Université libre de Bruxelles

Analyzing MARL algorithms in dynamic environment: Evaluating performance with an additional unknown element

Preparatory work for the master thesis -- MEMO-F-403

Promoter:
Yannick MOLINGHEN

Author:
Kevin VANDERVAEREN

Supervisor:
Prof. Tom LENAERTS

Abstract

Table of Contents

I Introduction

I.1 Background and Objectives

Multi-Agent Reinforcement Learning (MARL) is a subfield of the Reinforcement Learning domain that focuses on the interaction between multiple agents in a shared environment. In recent years, an increasing amount of research has been conducted in this field to resolve issues that have arisen in the real world [1], [2]. However, most of the research has been done through simulations in environments that do not involve unknown elements in existing settings. This thesis aims to evaluate the learning performance of MARL algorithms when moving from a known environment with proven working results to a slightly modified environment by adding unknown elements. This work is carried out under the supervision of Prof. Tom Lenaerts and advisor Yannick Molinghen from the Machine Learning Group (MLG) of the Université Libre de Bruxelles (ULB).

Currently, the research is focused on the environment of LLE (Laser Learning Reinforcement), which is an environment created by Yannick Molinghen based on the original game. The environment is a 2D world, also known as a grid world, where one or more agents interact in a cooperative manner. The goal of each individual agent is to reach an exit point while acquiring rewards (in the form of gems) and avoiding obstacles.

The objective of the Master's thesis is to develop a new feature in the LLE environment that was also included in the original game Oxen. Moreover, this feature has another objective: to add a new element to the environment that is not included in the agents learning process. This allows for the re-evaluation of the performance of already fine-tuned algorithms trained on the original environment and the observation of any possible bottlenecks that may arise from the addition of these new elements.

I.2 Notations and Definitions

I.2.1 Notations

Notations	Description
$\approx$	approximately equal
$\in$	is an element of (e.g., $3.2 \in \mathbb{R}$)
$[a, b)$	the interval from a to b, where a is included and b is excluded
$\sum_{a \in A} a$	the sum of all elements a in the set A
$f : X \times Y \to Z$	a function f with a domain from the set $X \times Y$ and an image set Z
$\Pr(x\|y, z)$	the probability of x given y and z
$x \sim X$	a random variable x that follows the distribution X
$\mathbb{N}$	the set of natural numbers

Notations	Description
$\mathbb{R}$	the set of real numbers
Δ_X	the set of probability distributions over the set X
t	a time step that belongs to the set of natural numbers $t \in \mathbb{N}$
$\mathbb{E}$	the expected value
γ	the discount factor, which is a real number in the interval $[0, 1]$

II State of the Art

II.1 Introduction

!!(this section contains content from the article “Cooperative Multi-Agent Learning: The State of the Art” by [ref to article])!!

Distributed Artificial Intelligence (DAI) is a field of study that has been rising over the last two decades, mainly focused on distributed systems. A distributed system, as defined by [3], is “where a number of entities work together to cooperatively solve problems”. This kind of study is not new, it has been explored for a long time. What is new, however, is the rise of the internet and the multitude of electronic devices available today, which has created the need for a new field of study: DAI. DAI is essentially the study of the interaction between multiple artificial intelligences (AIs) or agents in a distributed system.

II.1.1 Multi-Agent Systems vs. Distributed Problem Solving

Within the field of DAI, two main subfields can be identified. The more traditional one is Distributed Problem Solving (DPS), which follows a divide-and-conquer paradigm. DPS focuses on distributing the problem to independent agents (or slaves) that solve it independently. On the other hand, Multi-Agent Systems (MAS) emphasize interaction between agents.

II.1.2 Multi-Agent Systems

In MAS, a few constraints are imposed on agents. Even though agents work together to solve a problem in the same environment, they are not able to share their knowledge of the environment with each other. They can only access the information they individually perceive, which in RL is often referred to as a local observation. This is an important point because if agents were able to share their knowledge, they could simply synchronize it and solve the problem as a DPS problem if the problem required no interaction between agents .

may require more writing

II.2 Multi-Agent Learning

Multi-Agent Learning (MAL) (todo):

- Use articles that explain different MAS approaches to clarify what MARL is
- Explain why MARL is interesting
- Use the Molinghen article to describe the LLE environment
- Explain why adding a new element in the environment is interesting
- Explain LLE agent standards

II.3 Machine Learning

(todo):

- Decide whether this section is needed to explain the basics of ML and to split between supervised, unsupervised, and RL

II.3.1 Supervised Learning

Supervised Learning (SL) is a subfield of Machine Learning (ML) focused on training a model from a set of labeled data. The goal of SL is to learn a function that maps the input data (e.g., an image) to output data (or labels, e.g., the class of the image) as accurately as possible. SL is often used in computer vision and natural language processing (e.g., [4]), where the goal is to create a model capable of classifying data into specific classes based on the data learned during training.

II.3.2 Reinforcement Learning

Reinforcement Learning (RL) is a subfield of Machine Learning (ML) that focuses on learning from the interaction between an agent and its environment. Compared to supervised learning, the learner (agent) is not provided with explicit information about the environment or which actions to perform. RL is based on trial and error: by interacting with the environment, the learner acquires or loses points, which serve as the only source of feedback. Thus, agents attempt to maximize the number of points they receive [5].

II.3.2.1 Agent

An agent in RL can be seen as a learner or decision-maker equipped with a set of tools to observe and interact with its environment. These tools are generally divided into two components:
- Sensors used to perceive the environment and gather information (e.g., the five human senses).
- Actuators used to interact with the environment and perform actions (e.g., human hands or legs).

II.4 Single Agent Reinforcement Learning

II.4.1 Markov Decision Process

In Single-Agent Reinforcement Learning (RL), the methodology used to model the environment is the Markov Decision Process (MDP) [6]. The MDP is a mathematical framework used to model the interaction between an agent and its environment (`todo` find the lost ref). It is often employed to represent the decision-making process of an agent in a stochastic environment. The MDP is a powerful tool that allows the environment to be modeled in a way that is both easy to understand and analyze.

The Markov Decision Process (MDP) [7] is often represented as a 5-tuple $\langle S, A, T, R, \rho_0 \rangle$, where the elements are:
- S is the state space
- A is the action space
- T is the transition function
- R is the reward function
- ρ_0 is the initial state distribution

One of the key properties of the MDP is that it is based on the Markov property, which states that the future state of a system depends only on the current state and not on previous states. In mathematical terms, this is often represented as: $\Pr(s_{t+1} \mid s_t, a_t) = \Pr(s_{t+1} \mid s_t, a_{t-1}, ..., s_0, a_0)$

Another strength of reducing a problem to an MDP is that it allows abstraction of all sensory, memory, and control aspects (ref: RL: An Introduction, Sutton and Barto) into three simple signals between the agent and the environment:
- the state s
- the action a
- the reward r

It also introduces key functions such as the Bellman equation, which uses the Markov property to represent the relationship between the value of a state and the value of its successor states.

II.4.1.1 State

One way to represent the environment is through a state. A state is an abstract way to describe the combined information of all elements in the environment. As an example, in the game of tic-tac-toe, the representation of the board at a given time, such as in this image Figure 1, is a state. However, a state is not only the representation of the board but also includes information about the player's turn. Therefore, a state represents the environment at a given time.

In mathematical notation, s is usually used to represent a state, and S to represent the state space. The state space is the set of all possible states for a given environment.
- S is the state space of the environment
- s is a state in the state space, given that $s \in S$ (s' may be used for a new state)
- s_t is the state at time t

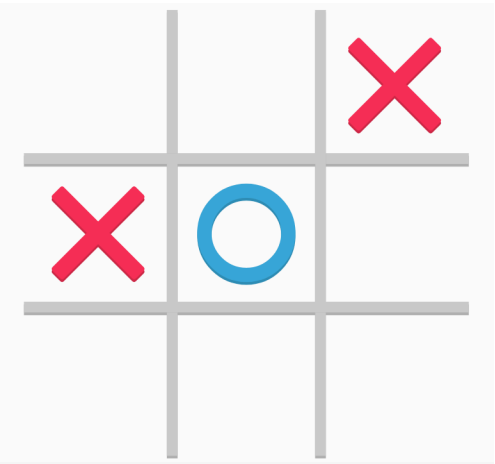


Figure 1: A state in the game of tick-tac-toe

II.4.2 Observation

An observation is a partial description of a state. Instead of providing complete information about the environment, the observation provides only the information acquired by the agent. Observations are often used when the agent does not have

access to complete information about the environment, such as in a partially observable environment (POMDP) (). The observation is denoted as:

getref

- O is the observation space
- o_t is the observation at time t in the observation space, given that $o_t \in O$

An analogy for an observation is being in a room where only what is in front is visible, while what is behind cannot be seen. In this case, the observation is the information visible in front, but not the complete information about the room.

II.4.3 Action

An action refers to the possible movement the agent can perform in the environment. In the case of the game of tic-tac-toe, the possible actions are placing a mark in one of the available cells out of the 9 cells. For example, in the previous example Figure 1, the "O" player has the following possible actions to choose from: [top left, top center, middle right, bottom left, bottom center, bottom right]. In mathematical notation, a is usually used to represent an action (e.g., "top left") and A to represent the action space (e.g., the list of all actions mentioned above).

- A is the action space of the environment
- $A(s)$ is the action space available in state s (e.g., the list of all actions available in the state s)
- a is an action in the action space, given that $a \in A$
- a_t is the action at time t in the action space

II.4.4 Transition

The transition is the function used to represent the change of a given state after taking an action. It is a probability function used to represent the stochasticity of an environment. A real-life example can be taken from sports: when about to perform an action like a squat or a sprint, a cramp or muscle tear may occur, placing the body in an unexpected state. This illustrates the stochastic nature of an environment. Using this example:

- s or s' is the state of the body when it is "healthy"
- c is the state of the body when it is "cramped" or "unhealthy"
- a is the action being performed

The transition function T represents the change of state of the body given an action. In this case:

- $T(s' \mid a, s)$ is the probability of nothing happening to the body given an action a
- $T(c \mid a, s)$ is the probability of having a cramp or muscle tear given an action a

They also possess certain properties such as:

- the function $T : S \times A \times S \to [0, 1]$
- $\sum_{s' \in S} T(s' \mid a, s) = 1$

Alternatively, the transition function can also be represented as a conditional probability function, which is often used in the literature. In this case:

- $T(\cdot \mid s, a)$ where $T : S \times A \to \Delta_S$ and Δ_S is the set of probability distributions over the state space S.

$T(s' \mid a, s) = \Pr(s' \mid a, s)$

II.4.5 Reward

The reward function takes an initial state, an action, and a final state as input. Unlike the transition function, which returns a probability, the reward function returns a scalar value that can be interpreted as a score. Instead of representing the change of a state, the reward function gives a purpose or goal to the agent.

Returning to the sports example, the score can be seen as the motivation to perform the action based on a certain goal. For example, on a treadmill when aiming to lose a certain amount of calories, the reward function is the calories burned. Running faster places the body in a state where more calories are burned but also increases the likelihood of a cramp.

The reward function is often represented as:

$$R(s' \mid a, s)$$

where s is the initial state, a is the action, and s' is the final state.

Mathematically, the reward function is:

$$R : S \times A \times S \to \mathbb{R}$$

The reward resulting from the reward function is often assigned to the variable r, and the reward at time t is commonly written as:

$$r_t = R(s_{t+1} \mid a_t, s_t)$$

II.4.6 Return

Unlike the reward, which is a scalar value given at a specific time, the return is the cumulative reward observed over a period of time. It can be either finite or infinite. In the finite case, the return is also called the **finite-horizon undiscounted return** and is often represented as: $R(\text{trajectory placeholder}) = \sum_{t=0}^{T-1} r_t$ where T is the time horizon and τ is the trajectory of the agent.

need better notation due to conflict with the reward and multi-agent notation

In the infinite case, the discount factor γ must be taken into account to avoid an unbounded return ().

do infinite case

II.4.7 Trajectory

A trajectory is a sequence of states, actions, and rewards that the agent experiences in the environment. The trajectory is written as trajectory placeholder $= (S_1, A_1, R_1, S_2, A_2, R_2, \ldots)$ where the initial state of the environment S_1 is randomly sampled from the start state distribution ρ_0: $S_1 \sim \rho_0$

The state transitions must follow the transition function T, and the actions must be sampled from the action space A at a given time t: $S_{t+1} \sim T(\cdot \mid S_t, A_t)$

II.4.8 History

A history is a sequence of actions, observations, and rewards that the agent experiences in the environment. It is often used to represent the past actions

and observations of the agent. The history is commonly written as: $h_t = (o_1, a_1, r_1, o_2, a_2, r_2, ..., o_{t-1}, a_{t-1}, r_{t-1})$ where o_t is the observation, a_t is the action, and r_t is the reward at time t.

The main difference between a trajectory and a history is that a trajectory contains all information about the environment, while a history contains only the information gathered by a specific agent. An analogy is an escape room: the history is what the player recalls from past actions and observations, while the trajectory is what the game master (who knows all the secret information that the player does not know) sees of the player's actions in the escape room.

II.4.9 Policy

A policy can be seen as the decision-making rule of the agent, where for any given state it recommends the actions to take. The policy is often represented as:

$$\pi : S \to \Delta_A$$

where π is the policy, S is the state space, and Δ_A is the set of probability distributions over the action space A.

II.4.9.1 Optimal Policy

finish the optimal policy definition

The optimal policy is the policy that maximizes the expected return of the agent. It is often represented as:

II.4.10 Action-Utility Function

double check the action-utility function definition

The action-utility function represents the expected return of a given state-action pair. It is often expressed as:

$$Q(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

where $Q(s, a)$ is the action-utility function, $\mathbb{E}$ is the expected value, γ is the discount factor, and r_t is the reward at time t.

The action-utility function is frequently used to evaluate the expected return of specific state-action combinations. It can also be used to derive the optimal policy by maximizing the expected return.

II.4.11 Value Function

same as action-utility function

The value function represents the expected return of a given state under a policy. It is often expressed as:

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, \pi\right]$$

where $V(s)$ is the value function, E is the expected value, γ is the discount factor, and r_t is the reward at time t.

The value function is used to evaluate the expected return of states under a given policy and can also be employed to find the optimal policy by maximizing the expected return.

II.5 Multi-Agent Reinforcement Learning

II.5.1 Stationary vs. Non-stationary

Originally, it might seem that adding multiple independent agents would not dramatically increase the complexity compared to single-agent RL. However, this assumption has been proven incorrect. (use references)

MARL can be naively viewed as simply adding more than one agent to an RL environment. This introduces new challenges, such as non-stationarity, since the presence of multiple agents alters the dynamics of the environment [3], [8].

Non-stationarity is one of the main challenges in MARL because multiple agents perceive each other as part of the environment that cause it to obseerve "undeterministic" behavior due to their own learning processes. Thus violates the Markov property by definition.

Based on this, two main research trends have emerged in the MARL field:

- The first, known as concurrent learning, is where agents learn independently from each other. However, this approach does not solve the non-stationarity problem . add depth here

II.5.2 Search Space

As the number of agents increases, the number of possible joint actions grows exponentially. This leads to a combinatorial explosion in the search space, making it computationally impossible to find the optimal joint action within a reasonable time.

II.5.3 Current Approaches

The current approaches to solving these challenges include...

III LLE Environment

III.1 Overview

The Laser Learning Environment (LLE) is a 2D grid world with discrete time steps and multiple cooperative agents. The game is based on the original game Oxen, where the goal of each agent is to reach an exit point while acquiring gems (bonus points). All agents cooperate to reach their respective exit points while avoiding obstacles. The environment is designed to be simple and easy to understand while still being challenging enough to test the performance of MARL algorithms.

III.2 Environment Challenges

The environment is aimed at testing the performance of MARL algorithms tailored for decentralized cooperative scenarios and includes challenges not present in other environments such as the StarCraft Multi-Agent Challenge (SMAC) [9] or the Hanabi environment [10]. Instead, this environment is designed to incorporate factors such as perfect coordination, interdependence, and zero-incentive dynamics [11].

III.3 Multi-Agent Markov Decision Process

The model of the environment is based on the Multi-Agent Markov Decision Process (MMDP) [12], a generalization of the Markov Decision Process (MDP) to multiple agents. The MMDP is a tuple $\langle n, S, \mathcal{A}, \mathcal{T}, \mathcal{R}, s_0, s_f \rangle$ where: -n is the number of agents -S is the set of states -$\mathcal{A} \equiv A^1 \times A^2 \times ... \times A^n$ is the joint action space, and A^i is the set of actions available to agent i[*]) -$\mathscr{a} \equiv (a^1, a^2, ..., a^n) \in \mathcal{A}$ is the joint action of all agents, where a^i is an action of agent i -$\mathcal{T} : S \times \mathcal{A} \to \Delta_S$ is a function that gives the probability of transitioning from state s to state s' given a joint action $\mathscr{a}$ -$\mathcal{R} : S \times \mathcal{A} \times S \to \mathbb{R}$ is the function that returns the reward obtained when transitioning from state s to state s' given a joint action $\mathscr{a}$ -$s_0 \in S$ is the initial state -$s_f \in S$ is the final state A transition is defined as $\tau = \langle s, \mathscr{a}, r, s' \rangle$ with $r \in \mathbb{R}$.

III.4 Algorithm

The algorithm used in the LLE environment is based on the CTDE approach mentioned previously. Based on the current state of the LLE environment, only a few algorithms have been tested [11].

III.4.1 Value Decomposition Networks

Value Decomposition Networks (VDN) [13] is a MARL algorithm that leverages the hypothesis of decomposing the joint action-value function into individual value functions for each agent:

[*] A^i was modified from the original notation A_i to avoid confusion with the action space at a given time t

$$Q((h^1, h^2, ..., h^n), (a^1, a^2, ..., a^n)) \approx \sum_{i=1}^{n} \tilde{Q}_i(h^i, a^i)$$

where $\tilde{Q}_i$ is the value function of agent i, and h^i is the history of agent i. This methodology allows agents to learn independently through $\tilde{Q}$ while still producing a global result for the group Q.

III.4.2 independent Q-learning

...

III.4.3 QMix

...

III.5 others ?

IV Objectives

IV.1 Research Questions

The main research question of this thesis is to evaluate the performance of MARL algorithms by comparing their performance using the original environment evaluation criteria. Additional questions include:

- How does the introduction of previously unseen environmental elements affect the convergence speed of Multi-Agent Reinforcement Learning (MARL) algorithms that use the CTDE method in cooperative tasks?
- Can pre-trained policies adapt without retraining when facing unseen elements?
- Can MARL policies trained in a fully known environment generalize to environments with partially unknown dynamics without retraining?
- Does incorporating a lift/lever mechanism as an unknown dynamic element lead to measurable differences in agent behavior compared to the baseline LLE environment?
- Can agents adapt to unknown elements faster if the algorithm employs centralized training with decentralized execution (CTDE) versus fully independent learning?

IV.2 Lift and Lever

The objective of this thesis is to develop a new feature in the LLE environment that consists of adding a lift, which allows agents to have more possible actions. With this new feature, the aim is to evaluate the performance of previously trained MARL algorithms on the original environment and observe whether potential bottlenecks arise from the addition of this element. The lift is designed to be used in conjunction with the lever, which activates the lift.

IV.2.1 Lift

The lift is a terrain type that allows agents to reach higher levels in the environment. It is designed to work with the lever, which activates it. The lift can be used to access new areas of the environment, enabling agents to explore and find alternative paths to their goals.

IV.2.2 Lever

The lever is a terrain type that agents can interact with when standing on it. The lever activates the lift, allowing agents on the lift to switch floors. It is intended to work in conjunction with the lift, enabling agents to reach new areas of the environment.

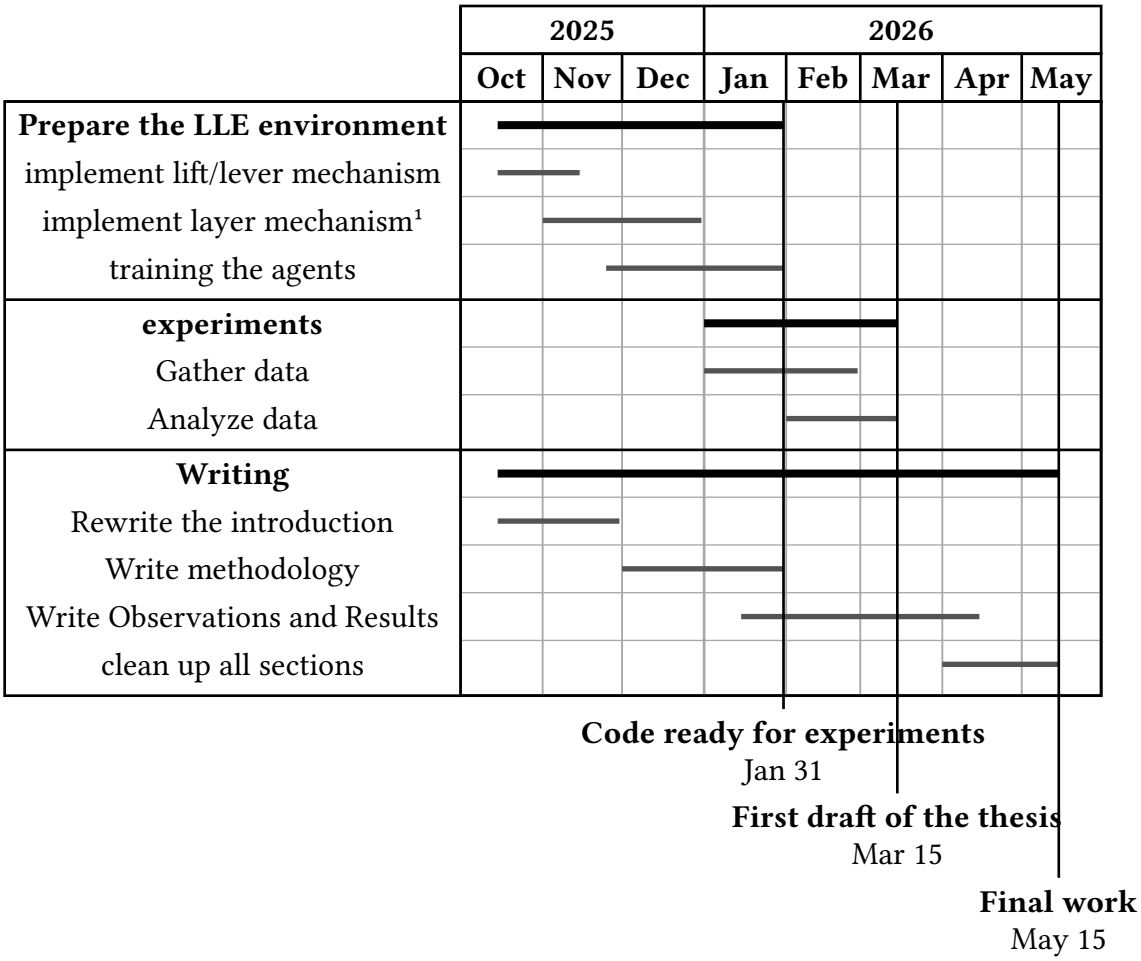
IV.2.3 Plane Extension

The plane extension is the addition of a new dimension that allows the lift to move vertically...

IV.3 Evaluation

...

V Time Plan


2025
2026
Oct
Nov
Dec
Jan
Feb
Mar
Apr
May
Prepare the LLE environment
implement lift/lever mechanism
implement layer mechanism[1]
training the agents
experiments
Gather data
Analyze data
Writing
Rewrite the introduction
Write methodology
Write Observations and Results
clean up all sections
Code ready for experiments
Jan 31
First draft of the thesis
Mar 15
Final work
May 15

Bibliography

[1] G. Weiss, Ed., *Multiagent systems: a modern approach to distributed artificial intelligence*, 3. print. Cambridge, Mass.: MIT Press, 2001.

[2] P. Stone and M. Veloso, "Multiagent Systems: A Survey from a Machine Learning Perspective:," Fort Belvoir, VA, Dec. 1997. doi: 10.21236/ADA333248.

[3] L. Panait and S. Luke, "Cooperative Multi-Agent Learning: The State of the Art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, Nov. 2005, doi: 10.1007/s10458-005-2631-2.

[4] U. Kamath, J. Liu, and J. Whitaker, *Deep Learning for NLP and Speech Recognition*. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-030-14596-5.

[5] R. S. Sutton and A. Barto, *Reinforcement learning: an introduction*, Nachdruck. in Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2014.

[6] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, no. v.414. in Wiley Series in Probability and Statistics. Hoboken: John Wiley & Sons, Inc, 2009.

[7] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[8] M. Bowling and M. Veloso, "An Analysis of Stochastic Game Theory for Multiagent Reinforcement Learning."

[9] M. Samvelyan *et al.*, "The StarCraft Multi-Agent Challenge." Accessed: Jan. 31, 2025. [Online]. Available: http://arxiv.org/abs/1902.04043

[10] N. Bard *et al.*, "The Hanabi challenge: A new frontier for AI research," *Artificial Intelligence*, vol. 280, p. 103216, Mar. 2020, doi: 10.1016/j.artint.2019.103216.

[11] Y. Molinghen, R. Avalos, M. V. Achter, A. Nowé, and T. Lenaerts, "Laser Learning Environment: A new environment for coordination-critical multi-agent tasks." Accessed: Jan. 31, 2025. [Online]. Available: http://arxiv.org/abs/2404.03596

[12] C. Boutilier, "Planning, Learning and Coordination in Multiagent Decision Processes."

[13] P. Sunehag *et al.*, "Value-Decomposition Networks For Cooperative Multi-Agent Learning." Accessed: Jan. 31, 2025. [Online]. Available: http://arxiv.org/abs/1706.05296