 1993.

J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 2021.

G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 1994.

T. Théate, A. Wehenkel, A. Bolland, G. Louppe, and D. Ernst. Distributional reinforcement learning with unconstrained monotonic neural networks. *Neurocomputing*, 2023.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ international conference on intelligent robots and systems*, 2012.

M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis. Gymnasium, 2023.

H. Van Hasselt. Double Q-learning. *Advances in neural information processing systems*, 2010.

H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, 2016.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 2017.

E. Vinitsky, N. Lichtlé, X. Yang, B. Amos, and J. Foerster. Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world. *arXiv:2206.09889*, 2022.

O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. KÃŒttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. V. Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing. StarCraft II: A new challenge for reinforcement learning. *arXiv:1708.04782*, 2017.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 2019.

J. Von Neumann and O. Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1944.

J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang. QPLEX: Duplex dueling multi-agent Q-learning. In *International Conference on Learning Representations*, 2021.

Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. PMLR, 2016.

C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 1992.

L. Weaver and N. Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, 2001.

M. Wen, J. G. Kuba, R. Lin, W. Zhang, Y. Wen, J. Wang, and Y. Yang. Multi-agent reinforcement learning is a sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2022.

M. A. Wiering. QV(λ)-learning: A new on-policy reinforcement learning algrithm. In *Proceedings of the 7th European Workshop on Reinforcement Learning*, 2005.

M. A. Wiering and H. Van Hasselt. The QV family compared to other reinforcement learning algorithms. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*. IEEE, 2009.

D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Recurrent policy gradients. *Logic Journal of IGPL*, 2010.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, 1992.

R. J. Williams and J. Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 1991.

D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 2001.

Y. Yang, J. Hao, B. Liao, K. Shao, G. Chen, W. Liu, and H. Tang. Qatten: A general framework for cooperative multiagent reinforcement learning. *arXiv:2002.03939*, 2020.

A. B. Yoo, M. A. Jette, and M. Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing: 9th International Workshopmorato2022syst*. Springer, 2003.

C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 2022.

A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023.

H. Zhang, S. Feng, C. Liu, Y. Ding, Y. Zhu, Z. Zhou, W. Zhang, Y. Yu, H. Jin, and Z. Li. CityFlow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The world wide web conference*, 2019.