

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

In recent years, artificial intelligence (AI) and machine learning (ML) systems have demonstrated remarkable capabilities in real world applications. However, their performance often degrades in the presence of untrained elements or domain shifts. One pertinent challenge is evaluating how the robustness and performance of a given algorithm-particularly those used in autonomous systems-change when the environment is perturbed by the introduction of unknown or novel elements. This research is situated within the broader field of AI robustness and domain generalization, with applications of intelligent decision-making systems.

Table of Contents

I	Introduction	1
II	State of the Art	2
II.1	Introduction	2
II.2	Multi-Agent Learning	2
II.3	Machine Learning	2
II.4	Single Agent Reinforcement Learning	3
II.5	Multi-Agent Reinforcement Learning	8
III	Laser Learning Environment	10
III.1	Overview	10
III.2	Environment Features	10
III.3	Implementation	13
III.4	Environment Challenges	15
III.5	Multi-Agent Markov Decision Process	15
III.6	Algorithms	15
IV	Research Questions	17
V	Evaluation method	18
V.1	Result comparison	18
V.2	Evaluation Metrics	18
VI	Implementation requirements	19
VI.1	Lift	19
VI.2	Lever	19
VI.3	Extra dimension	19
VI.4	Other possibilities	20
	Bibliography	21
VII	Appendix	25
VII.1	Notations	25
VII.2	Acronyms	25
VII.3	Time Plan	26

I Introduction

Imagine a fleet of autonomous vehicles navigating through a city, each equipped with advanced Artificial Intelligence (AI) algorithms to accomplish a specific mission, while navigating through a complex and hazardous environment, and still managing to achieve a optimum solution by cooperating. This scenario is no longer a far-fetched vision of the future, but a current reality, driven by recent advancements in AI and Machine Learning (ML). In recent years, AI and ML research has made significant leaps. However, even with these advancements, AI systems still struggle sometimes to escape bottlenecks which impact drastically the learning efficiency.

This work aims to investigate how bottlenecks arise by trying to challenge the existing environment with the addition of new elements in the environment, By doing so, we can observe how the performance of previously designed algorithms is affected by the introduction of these new perturbations.

The current work is focused on the environment of Laser Learning Reinforcement (LLE) [1] , which a toy environment created with the goal of evaluating the performance of Multi-Agent Reinforcement Learning (MARL) algorithms. The goal of the environment is by controlling a agent in a group of agents to reach an exit point while collecting gems and avoiding obstacles.

The core objective of the Master's thesis Work 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: by adding adding new element, try to observe if it is possible to have emerging bottlenecks. This allows for the re-evaluation of the performance of already fine-tuned algorithms from original environment and the observation of any possible bottlenecks that may arise from the addition of these new elements. This investigation not only contributed to the extension of the LLE, but also provide insights into the robustness of MARL algorithms in increasingly complex environments.

II State of the Art

II.1 Introduction

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 L. Panait and S. Luke [2], 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[3], [4]. 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 .

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

may require more writing

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., [5]), 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 [6].

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) [7]. 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) [8] 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 represented as: $\Pr(s_{t+1} \mid s_t, a_t) = \Pr(s_{t+1} \mid s_t, a_{t-1}, \dots, s_0, a_0)$

Another strength of formalizing 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 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

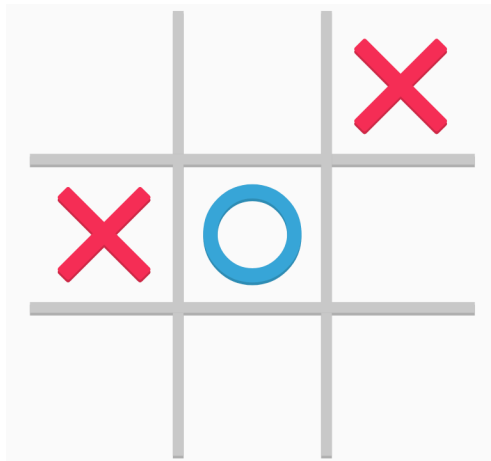


Figure 1: A state in the game of tick-tac-toe

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:

- 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 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:

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

They also possess certain properties such as:

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

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

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

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

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 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:

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

The reward resulting from the reward function is 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 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.

In the infinite case, the discount factor γ must be taken into account to avoid an unbounded, the infinite-horizon discounted return is often represented as:

$$R(\text{trajectory placeholder}) = \sum_{t=0}^{\infty} \gamma^t r_t$$

The return is often a better measure for evaluating the performance of certain trajectory and by adding the discount factor γ , it allows the agent to put more importance on recent rewards over distance rewards. And given that γ is a parameter in the interval $[0, 1]$, it can be used to control the importance of future rewards. A value of $\gamma = 0$ means that only the immediate reward is considered, while a value of $\gamma = 1$ means that all future rewards are considered equally important.

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

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

$$\text{trajectory placeholder} = (S_1, A_1, R_1, S_2, A_2, R_2, \dots)$$

where the initial state of the environment S_1 is randomly sampled from the start state distribution $\rho_0 : S_1 \sim \rho_0$. 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 : $S_{t+1} \sim T(\cdot | S_t, A_t)$

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:

$$h_t = (o_1, a_1, r_1, o_2, a_2, r_2, \dots, o_{t-1}, a_{t-1}, r_{t-1})$$

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 has a mapping to a set of probabilities over the possible actions. The policy is represented as:

$$\pi_\theta : S \rightarrow \Delta_A$$

where π is the policy, θ is the parameter of the policy, S is the state space, and Δ_A is the set of probability distributions over the action space A . thus maze solving agent that follows a policy π (e.g. always taking same turn) with θ (e.g. prefer left) will be answering "LEFT" to the question "What is the next action to take in state s ?"

A policy parameter θ is the set of parameters that is used in the policy to determine the action probabilities. These parameters are typically learned from data through a training process.

II.4.9.1 Optimal Policy

The optimal policy is the policy that maximizes the expected return (or value) of the agent. It is represented as:

$$\pi^* = \operatorname{argmax}_\pi \mathbb{E}[R(\text{trajectory placeholder}) | \pi]$$

II.4.10 Action-Utility Function

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

The value function represents the expected return of a given state under a policy. It is 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 [2], [9].

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 observe “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

- The second, known as team learning, is where agents learn together as a team. This approach aims to mitigate the non-stationarity problem by allowing agents to act as a single entity while in the training phase. But other challenges arise, such as the search space explosion, or the credit assignment problem.

II.5.2 Search Space

By using method such as team learning, the agents can be seen as a single entity. However, this leads to a combinatorial explosion in the search space. Let take the joint action space, given that is the cartesian product of the individual action spaces. if grid world of a certain number of cells and 2 agents, each with 4 possible actions (up, down, left, right). The joint action space would be $A^1 \times A^2 = 4 \times 4 = 16$ possible joint actions and by adding a third agent, would be 64 possible joint actions. By applying this geometric growth, and giving that we have n agents, each with m possible actions, the joint action space is given by $\mathcal{A} = m^n$ thus this model is not possible but not practical to use in real-world scenarios. This is known as the curse of dimensionality.

II.5.3 Credit Assignment Problem

The credit assignment problem is a challenge that arises in MARL when trying to determine which agent is responsible for a specific outcome. In a cooperative environment, agents must work together to achieve a common goal, but it can be difficult to determine which agent's actions contributed to the success or failure of the team. For example, consider a soccer game where the team wins; it is difficult to determine which player is responsible for the victory. The group of players is graded as a whole, but it is not clear which player contributed most and which player slacked off.

In MARL, this problem is particularly pronounced because rewards are often shared among agents, making it hard to assign credit to individual actions. This ambiguity can hinder learning efficiency, as agents may not receive appropriate feedback for their contributions. Various approaches have been proposed to address the credit assignment problem, such as difference rewards, value decomposition, and counterfactual reasoning. These methods aim to provide more informative feedback to each agent, enabling them to better understand the impact of their actions on the overall outcome and improve cooperative behavior.

II.5.4 Current Approaches

The current approaches to solving the challenges of credit assignment and non-stationarity in MARL is to use the Centralized Training with Decentralized Execution (CTDE) paradigm [10] this approach allows agents to learn in a centralized manner during training while executing their policies independently during inference or real world deployment. This methode has been shown to be effective in various MARL environments [6], [11].

III Laser Learning 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 Features

III.2.1 State

The LLE state is composed of a two dimensional grid of cells, where each cell can be either a wall, a gem, a laser beam, laser source, or an exit point. On top of that, a cell can also be occupied by an agent, there is also collision between agent thus each cell is only occupied by one agent at a time. each agent has a unique color and it will also have a exit point of the same color.

III.2.1.1 Empty Cell

An empty cell is the default cell type in the environment. It is a crossable and occupiable cell that does not contain any obstacles or rewards, neither interactable.

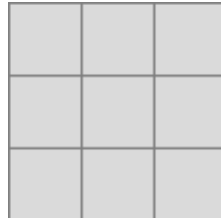


Figure 2: An empty cell inbetween walls and gems in the LLE environment

III.2.1.2 Wall

The wall is a cell that cannot be crossed by agents. It is used to create obstacles in the environment. A wall is not a valid state for an agent to occupy, and is not interactable by agents.

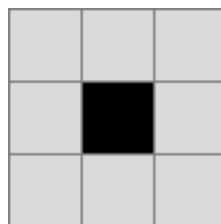


Figure 3: A wall cell inbetween empty cells in the LLE environment

III.2.1.3 Gem

The gem is a ephemeral cell that will be collected by the agent when it occupies the cell. The gem is used to reward the team of agents for their cooperation and to encourage them to explore/guide them through the environment. After being collected the cell will became an empty cell.



Figure 4: A gem cell inbetween empty cells in the LLE environment

III.2.1.4 Laser Beam

The laser beam is a dynamic cell can only exist by the presence of a laser source, it is crossable and occupiable and have special interaction.

The Laser beam possess the same color as the source. It exist continuously unless a agent of the same color occupies a cell that located in a position that is inbetween the laser source and the current cell. When this happens the laser beam will be removed and the cell will become an empty cell. After being removed, the laser beam will only be reactivated when the agent step aside.

A agent that is not of the same color as the laser beam can still occupy the same cell as the laser beam, but it will be killed and removed from the environment.

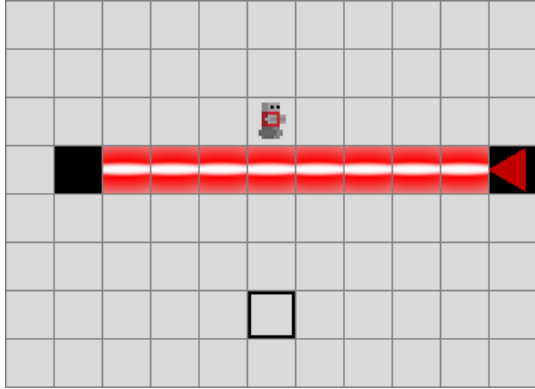


Figure 5: Red laser beam with red agent standing in front

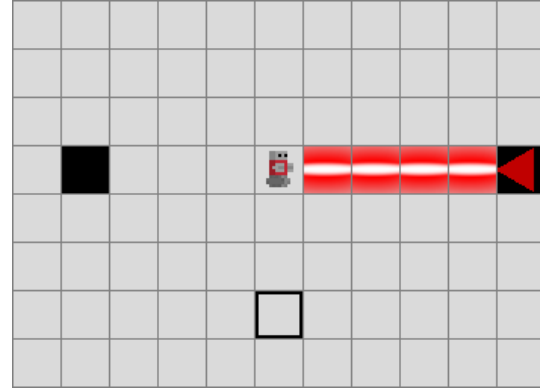


Figure 6: Red agent in the path of the laser beam

III.2.1.5 Laser Source

The laser source is not crossable, not occupiable, and does not interact with agents. It has a color and direction in which every empty cell becomes a laser beam and propagates until it hits a wall or a agent of the same color.

The propagation of the laser beam is continuous and is reapplied to every empty cell in the direction of the laser source instantaneously.

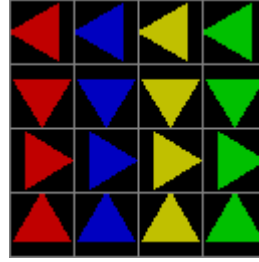


Figure 7: all Laser source orientation in the LLE with different colors

III.2.1.6 Exit Point

The exit point is a special cell that validates the completion of the task for each agent. The game ends when all agents either reach their exit points or are removed from the environment.

When agent reaches its exit point, it is considered successful and will not be able to do other action than “Wait”.

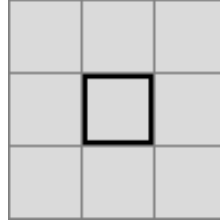


Figure 8: An exit point cell inbetween empty cells in the LLE environment

III.2.1.7 Agent

An agent is a dynamic entity that can move from one cell to another within the environment by executing actions. Agents can only occupy certain types of cells. The objective of the agent is to collaborate with teammates to attain a exit point, each agent has a unique color Figure 9

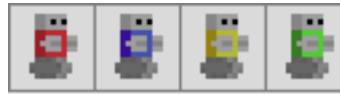


Figure 9: Agents in the LLE environment, each with a unique color

III.2.2 Actions And Transitions

In LLE, agents can perform the following actions:

- Move Up
- Move Down
- Move Left
- Move Right
- Wait (do nothing)

Each action corresponds to a movement in the grid world, allowing agents to navigate through the environment and interact with cells. The action space is discrete, meaning that agents can only choose from a predefined set of actions at each time step.

The transition function in LLE is deterministic, meaning that the outcome of an action is predictable and does not involve any randomness. thus the transition function

$$\mathcal{T}(\cdot \mid s, a)^*$$

will always result in one specific state.

III.2.3 Rewards

The reward function in LLE is only defined by 2 types of rewards:

- Positive reward:
 - Collecting a gem: When an agent occupies a cell with a gem, it collects the gem and receives a positive reward.
 - Reaching an exit point: When an agent reaches its corresponding exit point, it receives a positive reward.
- Negative reward:
 - Being killed by a laser beam: When an agent occupies a cell with a laser beam of a different color, it is killed and set the reward counter to -1.

III.3 Implementation

The core of the LLE is Implemented in Rust with an additional layer of python for simple manipulation and is available on GitHub LLE.

III.3.1 Structure

The structure used at the core level and the python layer are generally the same, with the main difference being that the python layer is designed to be more user-friendly and have a additional feature which is the “observation”


III.3.2 World

The world is an important component of the LLE. It represents the environment as a whole, including the grid, agents, and all other elements. And it is represented by the World struct in Rust:

```

1  pub struct World {
2      width: usize,
3      height: usize,
4
5      grid: Vec<Vec<Tile>>,
6      agents: Vec<Agent>,
7      laser_source_positions: Vec<Position>,
8      lasers_positions: Vec<Position>,
9      gems_positions: Vec<Position>,
10     /// Possible random start position of each agent.

```

 Rust

*defined in Chapter 3.5 Multi-Agents Markov Decision Process


```

11  random_start_positions: Vec<Vec<Position>>,
12  void_positions: Vec<Position>,
13  exits: Vec<Position>,
14  agents_positions: Vec<Position>,
15  wall_positions: Vec<Position>,
16
17  available_actions: Vec<Vec<Action>>,
18  /// The actual start position of the agents since the last `reset`.
19  start_positions: Vec<Position>,
20  rng: rand::rngs::StdRng,
21 }

```

where:

- width and height are the dimensions of the grid world.
- grid is a 2D vector representing the grid world, where each cell is a Tile (e.g., wall, gem, laser beam, etc.).
- agents is a vector of Agent
- laser_source_positions is a vector of positions (tuples) of where the laser sources are located in the grid.
- lasers_positions is a vector of positions of where the laser beams are located in the grid.
- gems_positions is a vector of positions of where the gems are located in the grid.
- random_start_positions is a vector of vectors of positions where the agents can start randomly (this is not used in the current state of the environment).
- void_positions is a vector of positions where the agents can move freely without any obstacles (this is not used in the current state of the environment).
- exits is a vector of positions where the exit points are located in the grid.
- agents_positions is a vector of positions where the agents are located in the grid.
- wall_positions is a vector of positions where the walls are located in the grid.
- available_actions is a vector of vectors of actions available for each agent at a given time.
- start_positions is a vector of positions where the agents start at the beginning of the episode.
- rng is a random number generator used to generate random numbers for the environment.

III.3.3 Observation

The observation in LLE is a simplified representation of the environment that agents can use to make decisions. It is located in the python layer and is a condensed version of the world state. The observation has multiple formats, including:

- Layered
- Flattened
- Partial
- RGB_Image
- ...

where not all formats are interesting for the current research, The most interesting observation format for the current research is the Layered observation, Which is a

3D tensor (3 level of depth) where 2 dimensions represent the grid world and the third dimension represents the different layers of the observation. The layers are split into:

- one layer per agent, where each layer represents the agent's position and its color
- one layer for the walls, where the wall cells are represented by 1 and the empty cells by 0
- one layer for the gems, where the gem cells are represented by 1 and the empty cells by 0
- one layer per colored laser, where -1 represents the laser source, 1 represents the laser beam, and 0 represents an empty cell
- one layer for the exit points, where the exit cells are represented by 1 and the empty cells by 0

III.4 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) [12] or the Hanabi environment [13]. Instead, this environment is designed to incorporate factors such as perfect coordination, interdependence, and zero-incentive dynamics [14].

maybe add
subsec for the
3 factors

III.5 Multi-Agent Markov Decision Process

The model of the environment is based on the Multi-Agent Markov Decision Process (MMDP) [15], a generalization of the Markov Decision Process (MDP) for 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
- $\mathcal{A} \equiv A^1 \times A^2 \times \dots \times A^n$ is the joint action space, where A^i is the set of actions available to agent i^\dagger
- $\mathcal{a} \equiv (a^1, a^2, \dots, a^n) \in \mathcal{A}$ is the joint action of all agents, where a^i is an action of agent i
- $\mathcal{T} : S \times \mathcal{A} \rightarrow \Delta_S$ is a function that gives the probability of transitioning from state s to state s' given a joint action \mathcal{a}
- $\mathcal{R} : S \times \mathcal{A} \times S \rightarrow \mathbb{R}$ is the function that returns the reward obtained when transitioning from state s to state s' given a joint action \mathcal{a}
- $s_0 \in S$ is the initial state
- $s_f \in S$ is the final state

A transition is defined as $\tau = \langle s, \mathcal{a}, r, s' \rangle$ with $r \in \mathbb{R}$.

III.6 Algorithms

Based on the MMDP model, the LLE environment is designed to be used with MARL algorithms that follow the Centralized Training with Decentralized Execution (CTDE) paradigm. The environment is compatible with various MARL algorithms, including Value Decomposition Networks (VDN) and Q-Mix. There are also imple-

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

mentations of Independent Q-Learning (IQL) which can be used for comparison purposes.

III.6.1 Value Decomposition Networks

Value Decomposition Networks (VDN) [16] is a MARL algorithm that leverages the hypothesis of decomposing the joint action-value function into individual value functions for each agent:

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

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.6.2 independent Q-learning

III.6.3 QMix

...

ask yannick
about this

IV Research Questions

The main research question of this thesis is

- How does the performance of a given algorithm change when unknown elements are introduced into the environment

this question can be further divided into sub-questions:

- What metrics can effectively quantify the algorithm robustness to novel elements ?
- Can existing adaptation mechanisms mitigate the impact of unknown elements on algorithm performance ?
- 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?
- 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?

V Evaluation method

The evaluation method will be separated into 2 parts:

V.1 Result comparison

V.1.1 Same model, different environment

The first part of the evaluation will consist of evaluating the performance of the algorithms on a map of similar difficulty as the original LLE map, but with the addition of the lift and lever. The goal is to observe how the algorithms perform in this new environment and whether they can adapt to the new element.

For this part, the need to get a map with equivalent difficulty as the original LLE map is important. The complexity of the map will be determined by the zero-incentive, interdependence, and perfect coordination factors. Given that these factors are not numerically defined, the equivalent difficulty may be subject to subjectivity. Another approach for comparison is to use the number of steps required to reach the goal.

V.1.2 Best model performance

The second part of the evaluation will consist of evaluating the performance of the best algorithm trained on the modified environment and comparing it to the performance of the best algorithm trained on the original environment. The goal is to observe whether the addition of the lift and lever has a significant impact on the performance of the

In the same aspect this evaluation will also consider the training process of the algorithms, including the impact on the convergence speed and the number of training episodes required to reach a certain level of performance.

V.2 Evaluation Metrics

The evaluation metrics will be based on the reward obtained by the team of agents in the environment. The reward is a scalar value that represents the number of gems collected by the agents and the rate of success in reaching the exit point. The reward is calculated as follows:

- The reward is the sum of the number of gems collected by the agents.
- The reward is the number of agents that reached the exit point.

VI Implementation requirements

In order to achieve the objectives of this thesis, some features must be implemented in the LLE environment. These features include the lift and lever, ut also add a new architecture for the environment allowing addtionnal layer which will be needed for the lift. With those new features, 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.

VI.1 Lift

The lift is a terrain 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.

Implementation-wise, the lift need to be added as a new tile type in the environment and possess the following properties:

- It is crossable and occupiable by agents.
- It should be able to linked to a lever.
- It need to be able to have a spatial representation.
- If the lift is not on the same floor as the agent, the agent will not be able to go on the lift.
- The lift should be able to switch floors when the lever is activated.

VI.2 Lever

The lever is a terrain 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.

Implementation-wise, the lever need to be added as a new tile type in the environment and possess the following properties:

- It is crossable and occupiable by agents.
- It need to be able to have a spatial representation.
- If the lever is not activated, the lift will not switch floors.
- Add a mechanism of interaction to be able to activate the lever when an agent is standing on it.

VI.3 Extra dimension

By adding a third dimension to the environment, there is a need to modify the current architecture of the environment to allow for the representation of all features mentioned above. This includes the lift and lever, as well as the additional layer for the observation.

VI.4 Other possibilities

VI.4.1 Proximity channel

A instresting feature that could be added to the LLE environment is a proximity channel. This channel would allow agents to communicate with each other when they are close enough, enabling them to share information in some limited way.

Bibliography

- [1] Y. Molinghen, R. Avalos, M. Van Achter, A. Nowé, and T. Lenaerts, “Laser Learning Environment: A new environment for coordination-critical multi-agent tasks,” in *BNAIC 2023*, in BeNeLux Artificial Intelligence Conference. 2023.
- [2] 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.
- [3] G. Weiss, Ed., *Multiagent systems: a modern approach to distributed artificial intelligence*, 3. print. Cambridge, Mass.: MIT Press, 2001.
- [4] P. Stone and M. Veloso, “Multiagent Systems: A Survey from a Machine Learning Perspective,” Fort Belvoir, VA, Dec. 1997. doi: 10.21236/ADA333248.
- [5] 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.
- [6] R. S. Sutton and A. Barto, *Reinforcement learning: an introduction*, Nachdruck. in Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2014.
- [7] 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.
- [8] J. Achiam, “Spinning Up in Deep Reinforcement Learning,” 2018.
- [9] M. Bowling and M. Veloso, “An Analysis of Stochastic Game Theory for Multi-agent Reinforcement Learning.”
- [10] DMAP 2020 ICAPS, “Factored Value Functions for Cooperative Multi-Agent Reinforcement Learning.” Accessed: Mar. 30, 2025. [Online]. Available: <https://www.youtube.com/watch?v=EsCwDYtBZ8M>
- [11] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized Experience Replay.” Accessed: Aug. 04, 2025. [Online]. Available: <http://arxiv.org/abs/1511.05952>
- [12] M. Samvelyan *et al.*, “The StarCraft Multi-Agent Challenge.” Accessed: Jan. 31, 2025. [Online]. Available: <http://arxiv.org/abs/1902.04043>
- [13] 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.
- [14] 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>
- [15] C. Boutilier, “Planning, Learning and Coordination in Multiagent Decision Processes.”
- [16] P. Sunehag *et al.*, “Value-Decomposition Networks For Cooperative Multi-Agent Learning.” Accessed: Jan. 31, 2025. [Online]. Available: <http://arxiv.org/abs/1706.05296>

-
- [17] A. Tampuu *et al.*, “Multiagent Cooperation and Competition with Deep Reinforcement Learning.” Accessed: Feb. 27, 2025. [Online]. Available: <http://arxiv.org/abs/1511.08779>
 - [18] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A Search Space Odyssey,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017, doi: 10.1109/TNNLS.2016.2582924.
 - [19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition.” Accessed: Feb. 22, 2025. [Online]. Available: <http://arxiv.org/abs/1512.03385>
 - [20] “<https://arxiv.org/pdf/1512.03385>.” Accessed: Feb. 22, 2025. [Online]. Available: <https://arxiv.org/pdf/1512.03385>
 - [21] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.
 - [22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” 1986.
 - [23] C. Gulcehre, K. Cho, R. Pascanu, and Y. Bengio, “Learned-Norm Pooling for Deep Feedforward and Recurrent Neural Networks,” *Machine Learning and Knowledge Discovery in Databases*, vol. 8724. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 530–546, 2014. doi: 10.1007/978-3-662-44848-9_34.
 - [24] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.” Accessed: Feb. 19, 2025. [Online]. Available: <http://arxiv.org/abs/1412.3555>
 - [25] L. Weng, “A (Long) Peek into Reinforcement Learning.” Accessed: Feb. 15, 2025. [Online]. Available: <https://lilianweng.github.io/posts/2018-02-19-rl-overview/>
 - [26] Google DeepMind, “RL Course by David Silver - Lecture 1: Introduction to Reinforcement Learning.” Accessed: Feb. 15, 2025. [Online]. Available: <https://www.youtube.com/watch?v=2pWv7GOvuf0>
 - [27] “Understanding LSTM Networks – colah's blog.” Accessed: Feb. 15, 2025. [Online]. Available: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
 - [28] “The Unreasonable Effectiveness of Recurrent Neural Networks.” Accessed: Feb. 15, 2025. [Online]. Available: <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>
 - [29] “Unsupervised Feature Learning and Deep Learning Tutorial.” Accessed: Feb. 15, 2025. [Online]. Available: <http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/>
 - [30] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang, “QPLEX: Duplex Dueling Multi-Agent Q-Learning.” Accessed: Jan. 31, 2025. [Online]. Available: <http://arxiv.org/abs/2008.01062>
 - [31] T. Rashid, M. Samvelyan, C. S. d. Witt, G. Farquhar, J. Foerster, and S. Whiteson, “QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Rein-

-
- forcement Learning.” Accessed: Jan. 31, 2025. [Online]. Available: <http://arxiv.org/abs/1803.11485>
- [32] Y. Cao, W. Yu, W. Ren, and G. Chen, “An Overview of Recent Progress in the Study of Distributed Multi-Agent Coordination,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 427–438, Feb. 2013, doi: 10.1109/TII.2012.2219061.
- [33] R. Klima *et al.*, “Space Debris Removal: Learning to Cooperate and the Price of Anarchy,” *Frontiers in Robotics and AI*, vol. 5, Jun. 2018, doi: 10.3389/frobt.2018.00054.
- [34] G. Minelli and M. Musolesi, “CoMIX: A Multi-agent Reinforcement Learning Training Architecture for Efficient Decentralized Coordination and Independent Decision-Making.” Accessed: Mar. 30, 2025. [Online]. Available: <http://arxiv.org/abs/2308.10721>
- [35] T. Rashid, G. Farquhar, B. Peng, and S. Whiteson, “Weighted QMIX: Expanding Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning.” Accessed: Mar. 30, 2025. [Online]. Available: <http://arxiv.org/abs/2006.10800>
- [36] G. J. Laurent, L. Matignon, and N. Le Fort-Piat, “The world of independent learners is not markovian,” *International Journal of Knowledge-based and Intelligent Engineering Systems*, vol. 15, no. 1, pp. 55–64, Mar. 2011, doi: 10.3233/KES-2010-0206.
- [37] “Agent Laboratory: Using LLMs as Research Assistants.” Accessed: Mar. 15, 2025. [Online]. Available: <https://agentlaboratory.github.io/>
- [38] A. Vaswani *et al.*, “Attention Is All You Need.” Accessed: Apr. 07, 2025. [Online]. Available: <http://arxiv.org/abs/1706.03762>
- [39] “Zotero | Your personal research assistant.” Accessed: Apr. 18, 2025. [Online]. Available: <https://www.zotero.org/settings/storage?ref=usb>
- [40] F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis, “Optimal and Approximate Q-value Functions for Decentralized POMDPs,” *Journal of Artificial Intelligence Research*, vol. 32, pp. 289–353, May 2008, doi: 10.1613/jair.2447.
- [41] L. Busoniu, R. Babuska, and B. De Schutter, “A Comprehensive Survey of Multi-agent Reinforcement Learning,” 2008, Accessed: Apr. 18, 2025. [Online]. Available: <https://repository.tudelft.nl/record/uuid:4c7d3b49-06fc-400c-923e-3903b8d230fe>
- [42] “<https://jair.org/index.php/jair/article/download/10952/26090>.” Accessed: Apr. 21, 2025. [Online]. Available: <https://jair.org/index.php/jair/article/download/10952/26090>
- [43] D. Bloembergen, K. Tuyls, D. Hennes, and M. Kaisers, “Evolutionary Dynamics of Multi-Agent Learning: A Survey,” *Journal of Artificial Intelligence Research*, vol. 53, pp. 659–697, Aug. 2015, doi: 10.1613/jair.4818.
- [44] V. Elango, N. Rubin, M. Ravishankar, H. Sandanagobalane, and V. Grover, “Diesel: DSL for linear algebra and neural net computations on GPUs,” in *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning*

-
- and Programming Languages*, Philadelphia PA USA: ACM, Jun. 2018, pp. 42–51. doi: 10.1145/3211346.3211354.
- [45] A. Singh, Y. Sarita, C. Mendis, and G. Singh, “ConstraintFlow: A DSL for Specification and Verification of Neural Network Analyses.” Accessed: May 05, 2025. [Online]. Available: <http://arxiv.org/abs/2403.18729>
- [46] R. Bellman, “A Markovian Decision Process,” *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957, Accessed: Apr. 25, 2025. [Online]. Available: <https://www.jstor.org/stable/24900506>
- [47] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.
- [48] M. Bowling and M. Veloso, “An Analysis of Stochastic Game Theory for Multi-agent Reinforcement Learning,” p. , 2001.

VII Appendix

VII.1 Notations

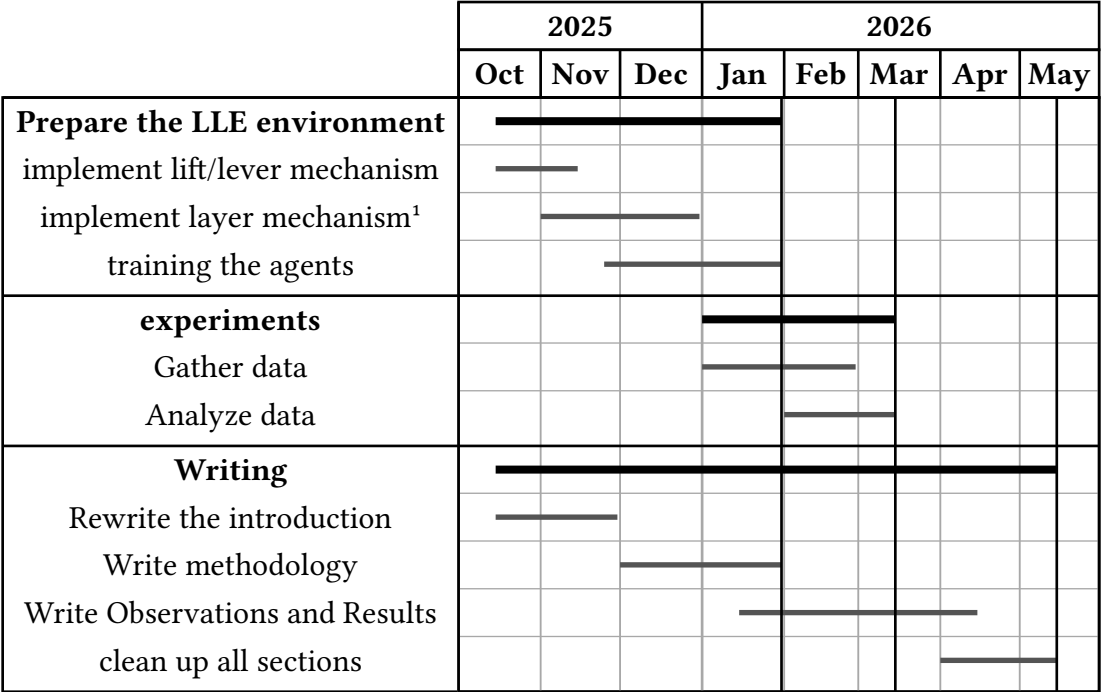
Notations	Description
\approx	approximately equal
\in	is an element of (e.g., $3.2 \in \mathbb{R}$)
$[a, b)$	the interval from a to b , where a is included and b is excluded
$\sum_{a \in A} a$	the sum of all elements a in the set A
$f : X \times Y \rightarrow Z$	a function f with a domain from the set $X \times Y$ 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
\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}_x	the expected value over the value x
γ	the discount factor, which is a real number in the interval $[0, 1]$

VII.2 Acronyms

Acronym	Full Name
AI	Artificial Intelligence
DAI	Distributed Artificial Intelligence
DPS	Distributed Problem Solving
MAS	Multi-Agent Systems
MAL	Multi-Agent Learning
RL	Reinforcement Learning
SL	Supervised Learning
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
CTDE	Centralized Training with Decentralized Execution
VDN	Value Decomposition Networks
IQL	Independent Q-Learning
MMDP	Multi-Agent Markov Decision Process
SMAC	StarCraft Multi-Agent Challenge
LLE	Laser Learning Environment

Acronym	Full Name
MARL	Multi-Agent Reinforcement Learning

VII.3 Time Plan



Code ready for experiments
Jan 31

First draft of the thesis
Mar 15

Final work
May 15