Автономная некоммерческая организация высшего образования «Университет Иннополис»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)
по направлению подготовки
09.03.01 - «Информатика и вычислительная техника»

GRADUATION THESIS (BACHELOR’S GRADUATION THESIS)
Field of Study
09.03.01 – «Computer Science»

Направленность (профиль) образовательной программы «Информатика и вычислительная техника»
Area of Specialization / Academic Program Title: «Computer Science»

Тема / Topic	**Изучение поведенческих стратегий для системы с несколькими роботами в среде хищник-жертва с использованием обучения с подкреплением /** **Learning behavioral strategies for a multi-robot system in a predator-prey environment using Reinforcement Learning**

Работу выполнил / Thesis is executed by	**Хани Хамед Али Эль Анвар Хамед /** **Hany Hamed Ali Elanwar Hamed**	подпись / signature
Руководитель выпускной квалификационной работы / Supervisor of Graduation Thesis	**Александр Климчик** **Alexandr Klimchik**	подпись / signature

Консультанты / Consultants	**Стефано Нольфи / Stefano Nolfi**	подпись / signature

Иннополис, Innopolis, 2022

Contents

List of Tables

List of Figures

Abstract

[redacted]

Acknowledgement

[redacted]

Chapter 1

Introduction

1.1 Multi-Agent System

Many real-life scenarios contain multiple of agents that acting to achieve a goal cooperatively or competitively. Therefore, Multi-Agent System (MAS) is an interesting field to research and study [1].

MAS is a system composed of multiple decision making agents exist in an environment, these agents can interact with the surrounded environment as well as interacting with the other agents [2]. These agents can be either virtual or physical agents represented by a robot, such a system is known as a Multi-Robot System (MRS) [3], [4].

The goal for the agents is to successfully achieving a task through cooperation or competition with other agents. Therefore, MAS studies the interaction between the different agents in the system. An interesting direction in MAS is the Collective Intelligence (CI) that concerns with the emergence of the agents' behaviors through the cooperation and the competition between the agents [5].

MAS has gained huge attention in research and in industry [2] due to the presence of such systems in our daily life. Thus, MAS has been discussed in various fields including robotics [6], civil engineering [7], electrical engineering [8], computational biology [9], protein simulation [10], and many other fields. Some examples for multi-agent systems are as following:

- Multiplayer games contain more than one player that follows the rules and compete against each other, and at the same time cooperate with the teammates. Recent advances in research exposed learning-based methods that were used to produce agents that performed better than humans. These solutions managed to explore complex behaviors in complex games with high-dimensional state and action spaces, for example: Go [11] and StarCraft 2 [12].

Figure 1.1: The left image shows the Go game such that there are two players with black and white stones. The right figure shows StarCraft 2 with two armies of units for the red (Human army) and blue (Protoss army) players

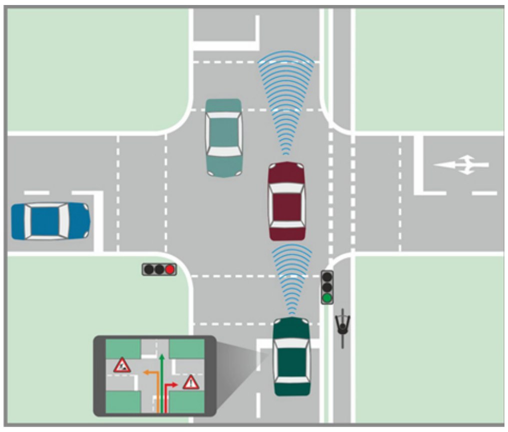
- Traffic management problem contains modeling multiple of vehicles and different routes while each vehicle tries to minimize the time taken to go from a certain location to another location. [13]

- Warehouse management using a fleet of robots instead of the workers. This enables fast and safe operations inside the warehouse. Amazon has developed such system which is effectively working in multiple of places around the world [14].

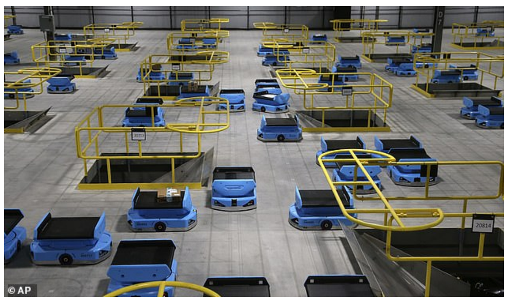
- Autonomous goods delivery using a network of mobile robots, this enables the fast and autonomous delivery from point to point. Yandex has promoted its own robots to replace the normal supply chain between the restaurants and the customers using robots [15].

- Objects transportation using multiple of robots. In some cases, it is hard to move an object from one place to another using a single robot, but it is possible to move the object through the cooperation of multiple robots [16], [17].

- Human-robot cooperation where the robots are cooperating with humans to achieve a common goal. Such a cooperation enables achieving hard tasks for humans with the help of robots [18].

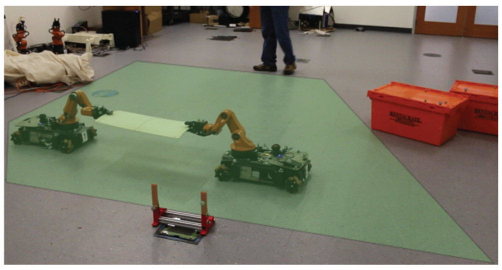


(a) Traffic management problem. Source: [19]



(b) Amazon Warehouse robots. Source: [20]

(c) Yandex's autonomous goods delivery robot



(d) Object transportation. Source: [17]

Figure 1.2: Examples for the real-life applications of multi-agent and multi-robot systems

1.2 Predator and Prey

Predation is a biological process that includes two populations: the predator and the prey. The predator is always searching on and consuming the prey's population in order not to die, on the other hand, the prey is trying to escape and survive to live [21]. Each population shares the same goal of survival, however, they achieve it by a different set of behaviors that entirely oppose the other population's goal. Thus, in order to achieve its own goal, the population must oppose the other population's goal.

Predator-prey problem is considered as a multi-agent system in which there are two populations/teams in the environment and each population can consists of multiple agents [22]. Additionally, predator-prey problem appears in many natural situations, for example, the lion-zebra environment such that the lion is trying to run after the zebra to eat while the zebra is trying to run and survive. Therefore, it draws much attention to study different aspects of the problem such as the growth rate of the populations and the global dynamics in biology [23]. In this thesis, we are interested in studying the behaviours emergence for the agents to archive their goals while evolving progressively through the simulation time using a learning-based method called Self-Play (SP) reinforcement learning [24].

Figure 1.3: A Lion (predator) is chasing a zebra (prey). Source: [25]

Each population tries to beat the other population. Therefore, the population must always evolve and learns behaviors that is performing better than the other population's behavior and at the same time the other population is learning an improved behavior as well. This makes the learning problem dynamically changing due to the behaviors' evolution. Our main interest in predator-prey problem is around the behavior emergence while the progressive evolution for both populations.

We have chosen predator-prey environment to test the implemented methods in this thesis due to be considered as a potential benchmark environment for multi-agent systems [26]–[28]. Additionally, there are similar problems to predator-prey scenario that we present in Table I.

Table I: This table illustrates problems similar to the predator-prey problem and shows the different used terminologies.

Problem name	**1st population name**	**2nd population name**	**Goal**	**References**
Predator-prey	Predator	Prey	The predator is chasing the prey, and the prey is running away from the predator.	[21], [26]
Hide-seek	Hider	Seeker	The hider is hiding from the seeker in places or behind objects that situated in the environment while the seeker is trying to find the hider	[29], [30]

Pursuit-evasion games	Pursuer	Evader	Similar to predator and prey environment, such that the pursuer and the evader act similar to the predator and the prey, respectively. Furthermore, it is a family of problems in game theory and computer science that projects the predator-prey problem in a mathematical framework regarding the agents' behaviors.	[31]–[34]
Attack and defense	Attacker	Defender	The defender has resources to secure, while the attacker attacks these resources to use them.	[35], [36]

All these problems shares the same foundations that they are fundamentally zero-sum games and adversarial games. These class of problems have been studied extensively in literature of the game theory field [37]–[39]. Game theory is a branch of mathematics which is concerned with modeling and analysing the agents' strategies in an environment in which the outcomes of the agent are coupled with the other agents' strategies [37].

Zero-sum games is a definition from game theory that describes the games or environments that has multiple of agents and each agent's gain is other's loss such that the summation of all the agents' payoff is always equal to zero [38]. Moreover, adversarial games emphasize the fact that agents are adversarial to each other in achieving their goals. Thus, in order to win the agent has to beat the other agents. In addition, there can be competition besides the cooperation from the agents of the same population/team.

Game theory provides powerful means to solve zero-sum games under certain assumptions regarding the dimensions of state and action spaces, the game dynamics, and the observability of the states, such assumptions do not exist in many realistic situations [40]. Game theory and learning-based methods (e.g. Reinforcement learning) are often intersected to solve multi-agent environments [41], [42]. However, in this thesis, we are only concerned with self-play methods to create an auto-curricula to evolve the agents' emergent behaviors while having no prior assumptions about the game dynamics. Additionally, self-play methods enable exploring different complex behaviors through learning via the interaction with the environment and the other agents.

Predator-prey has multiple applications in the real-life, however, many works discussed its usage in the military domain which we are completely opposed to use this research in that field. For the civil applications, predator-prey and pursuit-evasion games have been discussed to be used for search and rescue operations using mobile robots [43].

1.3 Robot Learning

Robot learning is a research field that describes the intersection between machine learning and robotics. Robot learning is concerned with studying the techniques and the algorithms that enable robots to achieve difficult tasks through using the data and learning how to solve it optimally.

We present some examples of using learning-based approaches in the robotics field:

- Using deep neural networks for computer vision in self-driving cars to detect and segment the objects for safe driving [44]
- Improving the human-robot interaction by using natural-language processing models [45], [46].
- Enhance the autonomous drone's control for racing purposes to perform fast and complex maneuvers [47]
- Exploring the usage of data-driven controllers using Koopman operators [48].
- Using demonstrations from humans to enable the robot to learn how to do the task [49], [50].
- Building much realistic and faster physics simulators for robotics using deep neural networks [51], [52].

1.4 Reinforcement Learning (RL)

In recent years, researchers draws a great attention to Reinforcement Learning (RL) due to the impressive results in multiple of fields [53]. For example, reinforcement learning has been extensively researched and studied to be used with robotics applications (Autonomous drones [47], manipulators [54], [55], legged robots [56], and mobile robot navigation [57]). Recently, RL has recently achieved successful results to control nuclear fusion plasma [58]. Additionally, RL has been used to achieve a breakthrough in computer games by generating agents with strategies superior to human players [11], [12].

RL is a branch of machine learning that is concerned with solving decision-making problems that include receiving feedback signal through the interactions between the decision-making agent and the other elements exists in the agent's environment. RL consists of two main elements: the agent, and the environment [59].

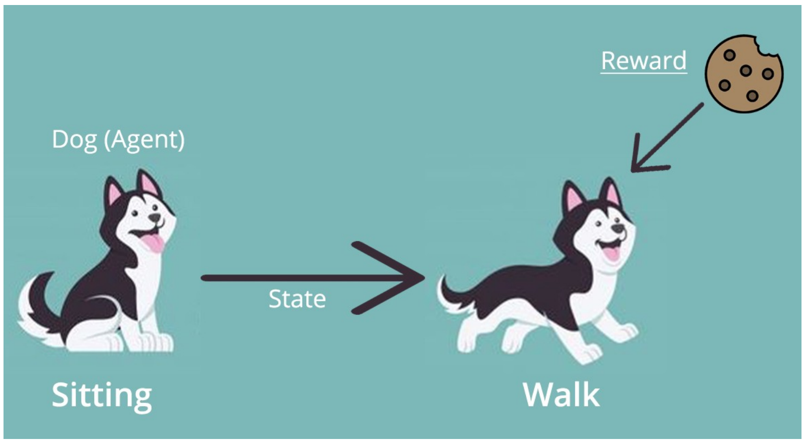


Figure 1.4: An example for a reinforcement learning environment such that the dog is the agent, the reward is the cookie, and the environment includes the interaction with the human. Source: [60]

Machine learning is a branch of artificial intelligence (AI) that is concerned with the study of algorithms that use the data to create an intelligent agent which acts optimally based on the input. Therefore, machine learning include three main branches:

- Supervised learning (SL) that is learns from a labeled dataset which means knowing the results' ground truth.
- Unsupervised learning that is using a dataset without having the ground truth such that it learns without the need for a supervision.

- Reinforcement learning (RL) is concerned of learning through the interaction without the necessity of having a dataset to learn from, instead it learns from the data collected by the interaction of the agent with the environment.

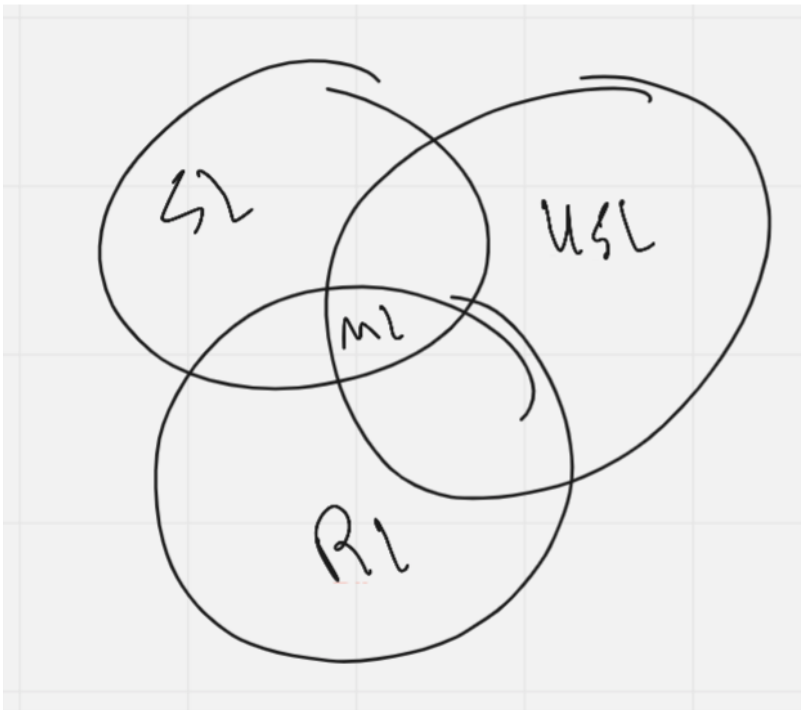


Figure 1.5: [Tmp] Reinforcement learning, supervised learning, and unsupervised learning as parts of machine learning

1.5 Learning in multi-agent systems

Having multiple agents in an environment makes it a highly dynamic and non-deterministic. Furthermore, defining an optimal strategy for every agent is a difficult task, thus, learning-based approaches has been introduced [61]. Moreover, the success of deep reinforcement learning motivated the research direction that is known by Multi-Agent Reinforcement Learning (MARL) [41]. MARL is concerned with the applying reinforcement learning with multi-agent systems such that the agents are learning through the interactions with the environment and the other agents. Furthermore, MARL is an open research field with multiple challenges such as: non-stationary environments, open multi-agent systems, limited access to the opponent information, convergence guarantees, mutli-agent credit assignment, and computational resources, such challenges and current solutions have been extensively discussed in [62], [63].

Using learning-based method with multi-agent system gives advantage to evolve complex behaviors for the agents that not explored by other methods. Therefore, we are motivated in this thesis to explore a class of MARL methods that is known by Self-Play (SP) [24]. In SP, single-agent RL algorithm is used and the agents are training in an alternative manner against the old history of other agents. SP does not require having experienced agents to train against as a baseline, instead it builds its auto-curricular for the agents and results in emerging complex behaviors to solve the task [64]. Self-play is mostly used in competitive environments where the agents competes against each other, additionally, we have selected predator-prey environment as the base competitive environment to evaluate our work in this thesis.

1.6 Applications

1.7 Research questions

At the beginning of this thesis, we have imposed 3 main research questions as follows:

TODO: Specifiy in more details the research questions and link it with the starting point of Bansal et al as in Bloembergen's thesis (Analyzing RL algorithms)

TODO: add explanation for the motivation behind each research question and why we present each question

- What are the current solutions for predator-prey? and what are the shortcomings/disadvantages of these methods?
- How can reinforcement learning be used to provide a progressive improvement for agents in a predator-prey environment?
- How can we evaluate and analysis the results of agents in a predator-prey environment while all agents are evolving through the training time? And how can we evaluate and say that a specific training method is better than another method?

By the end of the thesis, we will be able to conclude the answers for these questions through the discussed implementations and results. As well as explaining the extensions of these research questions for a future work in this direction.

TODO: how the RQs applied with the real tasks and important for it.

1.8 Contributions

- Self-play in predator-prey problem
- New environment for predator-prey (Based on PyBullet-drones) as PZ and Evo are already made for Predator-prey environments
- Self-play framework and open-source code with easy way to change multi-agent environments to self-play environments

- Population-based method

Hopefully ensemble-based method

- Master tournament results with crosstest

  TODO: Contributions then explain how it is unique and how it improved

1.9 Thesis Outline

The rest of the chapters paves the way to answer the previously stated research questions and explain the contributions made by the authors. Therefore, the following chapters are structured according to the following order

- Chapter 2 presents an extensive literature review for the related work and relevant literature in the following fields: Reinforcement Learning (RL) including breif review and explanation for Markov Decision Process (MDP), Deep Reinforcement Learning (DRL), and Multi-Agent Reinforcement Learning (MARL). Moreover, it presents a sufficient introduction for game theory and types of games related to predator-prey problem. Furthermore, it describes the basic fundamental concepts of Self-Play (SP) training algorithm.

- Chapter 3 elaborates the problem statement and explains the environment assumptions and settings used in this thesis. Furthermore, it presents the methodology used to solve the predator-prey problem in a setting of 1 predator versus 1 prey (1 vs 1) through self-play algorithms. Moreover, it presents the fundamentals and description for basic self-play algorithm and its variants that are considered a part of our contributions. Furthermore, that chapter describes the evaluation metrics and methods used to interpret our experiments and results that shown in the rest of the thesis. On top of that, this chapter details the assumptions and the tools used to implement the described methods and the justification for it.

- Chapter 4 presents the obtained results during the simulation and demonstrates the justifications for these results.

- Chapter 5 reports the observed limitations and difficulties in this work and this research direction. We see the importance of this chapter in order to show the remaining research gaps in this direction that can motivate future works for other students and researchers. Furthermore, it elaborates thoughts and ideas for future works that is intended to be continued after this work.

- Chapter 6 concludes the thesis' outcomes and presents a summary for the achieved goals during this thesis.

- Chapter A presents an extra overview for the predator-prey problem with details from the literature. This chapter is closely related to chapter 5 and is considered as a continuation for that chapter.

- Chapter B details the implementation key points and explains how to use the implemented methods with further explanation for all the parameters that were used in the implementation and how to tune them. Additionally, this chapter provides the details on how to modify the code to test it on another not discussed environment.

- Chapter C presents a subjective point of view and personal advice for the reader on how to conduct a research and useful tools in research. Moreover, it narrates the failures and success during this thesis to motivates the reader who has a similar topic.

Chapter 2

Background and Literature Review

In this chapter, we introduce the relevant research work to our thesis aims. This thesis focus on the research based on using self-play reinforcement learning methods to enable the agents learning behavioral strategies automatically without the need for a training dataset from experts. Additionally, we pay a close attention to the related methods to our interest that are used in a predator-prey environment. Moreover, in this chapter we present the necessary background to understand the problem formulation and pave the way for the methodology chapter 3. Thus, this chapter is organized in the following order"

- Section 2.1 establishes the necessary understanding for reinforcement learning problem formulation and introduces the fundamental literature. Furthermore, we provide a basic understanding for the State-of-The-Art (SoTA) reinforcement learning algorithms. Additionally, we show the current efforts in the literature for using reinforcement for multi-agent systems.

- Section 2.2 briefly presents a foundation for describing the predator-prey environment in a game-theoretic formulation and presenting the terminology used in game-theory for describing similar problems. Moreover, this section discusses the relevant work to the predator-prey problem from the point of view of game-theory.

- Section 2.4 outlines briefly the origins of self-play methods from game theory and demonstrates the current work related to self-play reinforcement learning.

- Section 2.5 summarizes the main findings in this chapter, provides the main gaps in the literature.

2.1 Reinforcement Learning (RL)

Reinforcement Learning (RL) is a branch of machine learning that focus on creating an intelligent decision-making agent able to learn from experience and the interaction with some environment. The fundamental idea of reinforcement learning is centered around that the agent is interacting with an environment and the agent is given a reward based on how good the action was at the current moment without specifying how exactly the task should be achieved and trying to maximize the total reward [59].

Reinforcement learning enables the decision making agent to learn without the necessity of having pre-collected data whether it is labeled as in supervised learning or not labeled as in unsupervised learning. RL allows the agent to learn by selecting a decision, observe how the decision will affect the state of the agent's world and how this will affect the total reward collected by the agent, then try to optimize the decision-making process to select a better decision next time [65].

In the recent years, reinforcement learning draws much attention due to the advancements in the field of deep learning with reinforcement learning that enabled creating agent that surpassed the humans in games like Atari [66] and challenging games like Go [11]. Furthermore, the improvements in the hardware have helped to accelerate the training process that speed up and encouraged the research in the direction of Deep Reinforcement Learning (DRL) to create scalable, faster and efficient algorithms [67].

2.1.1 Agent-Environment

In RL, we have 2 main elements for the problem: the agent and the environment. The agent is considered the learner and the decision maker, while the environment is which the agent is interacting with. Thus, given a current state of the environment and the agent, the agent takes an action, then the agent will have a new state based on the dynamics described by the environment. Figure 2.1 shows this interaction between the environment and the agent. Furthermore the figure shows the reward the agent gets by taking such an action in that state.

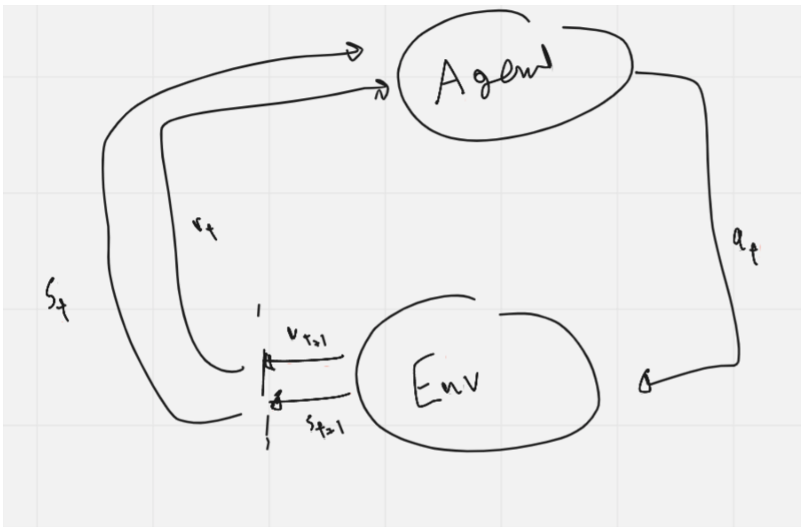


Figure 2.1: [tmp] Reinforcement learning environment and agent

With each action, the agent receives a reward, the RL agent tries to maximize the total reward. However, the agent does not know which action leads to higher rewards, but it learns that through experience.

2.1.2 Markov Decision Process (MDP)

Markov Decision process (MDP) is considered the classical formalization for the sequential decision making [65]. It is a mathematical framework that enables formulating and tracking the states, the actions and the results of these actions (i.e. next state, and the reward). Therefore, reinforcement learning problem can be formalised as an MDP.

In this case, formalizing as an MDP enforce the Markov property in which the future is independent on the past as it is only depends on the present (i.e. The states are memory-less, contain all the relevant information about the past, and knowing more about the previous states does not affect the transition to the next state) [65].

Definition 1. Markov property is a property that defines an important aspect in the transition model for MDPs such that the future is independent on the past given the present.

$$\mathbb{P}(S_{t+1} \mid S_t) = \mathbb{P}(S_{t+1} \mid S_1, S_2, ..., S_t)$$

such that S_{t+1} is the future state, S_t is the present state, and S_i is the past states $\forall\, i \in \{1, 2, ...t - 1\}$

Thus, it is important to maintain the Markov property in the RL problem as it is an assumption that many single-agent RL algorithms are built based on [62].

We have reviewed the literature and have adapted the definitions in this section which set the fundamentals of the reinforcement learning field from [53], [65], [68]–[70]. Therefore, Markov Decision process and its variants are defined formally as following:

Definition 2. Markov Decision Process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a set of all possible states.
- $\mathcal{A}$ is a set of all possible actions that the RL agent can take.
- $\mathcal{P}$ is the state transition dynamics. It maps the state-action-state pair to a probability distribution.

$$\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$$

  Furthermore, it expresses the likelihood of ending up to a new state (s') given the current state (s) and the selected action (a):

$$\mathcal{P}_{ss'} = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$$

- $\mathcal{R}$ is a reward function that describes the immediate reward given based on the current state, selected action, and the new state.

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

- γ is the discount factor that expresses the importance of the current rewards relative to the old rewards. $\gamma \in [0, 1]$

The following figure shows an example of an MDP with the transitions between the states (s) by taking actions (a) and the received rewards (r)

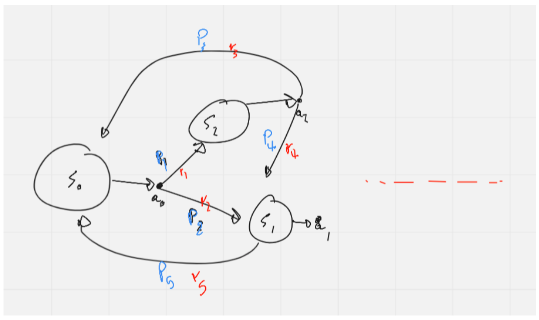


Figure 2.2: TMP remove it combine this figure with the following figure later

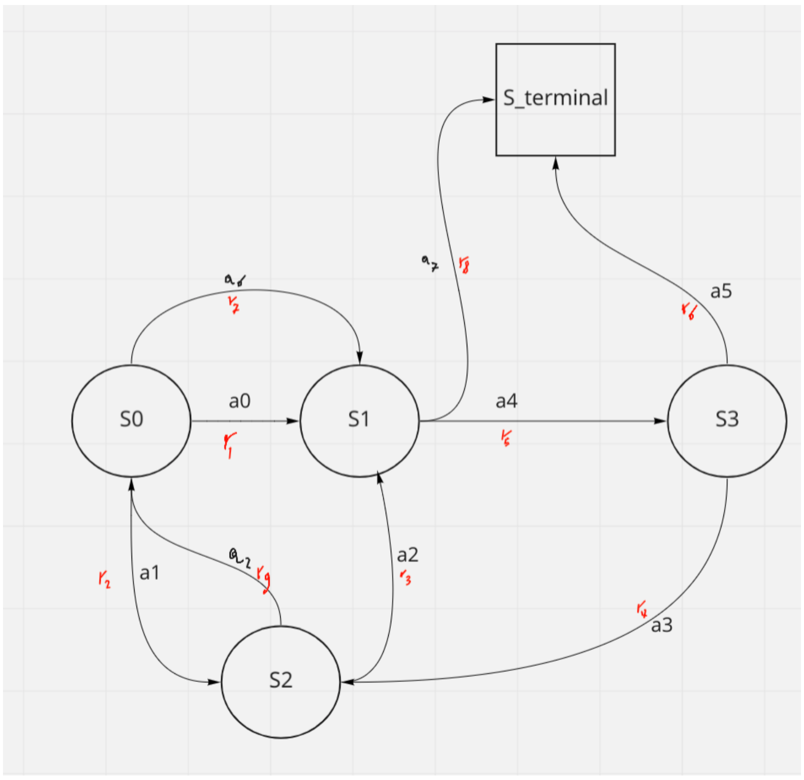


Figure 2.3: MDP example tmp

Taking an action from each state has a distribution for going to the next state, moreover, the agent receives a reward defined by the reward function R

Another terminology that we need to deinfe is the Partial Observable Markov Decision Process (POMDP). POMDP is an extension for MDP such that it is an MDP with hidden states (i.e. A hidden Markov model with actions) [69].

It deals with the cases when the agent is not able to sees the true full state of the environment, instead it receives an observation (a partial state). POMDP enables to define the problem to be much closer to the real-life problems where usually the agents are not able to observe the whole environment.

Definition 3. Partial Observable Markov Decision Process (POMDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{Z}, \mathcal{R}, \gamma \rangle$, such that $\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma$ are the same as in MDP.

- O is a set of observations that the agent can see.

- Z is an observation function. It maps the state-action-observation pair to a probability distribution.

  $$\mathcal{Z} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \rightarrow [0, 1]$$

  It defines the probability of receiving an observation (o) given the new state (s') and the current action that the agent takes (a).

  $$\mathcal{Z}^{a}_{s'o} = \mathbb{P}(O_{t+1} = o \mid S_{t+1} = s, A_t = a)$$

  Notice that the agent cannot access the states but only can see the observations.

  TODO: Add motivation for markov games

  TODO: Formulation of Markov Games and connection to pred-prey

Definition 4. Markov Game is ....

2.1.3 Returns and Episodes

In the RL problem, the agent is taking the actions in a sequential manner during an episode of finite number of steps (episodic task) or during infinite number of steps (continuous task). The episodic tasks has a termination state defined by a specific criteria, thus it has a finite number of steps. The episode starts with an initial state and terminates with a terminal state. During the episode, the agent takes an action, then receives an observation and a reward, in this way, the agent is collecting the experience to learn from it.

The goal of RL agent is to maximize the cumulative reward in the episode. We are interested in maximizing the expected return such that the return (G_t) is the summation of rewards after t step.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T = \sum_{k=0}^{T} R_{t+1+k} \tag{2.1}$$

such that T is the final time step and from MDP definition (2): $R_i = R(s_i, a_i, s_{i+1})$, $s_i \in S$ is the state at i time step (current state), $a_{t+1+k} \in A$ is the action at i time step, and $s_{t+k+2} \in S$ is the state at *i+1* time step (next state) $\forall\, i \in \{0, ..., T\}$.

Generally, in continuous tasks T=∞ as the tasks have infinite number of steps. Therefore, the discount factor is introduced with the return function to make the summation converges to a finite value, and to give importance to the immediate rewards over the past rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \tag{2.2}$$

where γ is the discount factor $\in [0, 1]$. Note that if $\gamma = 1$ in the continuous task, the return may diverge. The return function can be written in a recursive way as following:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...) \\ &= R_{t+1} + G_{t+1} \end{aligned} \tag{2.3}$$

2.1.4 Value functions

The agent has a policy that it selects an action that maximizes the total reward. The agent optimizes this policy by learning which actions in which states are good to choose.

Definition 5. A policy (π) is a function that maps from the current state to the action that the agent should take.

$$\pi : \mathcal{S} \rightarrow A$$

Furthermore, the policy defines the probability distribution of the actions given the states as following:

$$\pi(a \mid s) = \mathbb{P}(A_t = a \mid S_t = s)$$

where a is the selected action and s is the agent's state at t time step.

The policy is parameterized and denoted as the following mathematical notation: π_θ such that θ are the parameters. However, π and π_θ are being used interchangeably and indicates the same meaning.

The total reward has no direct formulation to the relation between the states and the actions from the policy. Therefore, the state-value and action-value functions are introduced, almost all reinforcement learning algorithms estimates at least one of them to be used in the learning algorithm indicating the goodness of a specific state and a specific action.

Definition 6. The state-value function $(V_\pi(s))$ is a function that maps the state s under a policy π to a real value which is the expected return when starting from state s and following the policy π.

$$V_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \mid S_t = s], \forall\, s \in S \tag{2.4}$$

$\mathbb{E}_\pi[\cdot]$ represents the expected value of a random variable such that the agent follows the policy π, and t is the time step.

The state-value function indicates how good to be in a given state (s) and follow the policy (π).

The continuation of the derivation is described in details in Appendix A.2 (equation A.2)

Definition 7. The action-value function $(Q_\pi(s, a))$ is a function that maps the state and the action to a real value which is the expected return when starting from state s and taking action a and then following the policy π

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \mid S_t = s, A_t = a], \forall (s, a) \in S \times A \tag{2.5}$$

The action-value function expresses how good taking an action (a) while being in state (s) then following the policy (π)

The continuation of the derivation is described in details in Appendix A.3, Eq. (A.3)

Definition 8. The advantage function ($A_\pi(s, a)$) is the difference between the action-value and state-value functions following the policy π

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{2.6}$$

The advantage function intuition relies in answering "How a particular decision contributed to the overall expected return?" It shows a value for what happened taking a specific action against the expected return of a state.

Definitions 6 and 8 defines the state-value and action-value functions, and the recursive derivations shown in A.3 and A.2 are known as the Bellman equations. These equations simplifies the computation of the value functions, by breaking down the problem into sub-problems and recursively solves the sub-problems and propagate the solution for the final solution.

Following up from the Bellman equations, the state-value function ($V_\pi(s)$) and action-value function ($Q_\pi(s)$) can be defined as functions of each other and are known as the Bellman expectation equations. These equations shows the relation between the state-value and action-value functions. The relation is defined as following:

$$V_\pi(s) = \sum_{a \in A(s)} \pi(a \mid s) Q_\pi(s, a) \tag{2.7}$$

$$Q_\pi(s, a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} \mathbb{P}(s', r \mid s, a)[r + \gamma V_\pi(s')] \tag{2.8}$$

The full derivation is detailed in A.4. The relation can be visualized in the following backup diagrams

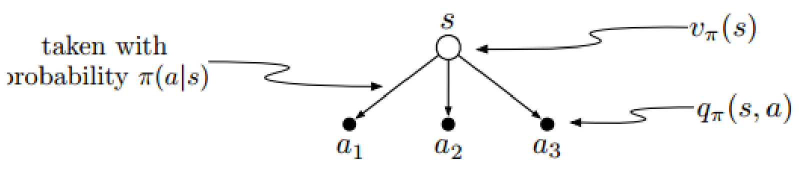


Figure 2.4: [tmp] State-value and action-value functions backup diagram

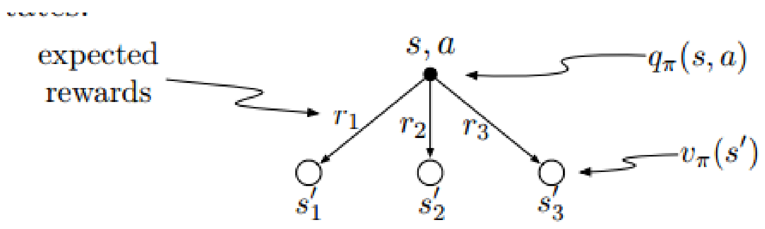


Figure 2.5: [tmp] Action-value and state-value functions backup diagram

These backup diagrams illustrate the mathematical equations 2.7 and 2.8 and shows the relation between the value functions.

To solve the RL problem, the agent needs to find a policy that achieves the maximum possible cumulative reward in the episode.

Therefore, a policy π is considered to be better than or equal to π' if and only if the expected return following π is greater than or equal to the expected return following π' for all the possible states which means

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \,\forall\, s \in \mathcal{S}$$

Thus, we can define the optimal policy as following:

Definition 9. The optimal policy (π^*) is a policy that is better than all other policies and maximizes the value functions

$$\pi^* > \pi, \,\forall\, \pi$$

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, V_\pi(s)$$

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, Q_\pi(s, a)$$

Thus, we can define **the optimal value functions** ($V^*_\pi(s)$, $Q^*_\pi(s, a)$) as following:

$$V^*_\pi(s) = \max_\pi V_\pi(s) \tag{2.9}$$

$$Q^*_\pi(s, a) = \max_\pi Q_\pi(s, a) \tag{2.10}$$

The Bellman optimality equations can be defined in the same manner as the derivation of Bellman equations and using the equations 2.9 and 2.10 and can be defined as following:

Definition 10. The Bellman optimality equations .....

(2.11)

(2.12)

The full derivation .....

2.1.5 Classification of RL algorithms

Reinforcement learning algorithms has been classified into many classes based on different aspect. Multiple works in the literature have discussed these classifications in details [59], [67], [71]. Figure 2.7 shows the taxonomy of reinforcement learning algorithms.

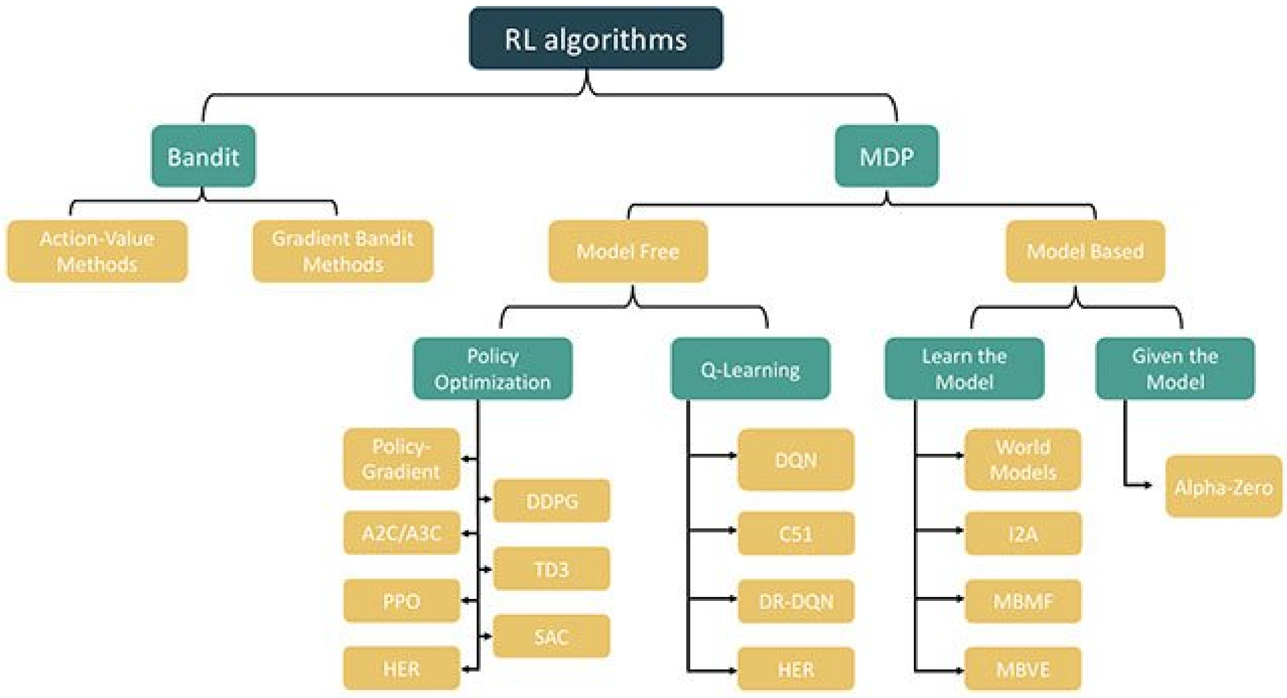


Figure 2.6: [Tmp] Reinforcement learning taxonomy

As it can be noticed from Fig. (2.7), RL algorithms that are based on MDP formulation are divided into to two classes of model-based and model-free. Model-based RL is depending on the environment, whether the goal is to learn the environment model or the model is already known and the agent is using this knowledge. On the other hand, model-free RL does not depend on the knowledge about the environment during the training. In this thesis, we are interested in the model-free algorithms as during the training we do not use any prior knowledge of the environment or trying to learn the environment dynamics.

Another important aspect to classify RL algorithms is on-policy and off-policy algorithms. On-policy RL algorithms uses samples from the same policy that the agent is optimizing, while off-policy uses samples from a different policy rather than the target policy for the optimization. In this thesis, we are using Proximal Policy Optimization (PPO) [72] which is an on-policy algorithm, Soft Actor-Critic (SAC) [73] as an off-policy algorithm.

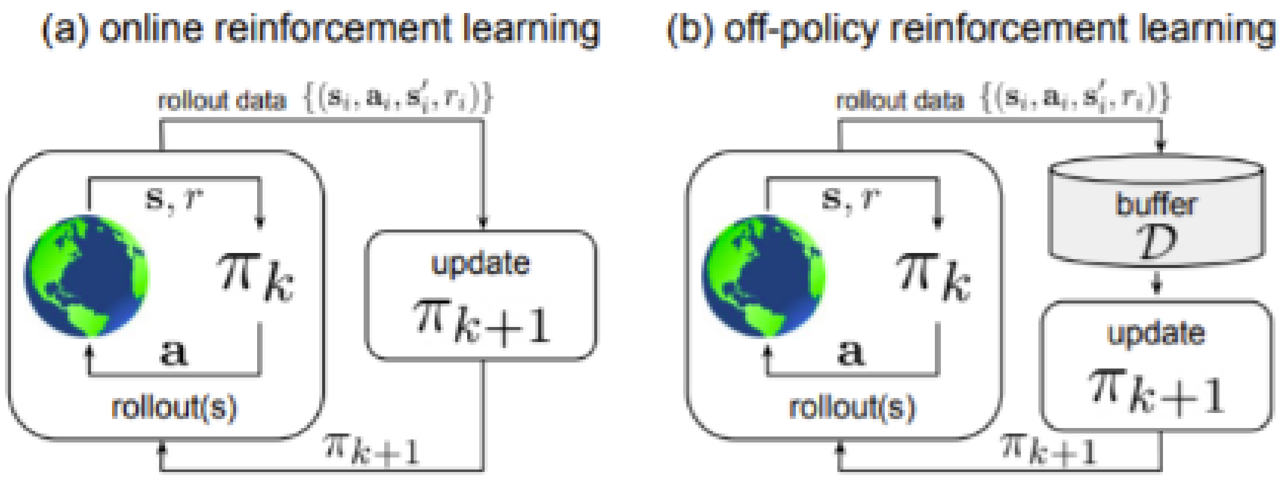

(a) online reinforcement learning
rollout data $\{(s_i, a_i, s'_i, r_i)\}$
s, r
π_k
a
rollout(s)
update
π_{k+1}
π_{k+1}
(b) off-policy reinforcement learning
rollout data $\{(s_i, a_i, s'_i, r_i)\}$
s, r
π_k
a
rollout(s)
buffer
$\mathcal{D}$
update
π_{k+1}
π_{k+1}

Figure 2.7: [Tmp] On-policy vs off-policy reinforcement learning. π_k is the target policy, π_{k+1} is the optimized policy, and $\pi^{'}$ is a different policy than the target

2.1.6 Deep Reinforcement Learning

Deep learning

NN and general approximators

First cool results from DQN (Atari)

Why DRL? Curse of dimensionality

Proximal Policy Optimization

PPO algorithm in details

Soft Actor-Critic

SAC algorithm in details

2.1.7 Multi-Agent Reinforcement Learning

- What is it?
- Why?
- When to use? cooperative, competitive, hybrid, independent agents, centralized, decentralized, super-agent....etc (Maybe add Infographic)
- Current methods and algorithms
- Current problems of non-stationarity, agent modeling, communication (topology), ...etc. (from surveys)
- MARL with predator-prey, similar games ...

2.2 Game Theory

TODO: explanation for Games properties and game theory ......

TODO: introduction to MG with some explanation for MG

Definition 11. Markov Game (MG) is ......

Definition 12. Multi-agent policy is ......

- Game specifications (Dynamic vs static, one-step vs multi-steps, zero-sum or not, mixed vs pure strategies, symmetric vs asymmetric, ...etc)
- Markov Games (Stochastic games) repeated games

- Extensive-Form game
- Normal-Form game (Maybe add infograph with information)
- Nash Equilibrium (Mention that it is hard to use in realistic environments, emphasize the motivation)
- Mention algorithms that currently being studied and giving cool results (Regret minimization, counterfactual regret minimization, counterfactual multi-agent policy gradients, ...etc

2.3 Games

- symmetric and asymmetric and examples
- Predator-Prey
- Why predator-prey
- applications and motivation in literature
- Pursuit-evasion games and other similar scenarios
- Current solutions for PEG and Predator-prey
- Classification of methods for Learning-based vs non-learning based solutions (linking between GT and RL)
- Drones with Predator-Prey

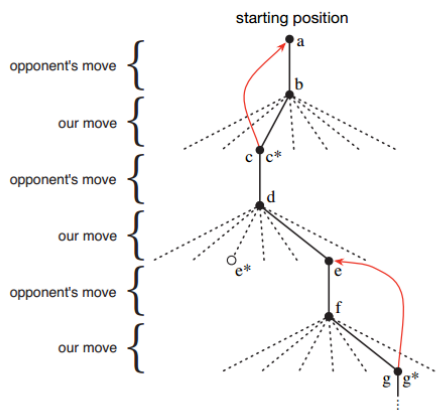


Figure 2.8: Example of two player decision making figure

2.4 Self-Play (SP)

Self-Play concept, origins and history

2.4.1 Fictitious-Play

Fictitious-Play

Counterfactual Regret Minimization

Game theory + Self-play (origins

2.4.2 Self-play with RL

SP with RL

General framework and formulation of SP (Add Infographic)

Where SP was used and how (analysis, training, opponent selection, evaluation, analysis)

2.4.3 Self-play and Multi-Agent Reinforcement Learning

Self-Play now (and how it is close to MARL, single-RL, continual learning, match making, curriculum learning, meta RL, Adversarial learning, behavioral learning, collective intelligence) self-play with similar games

self-play with predator-prey, hide and seek, ...

2.5 Summary

Connection of RQ and Literature

qualitive analysis for methods used to my problem

Chapter 3

Methodology and Implementation

The primary goal is to develop an algorithm based on Reinforcement Learning Self-Play <TODO: check the writing of this term> framework and present the results of applying such an algorithm on a predator-prey environment. Designing such an algorithm introduces challenges to analyze and interpret the results obtained. Therefore, this chapter establish the problem formulation observed in the thesis, describes the methods and the algorithms used, furthermore it explains the evaluation methods used for evaluation and analysis.

Section 3.1 introduces the problem statement of predator-prey environment according to the reinforcement learning settings. Section 3.2 presents the description of the implemented algorithm and illustrates the implementation details. Section 3.3 explains the evaluation methods and provides an the criteria and the intuition used for the evaluation.

3.1 Problem Statement

In order to verify our self-play algorithms and the evaluation methods, we have chosen predator-prey environment. This section formulates the problem of predator-prey in terms of a reinforcement learning environment. We presents the environment's and agents' characteristics, the possible actions and states of the embodied agents.

The predator-prey environment consists of two players/agents, one is the predator and the other is the prey. However, generally there are more than 2 players such that it is N vs M. In this thesis, the environment is a homogeneous multi-agent system, which means that all the agents have the same characteristics. Moreover, we assume that there is no obstacles situated in the environment. In predator-prey, either the predator wins or the prey, they cannot tie and they cannot both win, this property returns to the nature of the predator-prey problem.

We have decided to simulate the predator and prey as embodied agents with robotics realization instead of a particle environment [74]–[76]. Moreover, the main reason was that we wanted in the future to continue working on this research direction with an intention to transfer the learnt policy from the simulator to the reality (Sim2Real). Therefore, we have implemented the embodied agents in two ways: Section 3.1.4 presents the necessary information for an environment with agents represented as 2D differential drive wheeled robots. Section 3.1.5 describes the environment with agents simulated as quadcopter drones in a 3D space.

3.1.1 Terminal State

This environment is an episodic RL task that has a limit number of steps equals 1000 steps, after that the environment has two main natural terminal states:

1. The predator could catch the prey and the predator is the winner.
2. The predator could not catch the prey in the specified number of steps, and the prey is the winner.

3.1.2 Reward

In predator-prey, the reward is designed to be asymmetric as it is described by equation (3.1). This conforms with the zero-sum property of the predator-prey environment.

$$R_t(predator) = -R_t(prey)\,\forall\, t \in 1, ..., 1000 \tag{3.1}$$

Such that $R_t(.)$ is the immediate step reward for the agent and t is the timestep.

The reward function is essentially sparse that relates to the winner of the episode, moreover, it is augmented with a fixed constant survival reward each step. We selected the sparse reward for two reasons: First, the sparse reward that relates to the winner of the episode is the natural reward for the predator-prey. Second, this sparse reward function enables us to extend the work to different problems of zero-sum games. The survival reward is used to encourage the agent to its task, the prey to not to be caught and the predator to try to catch the prey as fast as possible.

In the terminal states, the agents receive specific rewards that demonstrate which agent has won the episode whether it is the predator or the prey.

The following equations (3.2), (3.3) describes the predator's agent reward function and the prey's agent reward function respectively. Hence, the prey agent's reward function is exactly the same but with an opposite sign according to equation (3.1).

$$\mathrm{R}_t(predator) = \begin{cases} -1, & \text{predator did not catch the prey at t (survival reward)} \\ -10, & \text{if t=T \& predator did not catch the prey (terminal rewa} \\ +10, & \text{if predator caught the prey} \end{cases}$$

$$\mathrm{R}_t(prey) = \begin{cases} +1, & \text{predator did not catch the prey at t (survival reward)} \\ +10, & \text{if t=T \& predator did not catch the prey (terminal reward)} \\ -10, & \text{if predator caught the prey at t (terminal reward)} \end{cases}$$

The above mentioned equations (3.2), (3.3) represent a version of the possible reward functions, however, this reward function can be designed use more dense reward signal. An example is to add the relative distance between the predator and the prey with the survival reward. This will provide a more informative reward signal for the agents. However, we tried not to design the reward function with such modifications to show that our training algorithm can get results with the simple version of the reward function that can be used with other environments.

3.1.3 Agents as particles

Action Space

Observation Space

Reward

3.1.4 Agents as differential drive robots

We have simulated the agents as 2D differential drive wheeled robots. We have used evorobotpy2 predator-prey environment that is written using Cython [77]. Therefore, it enabled fast training and experiments with our algorithms that speeded up the process of debugging and development. It uses MarXbot robots [78] to simulate the agents.

The simulated marXbot has .... sensors, motors, ...

* Sensors

* Actuators

Furthermore, the agents are situated in a constrained environment

(TODO: dimension) and the motion of the robots is only in the 2D space over the plane of the environment. The agents starts at ....(TODO)

Based on the previous described integrated sensors and actuators in the robot, we have used the following action space and observation space for the RL problem of this environment.

Action Space

Observation Space

3.1.5 Agents as quadcopter drones

Furthermore, we have simulated the predator and prey agents as quadcopter. Quadcopter is a drone with 4 rotors that can fly in the 3D space, hence we have chosen quadcopters to extend our results within the 3D space environment and explore the possible challenges with such environments. Moreover, we have selected working on simulating the environment with quadcopter agents due to the availability of the hardware at the university during working on the thesis aiming in the future to perform sim2real experiments with the learnt policies.

We have chosen gym-pybullet-drones [79] environment as our base environment. This environment is written as a gym-environment [80] which can be used easily and compatible with multiple of RL algorithms libraries, also it is based on the opensource PyBullet [81]; it is a physics engine with python wrappers that enables to simulate environments with the modeled physics.

gym-pybullet-drones [79] originally supports simulation of multiple drones by default as Bitcraze's Crazyflie 2.x. Crazyflie is a nano-quadrotor that is used extensively in the indoor research tasks. Furthermore, we have created a predator-prey environment with one agent as a predator and the other agent as a prey and we have bounded the environment with a closed box <TODO: dimensions> to constrain the drone's motion space.

[redacted]

Action Space

[redacted]

Observation Space

We have selected the kinematics information for each drone to be the main observations of the drone. The kinematics information consists of: (x, y, z) the Cartesian position, $(\mathit{Roll},\ \mathit{Pitch},\ \mathit{Yaw})$ the drone's orientation angles, $(\dot{x}, \dot{y}, \dot{z})$ the linear velocities, and (w_x, w_y, w_z) the angular velocities, that is in total 12 observation dimension for the single drone's kinematics information. Furthermore, all the values of the observations are normalized to be in the range of

$[-1, 1]$.

We have selected this set of observations <TODO: the reason>

Each agent's observation consists of the all the observed kinematics information for all the drones in the environment. For example in our case, we have two agents (predator and prey), the predator will observe the kinematics information of the predator itself and the kinematics information of the other agent, the prey (i.e. Predator's observation $\in \mathbb{R}^{24}$).

Moreover, we are using this set of observations in the training and execution, notice that it is Centralized during Training and Centralized during Execution, not Centralised during Training and Decentralized during Execution (CTDE) due to the architecture of the training network. Hence, the current work is limited to work with small number of agents in the environments, as if we increased the number of agents, the dimension of the observation space will increase as well and will cause the problem of curse of dimensionality. Further work can be done to use a centralized critic that is only used during the training and not used during the execution, and many other methods that can be used.

Reward and termination modifications

The drones' environment has introduced a problem due to 3D space motion. In the 3D space, the drones can flip and crash into the ground or they can get crashed into the boarders of the environment. These scenarios can happen with the wrong set of actions during the training. However, these scenarios were not possible in the 2D environment, as the motion is constrained in the planar 2D space. These scenarios made us introduce new termination and terminal conditions, and modifications to the reward function.

[redacted]

3.2 Self-Play Algorithm

Self-play (SP) is a training schema that is used in multi-agent environments. SP uses versions/copies of the agents as opponents to create an auto training curricular for the agent to improve. SP is considered as a subset of multi-agent reinforcement learning (MARL) but in SP not all the agents in the environment are training but only a subset and the rest are old copies and untrainable [82].

Self-play is used when the task to solve involves competitiveness such that the agent must compete an opponent to achieve its task, however, this does not prevent having cooperation between some agents. Hence, SP is a powerful technique to emerge competitive behaviors.

Self-play training algorithm can be separated into three main components, these component are represented in form of questions and different combination of answers for these questions can create a different SP algorithm:

- Which subset of agents is selected to be trainable? Self-play assumes that not all the agents are trainable, for example in the case of 1 vs 1, we have a single agent that is training against an untrainable agent. Therefore, we need to identify which agents to be trained, are they the last evolved agents or something else.

- How do we choose which opponents to train the agents against? This question is one of the key differences between different SP algorithms. As the opponents should be chosen such that the agents will progressively improving in terms of performance.

- How do we evaluate the agents? When to identify that the agents are progressively improving and the auto curriculum is useful for the agents such they are improving or not.

In this thesis, we always used the last evolved agent as the selected agent to be trained. Furthermore, the agents are training for a limited timesteps then they are switching the training that we call it an alternating training scheme and this iteration or round of training is repeated for a certain number of times. In 1 vs 1 scenario, the first agent is training against copies of the second agent as opponents, then training the second agent against copies of the first agent as opponents, then this round is being repeated multiple of times with different opponents.

We have implemented a simple naive SP algorithm described in 3.2.2, then we have implemented the work of [24] which we called it the standard algorithm as it has been observed that it was the base algorithm for multiple self-play works in the literature due to the simplicity of the idea, and we implemented the algorithm of δ-limit uniform described in [82]. Furthermore, we have implemented a variant of the standard SP algorithm based on using multiple populations during the training that adds a variety of versions of opponents to be selected from during the training.

3.2.1 General formulation

In the literature, self-play lacks a general definition that makes it difficult to trace back all the work on self-play. However, a recent work [82] had pointed out to this research gap and introduced a generalized definition of SP algorithm that we are going to use a similar way to describe our implemented variant.

Moreover, we are going to formalize SP using the same terminology in [64] such that: SP involves a hierarchical optimization problems consists of two levels:

1. The inner optimization problem that involves the reinforcement learning (RL) algorithm that is used to train the agents versus the opponents. Such that the agents are trainable while the opponents are untrainable, in this thesis, we focused on using Proximal Policy Optimization (PPO) [72] and Soft Actor Critic (SAC) [73] as the RL algorithms to solve the inner optimization problem. We focused on these two algorithms due to its importance in the literature and the fact that they are the current state-of-the-art for single-agent RL with continuous spaces for on-policy and off-policy RL algorithms respectively. However, we think that it is possible to use multi-agent reinforcement learning (MARL) algorithm in the case of having multiple agents more than 1 vs 1 case, such that the trainable agents are more than one agent, thus, we can use MARL algorithm to emerge a cooperative and competitive behavior inside the team and care about the non-stationary problem that is approached by MARL algorithms.

2. The outer optimization problem that involves selecting the hyper parameters for the inner optimization problem's solver, choosing which agents to

train, and choosing which agents as opponents (untrainable). Jaderberg et al. (2019) [64] showed an interesting rather complex method of using Population-Based Training (PBT) [83] to train multiple populations with different training parameters then choose the best outcome and mutate them to generate a new powerful offspring of parameters to be used for further training. This method has proven how much it is powerful [64], however, we rather focused on sampling-based methods for the opponents due to its simplicity and the required computational complexity. Furthermore, we created a new algorithm that combine the sampling-based and the population-based methods, as we call it the population-archive self-play method in 3.2.7.

In the followeing, we define a generalized self-play training scheme:

Math with sets and so on:

Pseudo-code:

Furthermore, sampling the opponents can be done either in with each reset to the agent's environment or with the beginning of each training's roll-out. In chapter 4, we will show the experimental results based on testing both configurations.

TODO: explain more about the alternative training, taking turns and rounds for training, training for certain amount of time using RL algorithm then change turns and so on.

Infographics:

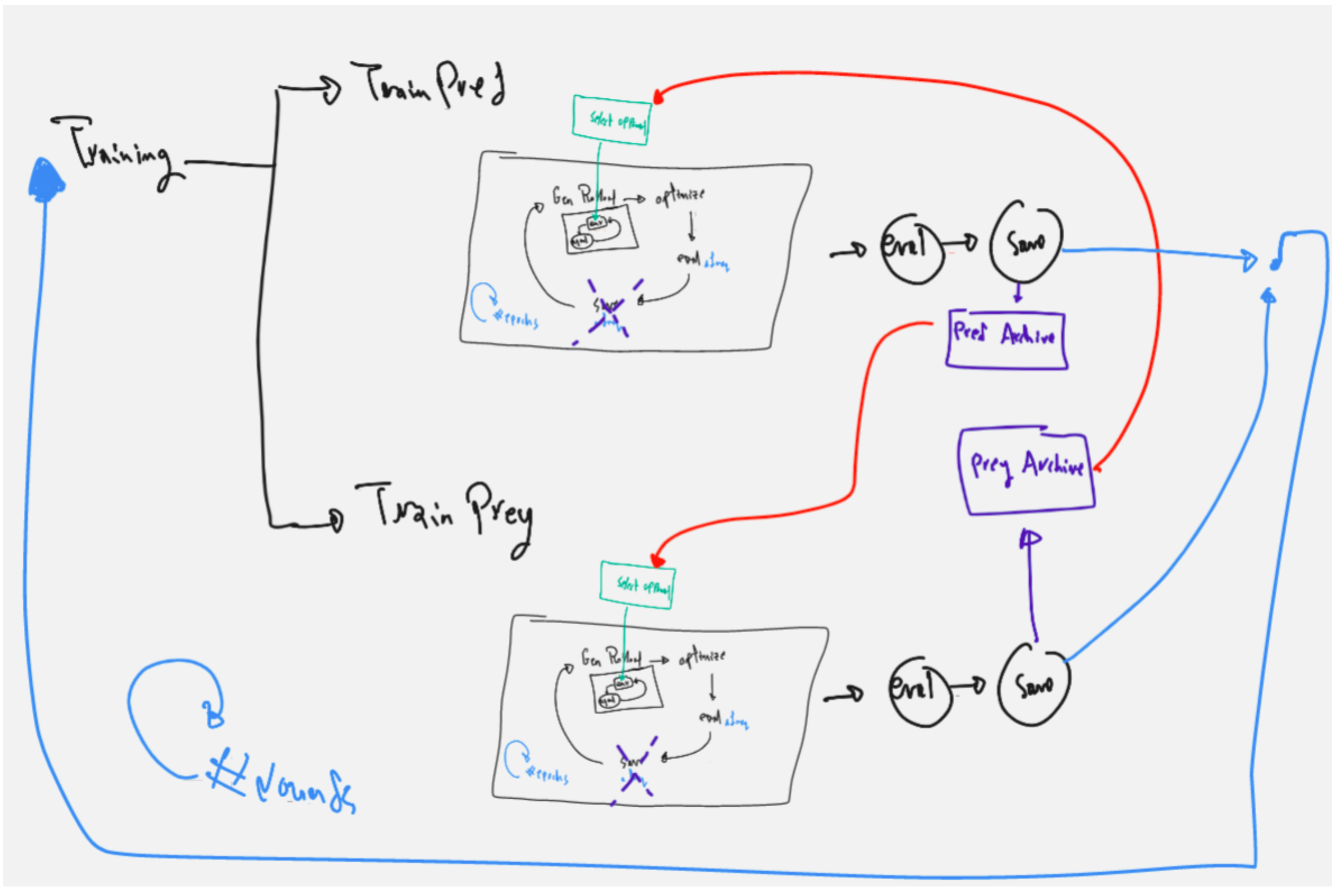


Figure 3.1: Self-play algorithm general diagram (part1)

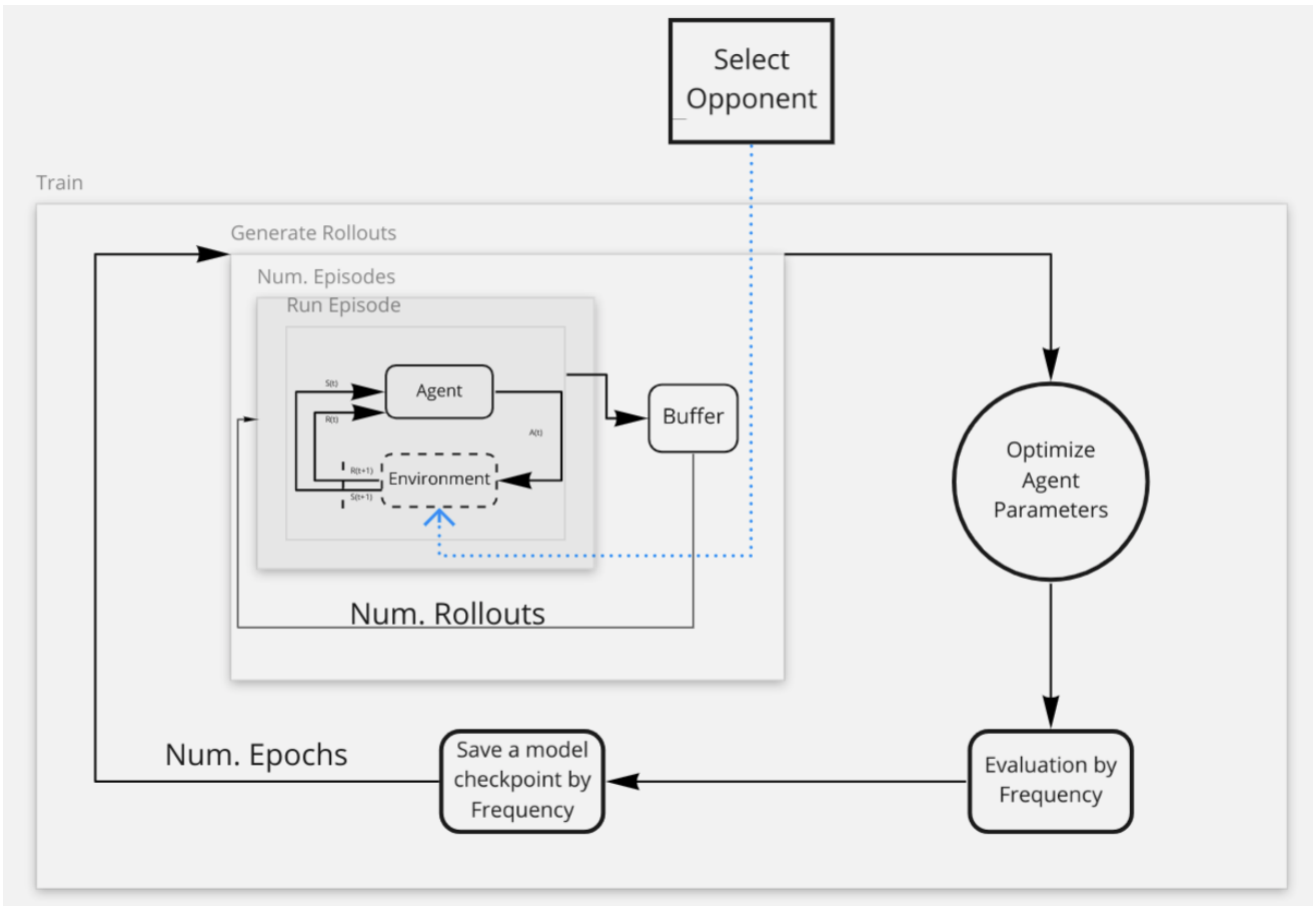


Figure 3.2: Self-play algorithm general diagram (part2) (Training block)

In the following subsections, we are going to explain the key differences between the different self-play algorithms that we have implemented, the fundamental key difference is in the opponent selection function that have a different sampling technique to choose the opponents that the agent is training against.

3.2.2 Naive algorithm

The naive algorithm is a basic fundamental sampling-based algorithm such that it selects the latest evolved version of the opponent as the current opponent for the agent. Using the latest version of the opponent to train the agent against on is the naive idea that can be used assuming that the latest version is the strongest version among the past (the history of versions), however, this does not always hold due to <TODO: It is RL :) write it, RL has instability in training>, therefore, there is no guarantee that the agent is always is competing the the strongest opponent.

TODO: explain the figure somehow (why the agents in front takes more frequency in general, why the agents in the middle takes more frequency in general)

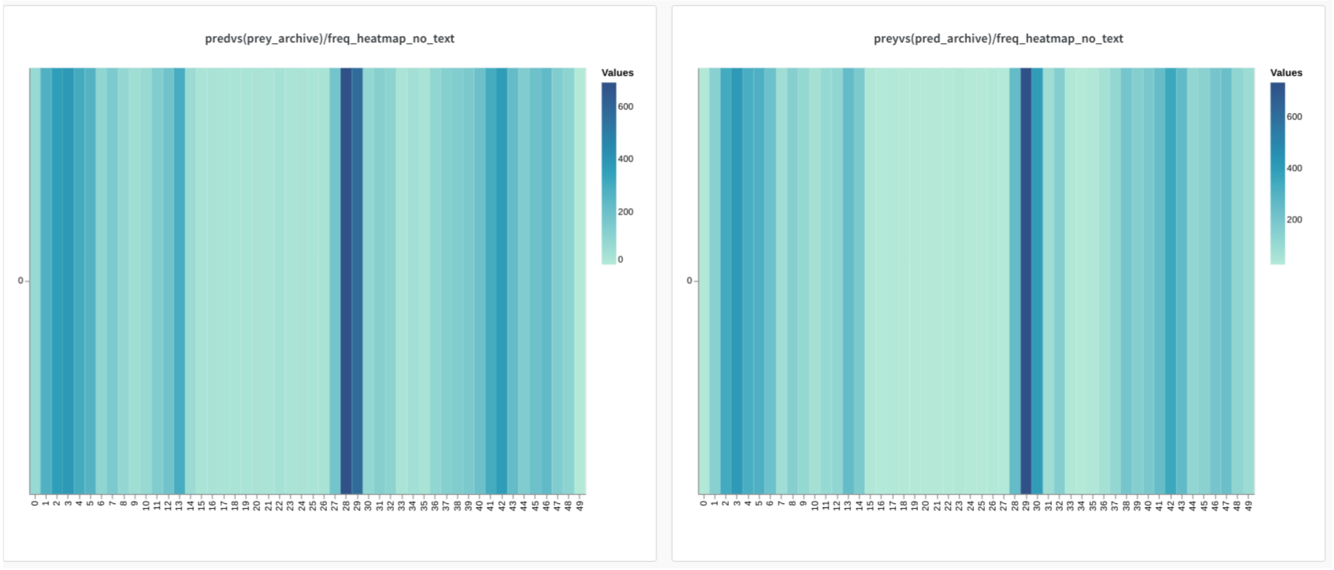


Figure 3.3: Frequency table for the used versions as opponents

3.2.3 Standard algorithm

This algorithm has been introduced in [24] in comparison with the naive algorithm such that it shown empirically better results than using the latest versions as opponents. Instead, they are sampling the opponents randomly from an archive that includes all the previous versions of the opponents. Moreover, the random sampling is based on a uniform distribution which gives an equal sampling probability in the same training round. Bansal et al. (2017) has shown that the choosing randomly is better than the choosing the last evolved opponent as competing random opponents increases the variability and the chance of competing good opponents and reduce the chance of overfitting against a specific behavior.

TODO: explain the figure somehow

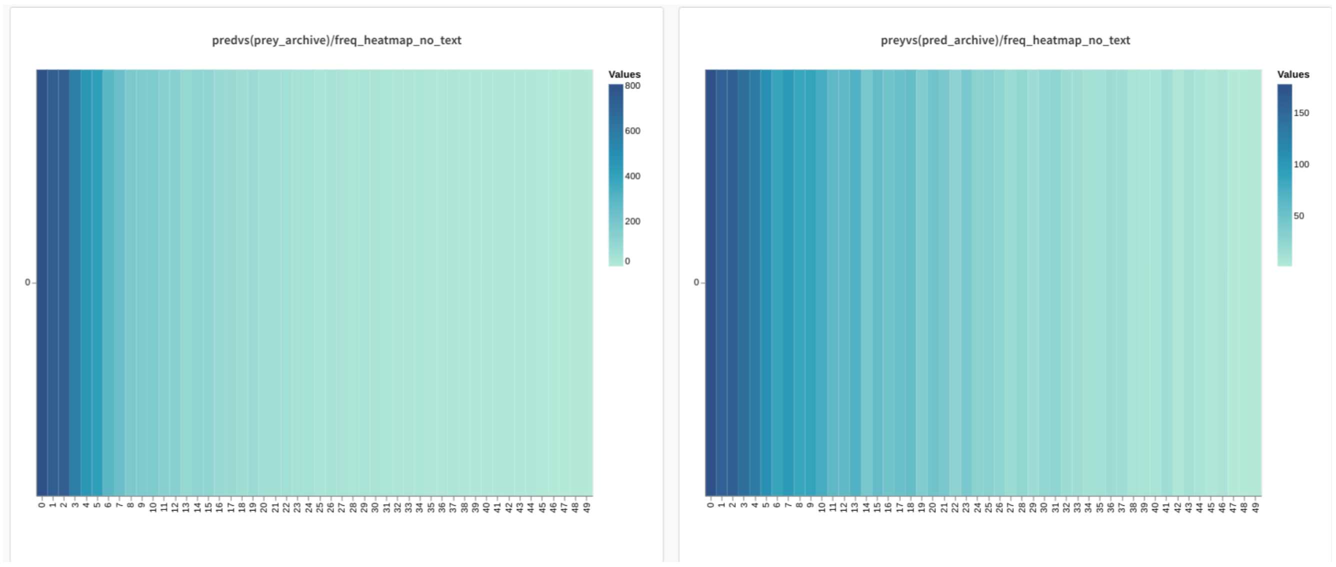


Figure 3.4: Frequency table for the used versions as opponents

3.2.4 Delta-last algorithm

Hernandez et al. (2020) has demonstrated a new algorithm based on sampling not using a uniform random distribution among all the archive of history/versions but using a subset of this history [82]. For example using the latest δ versions of the agent to sample from. If the archive has 10 versions of agents, and $\delta = 5$, thus, we will use only 5 versions to use it to sample from it uniformly.

TODO: explain the figure somehow

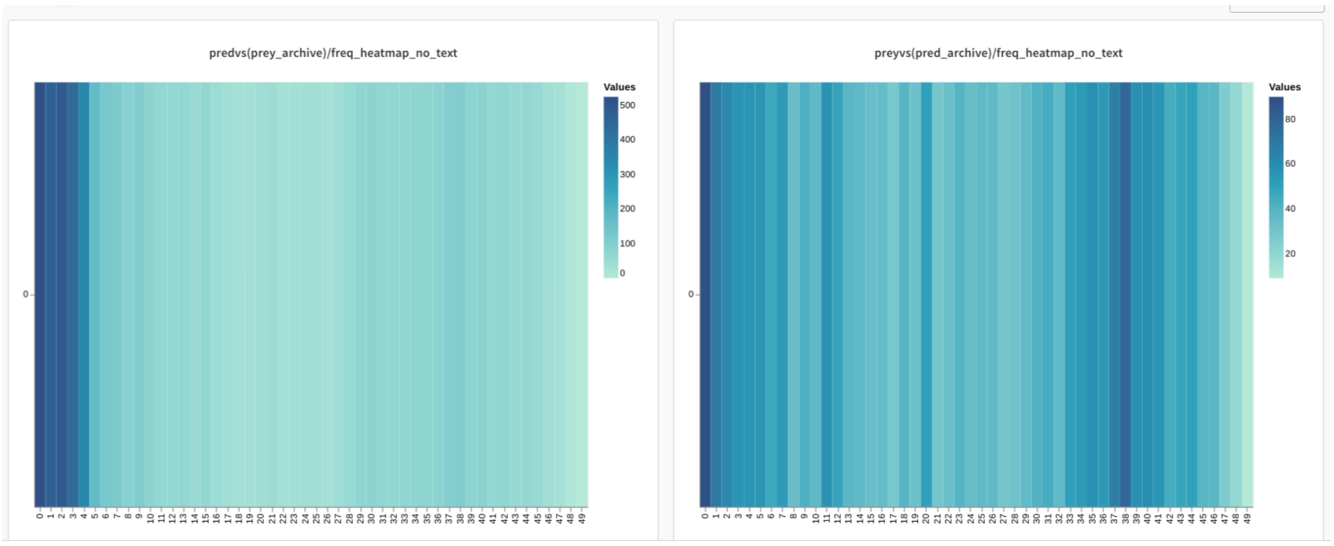


Figure 3.5: Frequency table for the used versions as opponents, $\delta = 10$

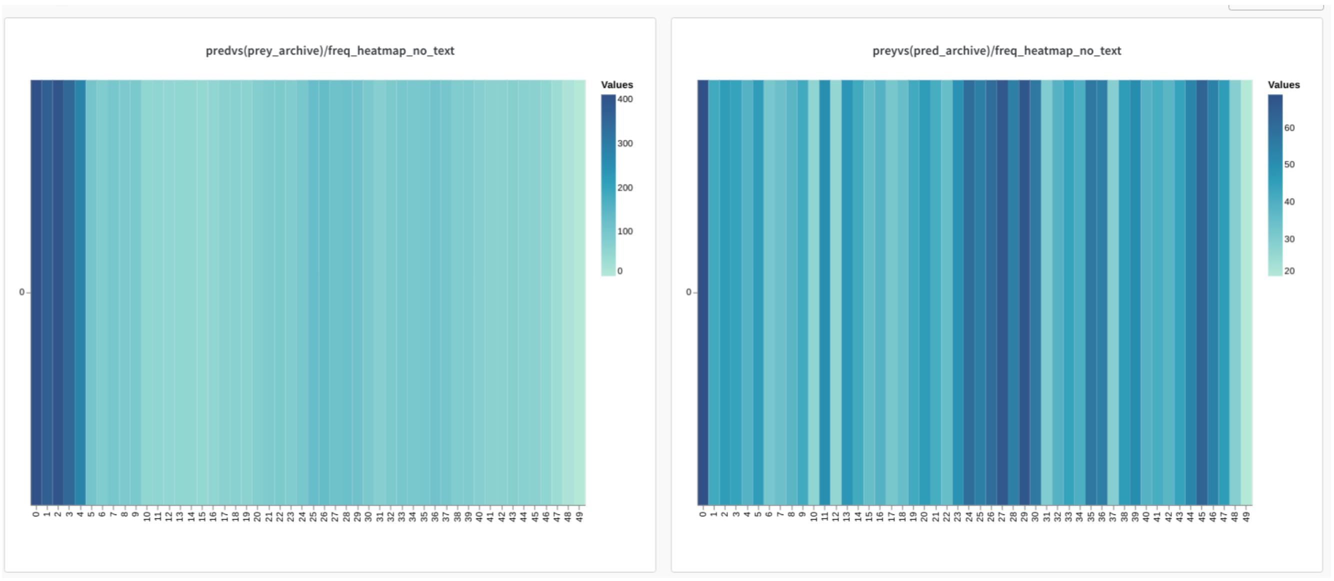


Figure 3.6: Frequency table for the used versions as opponents, $\delta = 5$

3.2.5 Cyclic algorithm

This variation of the standard algorithm is fairly simple. The agent is training against the opponents such that the opponents are stored in a cyclic buffer and the opponents are selected for training by iterating on this buffer. For example, if the current training round, the opponent archive has 10 versions and the current index is on the first opponent in the archive, the agent is training against the first opponent then for the second sampling time, the agent is training against the second opponent, the third, ...etc.

TODO: explain the figure somehow

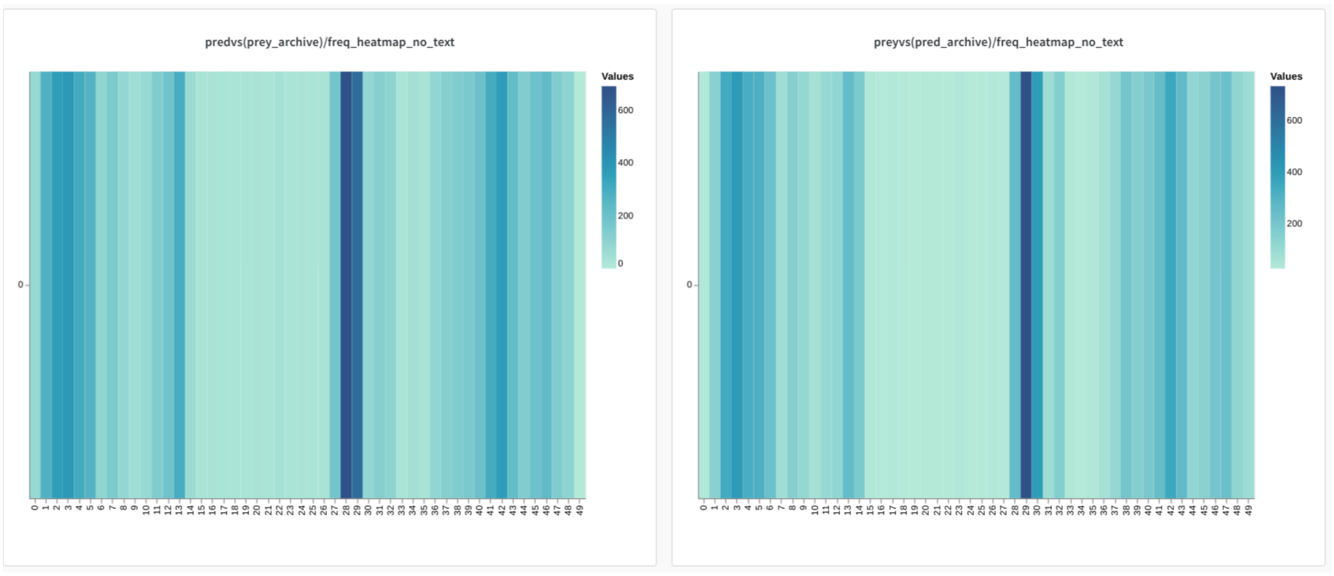


Figure 3.7: Frequency table for the used versions as opponents

3.2.6 Gauss-prob algorithm

TODO: explain the sampling function

3.2.7 Population-archive algorithm

In this method, we have added multiple populations self-play trained agents that they share the same archive, the populations can either use the same algorithm for sampling/selecting the opponent or they use different techniques. The intuition behind using multiple of populations and sharing the archive among them is that this adds more diversity and variation in the possible sampled opponents which contributes in the evolution and the emergence of the behaviors for the agents. Each population has a different initial seed and it is possible to have different hyper parameters as well. Although this algorithms is much slower than other algorithms due to running more than one population, it showed better results explained in chapter 4.

Furthermore, we are showing the modified pseudo-code as following:

TODO: pseudo-code

TODO: explain the figure somehow

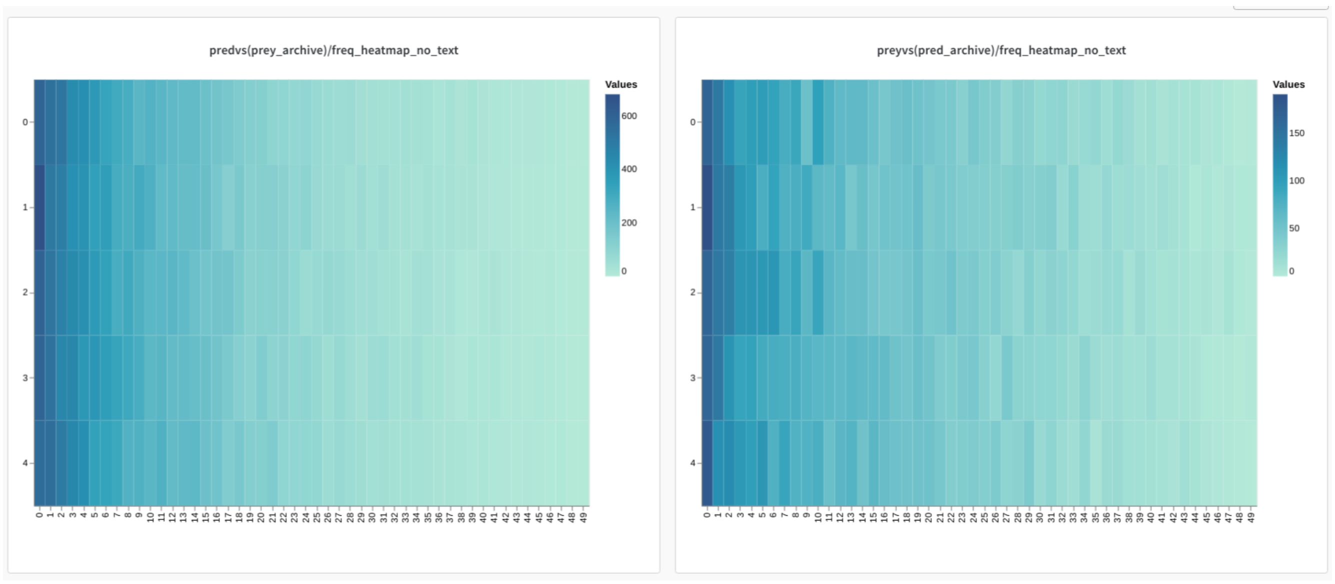


Figure 3.8: Frequency table for the used versions as opponents, populations=5

3.2.8 Ensemble-based algorithm

This method is using the concept of using ensemble model during the training and during the evaluation, during the training we are using the ensemble model for the opponents and during the evaluation we are using the ensemble model as the agent

3.2.9 Competitive-based algorithm

Calculate the master tournament in online manner and then give more probability for the agents that is hard to beat (episode length is small)

3.3 Evaluation Methods

TODO: partially explain here some of the figures

In this section, we are explaining the evaluation metrics that we have used. These evaluation metrics are used to explain how the agents are evolved during the training or after the training. Furthermore, we are interested to understand which method is better in evolving the agent than the others.

3.3.1 Fundamental evaluation metrics

Most intuitive metrics are win rate, mean reward, and mean episode length that closely describe the agent's performance during the training, moreover, these metrics can be used as units for more evaluation. In order to understand the relative performance between each version in the history of the agent versus the other agent, we are using an evaluation matrix that we call it master tournament analysis.

- Win rate is the ratio of winning of the agent to the total matches that was played, higher it is, better the performance of the agent against the opponent.

- Mean reward is the average of the rewards that the agent could get against the opponents. However, the boundaries is depending on the design of the reward matrix and might be different among different environments, therefore, we prefer to use the mean episode length as all the environments are limited by the same maximum number of steps.

- Mean episode length is the average of episode lengths that the agent experienced. Higher value is bad for one agent (the predator) and good for the second agent (the prey)

TODO: write about elo-score and how it is used

3.3.2 Master tournament

Master tournament is a NxN matrix such that N is the number of rounds for the training, each agent is being tested against each other opponent for multiple of episodes. The ideal master tournament matrix is a lower triangular matrix for one agent and upper triangular matrix for the other agent. The later means that the agent is progressively improving such that $(i+1)^{th}$ agent's version is better than {0, 1, ... i^{th}} opponents' versions and vice versa for the other agent.

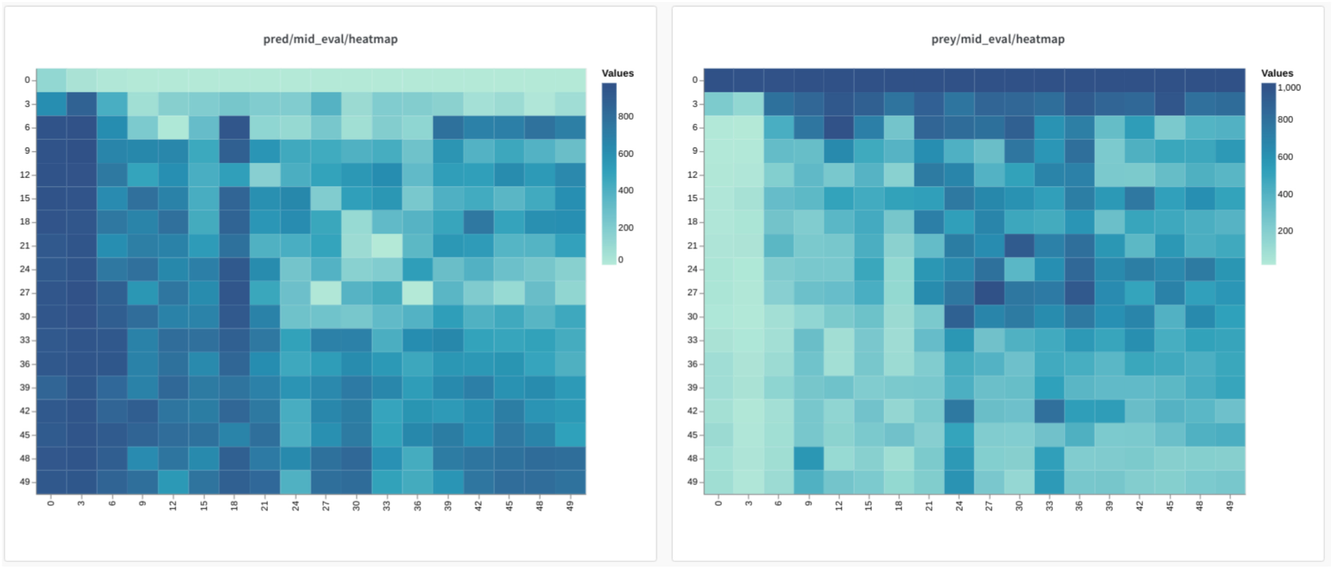


Figure 3.9: Ideal master tournament matrix

The ideal matrix is hard achieve in practice due to the instability of RL training and the ambiguity of the choosing the appropriate opponent to train against to generate a stronger agent. Therefore, many variations are being implemented in order to realize the ideal auto-curricular to have a progressive improving in the performance.

Computation of the master tournament can be done in an online manner after each checkpoint in which the agent's version is being saved, or in an offline manner at the end of the training. Unfortunately, the online computation is time consuming during the training, thus, it is not advised to use unless your are using this evaluation's information for choosing the appropriate opponent as it is described in 3.2.9

3.3.3 Crosstest

Crosstes is an evaluation method that we created to evaluate different algorithms and find out the best algorithm that generated the strongest agents.

[82] has used a similar evaluation metrics, they used an evaluation based on the computation of nash equilibrium which is not easy to compute accurately in most of the real world scenarios which make it hard to use it in our problem. [64] has used an evaluation based on elo-score and comparing it with the scores for the agents generated by the other method.

We have created a method that is extracting the best agents generated by each algorithm then making a match between these different agents as the following pseudo-code

[redacted]

And the computation of the score is based on the following formula:

TODO: score equation

TODO: add annotated equations latex

[redacted]

TODO: explanation of the formula That when it is greater than 0, then the first method is better ....

3.4 Implementation

Discuss about the implementation details (SB3, PyTorch, ...etc)

3.5 Conclusion

Chapter 4

Results and Evaluation

TODO: explain here the notation of the graphs

In this chapter, we are going to show the results from our experiments on using the methods and the evaluation metrics that was discussed in chapter 3.

Intro and outline

- Master Tournament graphs
- Mean reward, length, win-rate graphs
- trajectory graphs for the robots
- Crosstest results (Compare between the methods)
- Discuss the results of drone and differential drive robot
- Discuss different trials for the training
- Mention the hyperparameters
- Mention different effects of the reward and different RL algorithm

We want multiple experiments each experiment with different seed for each method

Repeated for each environment (3 envs)

- 10 experiments/seed for random

- 10 experiments/seed for latest

- 10 experiments/seed for cyclic

- 10 experiments/seed for reverse-cyclic (extra)

- 10 experiments/seed for delta 3

- 10 experiments/seed for delta 5

- 10 experiments/seed for delta 10

- 10 experiments/seed for population 2

- 10 experiments/seed for population 5

- 10 experiments/seed for population 10

- 10 experiments/seed for population 15

- 10 experiments/seed for ensemble-single

- 10 experiments/seed for ensemble-pop5

- 10 experiments/seed for competitive

Each 10 experiments/seeds:

- Graph for mean rewards, mean episode length, win-rate (mean and variance)

- Graph for

4.1 EvorobotPy2 Environment

[redacted]

4.2 PettingZoo-MPE Environment

[redacted]

4.3 Pybullet Drones Environment

TODO: select which experiments to do exactly:

- Experiments with different HP on one of RL algorithms and one of the environments

- Experiments with different RL algorithms on one of the methods and one of the environments

Chapter 5

Discussion and Future Works

Intro and outline

- Mention the difficulties in the above mentioned methods of evaluation and training algorithms
- Mention the limiations
- Mention the future possible works in terms of simulations (isaac gym), environments, types of games, methods, algorithms ...etc

Chapter 6

Conclusion

- Conclude everything (introduction, research questions, literature, methods, implementations, results, and findings)
- Mention the future work and possibility to extend this work
- Mention the current problems, limitations, bottlenecks
- Mention the vision in this field

Bibliography cited

[1] M. Wooldridge, *An introduction to multiagent systems*. John wiley & sons, 2009.

[2] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *Ieee Access*, vol. 6, pp. 28 573–28 593, 2018.

[3] T. Arai, E. Pagello, L. E. Parker, *et al.*, "Advances in multi-robot systems," *IEEE Transactions on robotics and automation*, vol. 18, no. 5, pp. 655–661, 2002.

[4] A. Gautam and S. Mohan, "A review of research in multi-robot systems," in *2012 IEEE 7th international conference on industrial and information systems (ICIIS)*, IEEE, 2012, pp. 1–5.

[5] D. Ha and Y. Tang, "Collective intelligence for deep learning: A survey of recent developments," *arXiv preprint arXiv:2111.14377*, 2021.

[6] A. Gautam and S. Mohan, "A review of research in multi-robot systems," in *2012 IEEE 7th international conference on industrial and information systems (ICIIS)*, IEEE, 2012, pp. 1–5.

[7] Z. Ren and C. J. Anumba, "Multi-agent systems in construction–state of the art and prospects," *Automation in Construction*, vol. 13, no. 3, pp. 421–434, 2004.

[8] S. D. McArthur, E. M. Davidson, V. M. Catterson, *et al.*, "Multi-agent systems for power engineering applications—part ii: Technologies, standards, and tools for building multi-agent systems," *IEEE Transactions on Power Systems*, vol. 22, no. 4, pp. 1753–1759, 2007.

[9] B. Roche, J.-F. Guégan, and F. Bousquet, "Multi-agent systems in epidemiology: A first step for computational biology in the study of vector-borne disease transmission," *BMC bioinformatics*, vol. 9, no. 1, pp. 1–9, 2008.

[10] L.-H. Ren, Y.-S. Ding, Y.-Z. Shen, and X.-F. Zhang, "Multi-agent-based bio-network for systems biology: Protein–protein interaction network as an example," *Amino acids*, vol. 35, no. 3, pp. 565–572, 2008.

[11] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[12] O. Vinyals, I. Babuschkin, W. M. Czarnecki, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[13] P. Balaji, G. Sachdeva, D. Srinivasan, and C.-K. Tham, "Multi-agent system based urban traffic management," in *2007 IEEE Congress on Evolutionary Computation*, IEEE, 2007, pp. 1740–1747.

[14] I. A. Videos, Mar. 2021. [Online]. Available: https://www.youtube.com/watch?v=cn86KkPNYjk.

[15] 2015. [Online]. Available: https://sdg.yandex.com/deliveryrobot/.

[16] E. Tuci, M. H. Alkilabi, and O. Akanyeti, “Cooperative object transport in multi-robot systems: A review of the state-of-the-art,” *Frontiers in Robotics and AI*, vol. 5, p. 59, 2018.

[17] J. Alonso-Mora, S. Baker, and D. Rus, “Multi-robot formation control and object transport in dynamic environments via constrained optimization,” *The International Journal of Robotics Research*, vol. 36, no. 9, pp. 1000–1021, 2017.

[18] T. Laengle and H. Wörn, “Human–robot cooperation using multi-agent-systems,” *Journal of intelligent and Robotic Systems*, vol. 32, no. 2, pp. 143–160, 2001.

[19] 2017. [Online]. Available: https://socialcars.github.io/b-decentralized-cooperative/.

[20] Associated and M. Lenthang, *How amazon’s fleet of warehouse robots are causing injury and stress to human staff*, Dec. 2019. [Online]. Available: https://www.dailymail.co.uk/news/article-7837379/How-Amazons-fleet-warehouse-robots-causing-injury-stress-human-staff.html.

[21] W. Wang and L. Chen, “A predator-prey system with stage-structure for predator,” *Computers & Mathematics with Applications*, vol. 33, no. 8, pp. 83–91, 1997.

[22] Y. Seitbekova and T. Bakibayev, “Predator-prey interaction multi-agent modelling,” in *2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT)*, IEEE, 2018, pp. 1–5.

[23] A. J. Lotka, *Elements of physical biology*. Williams & Wilkins, 1925.

[24] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, “Emergent complexity via multi-agent competition,” *arXiv preprint arXiv:1710.03748*, 2017.

[25] 2022. [Online]. Available: https://www.planetstillalive.com/gallery-categories/zambia-nsefu-lions-hunting/.

[26] D. E. Asher, E. Zaroukian, and S. L. Barton, “Adapting the predator-prey game theoretic environment to army tactical edge scenarios with computational multiagent systems,” *arXiv preprint arXiv:1807.05806*, 2018.

[27] J. Liu, S. Liu, H. Wu, and Y. Zhang, “A pursuit-evasion algorithm based on hierarchical reinforcement learning,” in *2009 International Conference on Measuring Technology and Mechatronics Automation*, IEEE, vol. 2, 2009, pp. 482–486.

[28] Y. Ishiwaka, T. Sato, and Y. Kakazu, “An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning,” *Robotics and Autonomous Systems*, vol. 43, no. 4, pp. 245–256, 2003.

[29] V. P. Crawford and N. Iriberri, “Fatal attraction: Salience, naivete, and sophistication in experimental" hide-and-seek" games,” *American Economic Review*, vol. 97, no. 5, pp. 1731–1750, 2007.

[30] S. Gal and J. Casas, “Succession of hide–seek and pursuit–evasion at heterogeneous locations,” *Journal of the royal society interface*, vol. 11, no. 94, p. 20 140 062, 2014.

[31] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry, “Probabilistic pursuit-evasion games: Theory, implementation, and experimental evaluation,” *IEEE transactions on robotics and automation*, vol. 18, no. 5, pp. 662–669, 2002.

[32] A. Von Moll, D. W. Casbeer, E. Garcia, and D. Milutinović, "Pursuit-evasion of an evader by multiple pursuers," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2018, pp. 133–142.

[33] M. V. Ramana and M. Kothari, "Pursuit-evasion games of high speed evader," *Journal of intelligent & robotic systems*, vol. 85, no. 2, pp. 293–306, 2017.

[34] A. Alexopoulos, T. Schmidt, and E. Badreddin, "A pursuit-evasion game between unmanned aerial vehicles," in *2014 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, IEEE, vol. 2, 2014, pp. 74–81.

[35] Z. Deng and Z. Kong, "Multi-agent cooperative pursuit-defense strategy against one single attacker," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5772–5778, 2020.

[36] Z. E. Fuchs, P. P. Khargonekar, and J. Evers, "Cooperative defense within a single-pursuer, two-evader pursuit evasion differential game," in *49th IEEE conference on decision and control (CDC)*, IEEE, 2010, pp. 3091–3097.

[37] M. D. Davis, *Game theory: a nontechnical introduction.* Courier Corporation, 2012.

[38] A. R. Washburn *et al.*, "Two-person zero-sum games," 2014.

[39] D. B. Gillies, "Solutions to general non-zero-sum games," *Contributions to the Theory of Games*, vol. 4, no. 40, pp. 47–85, 1959.

[40] Y. Wang, L. Dong, and C. Sun, "Cooperative control for multi-player pursuit-evasion games with reinforcement learning," *Neurocomputing*, vol. 412, pp. 101–114, 2020.

[41] M. Lanctot, V. Zambaldi, A. Gruslys, *et al.*, "A unified game-theoretic approach to multiagent reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[42] J. Parker-Holder, M. Jiang, M. Dennis, *et al.*, "Evolving curricula with regret-based environment design," *arXiv preprint arXiv:2203.01302*, 2022.

[43] T. H. Chung, G. A. Hollinger, and V. Isler, "Search and pursuit-evasion in mobile robotics," *Autonomous robots*, vol. 31, no. 4, pp. 299–316, 2011.

[44] A. Brunetti, D. Buongiorno, G. F. Trotta, and V. Bevilacqua, "Computer vision and deep learning techniques for pedestrian detection and tracking: A survey," *Neurocomputing*, vol. 300, pp. 17–33, 2018.

[45] E. Bastianelli, G. Castellucci, D. Croce, R. Basili, and D. Nardi, "Effective and robust natural language understanding for human-robot interaction.," in *ECAI*, 2014, pp. 57–62.

[46] P. Shah, M. Fiser, A. Faust, J. C. Kew, and D. Hakkani-Tur, "Follownet: Robot navigation by following natural language directions with deep reinforcement learning," *arXiv preprint arXiv:1805.06150*, 2018.

[47] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 1205–1212.

[48] I. Abraham and T. D. Murphey, “Active learning of dynamics for data-driven control using koopman operators,” *IEEE Transactions on Robotics*, vol. 35, no. 5, pp. 1071–1083, 2019.

[49] S. Young, D. Gandhi, S. Tulsiani, A. Gupta, P. Abbeel, and L. Pinto, “Visual imitation made easy,” *arXiv preprint arXiv:2008.04899*, 2020.

[50] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[51] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, “End-to-end differentiable physics for learning and control,” in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018. [Online]. Available: https : / / proceedings . neurips . cc / paper / 2018 / file / 842424a1d0595b76ec4fa03c46e8d755-Paper.pdf.

[52] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme, “NeuralSim: Augmenting differentiable simulators with neural networks,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021. [Online]. Available: https://github.com/google-research/tiny-differentiable-simulator.

[53] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[54] W. Zhao, J. P. Queralta, and T. Westerlund, “Sim-to-real transfer in deep reinforcement learning for robotics: A survey,” in *2020 IEEE Symposium*

Series on Computational Intelligence (SSCI), 2020, pp. 737–744. DOI: 10.1109/SSCI47803.2020.9308468.

[55] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3389–3396. DOI: 10.1109/ICRA.2017.7989385.

[56] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal, "Rapid locomotion via reinforcement learning," *arXiv preprint arXiv:2205.02824*, 2022.

[57] X. Ruan, D. Ren, X. Zhu, and J. Huang, "Mobile robot navigation based on deep reinforcement learning," in *2019 Chinese Control And Decision Conference (CCDC)*, 2019, pp. 6174–6178. DOI: 10.1109/CCDC.2019.8832393.

[58] J. Degrave, F. Felici, J. Buchli, *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, no. 7897, pp. 414–419, 2022.

[59] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[60] R. Kavishwara, Jun. 2021. [Online]. Available: https://medium.com/mlearning-ai/everything-you-need-to-know-about-reinforcement-learning-c7c2d333ed7a.

[61] D. Bloembergen, "Analyzing reinforcement learning algorithms using evolutionary game theory," Ph.D. dissertation, Jun. 2010.

[62] G. Papoudakis, F. Christianos, A. Rahman, and S. V. Albrecht, "Dealing with non-stationarity in multi-agent deep reinforcement learning," *arXiv preprint arXiv:1906.04737*, 2019.

[63] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, 2019.

[64] M. Jaderberg, W. M. Czarnecki, I. Dunning, *et al.*, "Human-level performance in 3d multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, pp. 859–865, 2019.

[65] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[66] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[67] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[68] E. Yang and D. Gu, "Multiagent reinforcement learning for multi-robot systems: A survey," tech. rep, Tech. Rep., 2004.

[69] D. Silver, "Lecture 2: Markov decision processes," *UCL. Retrieved from www0. cs. ucl. ac. uk/staff/d. silver/web/Teaching_files/MDP. pdf*, 2015.

[70] L. Weng, "A (long) peek into reinforcement learning," *lilianweng.github.io*, 2018. [Online]. Available: https://lilianweng.github.io/posts/2018-02-19-rl-overview/.

[71] H. Zhang and T. Yu, “Taxonomy of reinforcement learning algorithms,” in *Deep Reinforcement Learning*, Springer, 2020, pp. 125–133.

[72] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[73] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[74] I. Mordatch and P. Abbeel, “Emergence of grounded compositional language in multi-agent populations,” *arXiv preprint arXiv:1703.04908*, 2017.

[75] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *Neural Information Processing Systems (NIPS)*, 2017.

[76] J. K. Terry, B. Black, N. Grammel, *et al.*, “Pettingzoo: Gym for multi-agent reinforcement learning,” *arXiv preprint arXiv:2009.14471*, 2020.

[77] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, “Cython: The best of both worlds,” *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2010.

[78] M. Bonani, V. Longchamp, S. Magnenat, *et al.*, “The marxbot, a miniature mobile robot opening new perspectives for the collective-robotic research,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2010, pp. 4187–4193.

[79] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[80] G. Brockman, V. Cheung, L. Pettersson, *et al.*, "Openai gym (2016)," *arXiv preprint arXiv:1606.01540*, 2016.

[81] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, http://pybullet.org, 2016–2021.

[82] D. Hernandez, K. Denamganai, S. Devlin, S. Samothrakis, and J. A. Walker, "A comparison of self-play algorithms under a generalized framework," *arXiv preprint arXiv:2006.04471*, 2020.

[83] M. Jaderberg, V. Dalibard, S. Osindero, *et al.*, "Population based training of neural networks," *arXiv preprint arXiv:1711.09846*, 2017.

Appendix A

Derivations

A.1 Useful expectation laws used in the derivations

Law of iterated expectations

$$\mathbb{E}[Y \mid X = x] = \mathbb{E}[\mathbb{E}[Y \mid X = x, Z = z] \mid X = x] \tag{A.1}$$

A.2 State-value function derivation

State-value function full derivation

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&\text{Using the linearity of the expectation} \\
&= \mathbb{E}_\pi[R_{t+1} \mid S_t = s] + \mathbb{E}_\pi[\gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} \mid S_t = s] + \gamma\, \mathbb{E}_\pi[G_{t+1} \mid S_t = s]
\end{aligned}
$$

Using the law of iterated expectations (A.1):

$$
\begin{aligned}
\mathbb{E}_\pi[G_{t+1} \mid S_t = s] = \\
&= \mathbb{E}_\pi[\mathbb{E}_\pi[G_{t+1} \mid S_t = s, S_{t+1} = s'] \mid S_t = s] \\
&(G_{t+1} \text{ does not depend on } S_t \\
&\text{according to the Markov property of MDP)} \\
&= \mathbb{E}_\pi[\mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \mid S_t = s] \\
&= \mathbb{E}_\pi[V_\pi(s') \mid S_t = s] \\
&\text{where } s' \text{ is the next state}
\end{aligned}
$$

Then:

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[R_{t+1} \mid S_t = s] + \gamma\, \mathbb{E}_\pi[V_\pi(s') \mid S_t = s] \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma\, V_\pi(s') \mid S_t = s] \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} \mathbb{P}(s', r \mid s, a)[r + \gamma\, V_\pi(s')]
\end{aligned}
$$

(A.2)

Equation A.2 is the bellman equation for the value function (V_π) that defines the relationship between the value of a state (s) and the next states (s')

The following figure A.1 shows the backup diagram for the state-value function that describe the relation between the states, action, policy, rewards, and the transition probabilities.

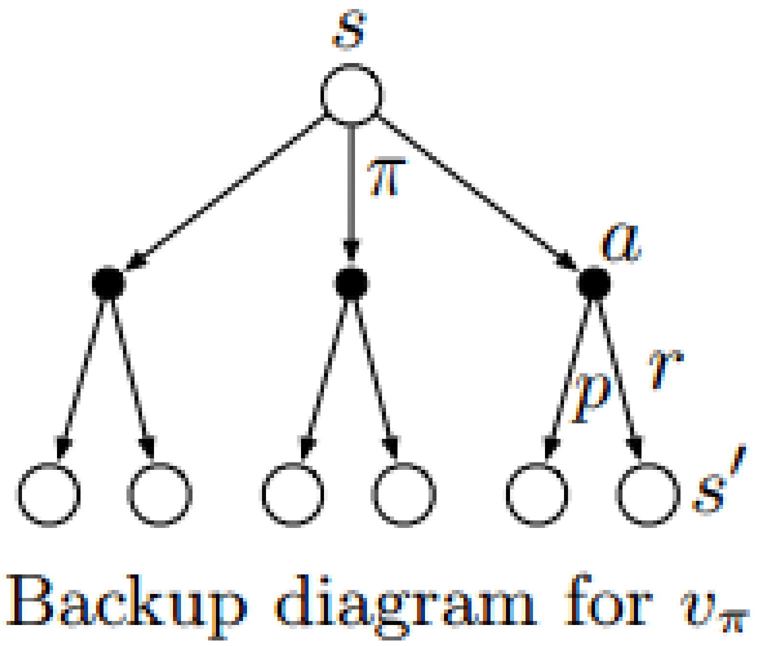


Figure A.1: [tmp] Value function backup diagram

A.3 Action-value function derivation

TODO Action-value function full derivation

.....

Then:

..... (A.3)

The following figure A.2 shows the backup diagram for the action-value function that describe the relation between the states, action, policy, rewards, and the transition probabilities.

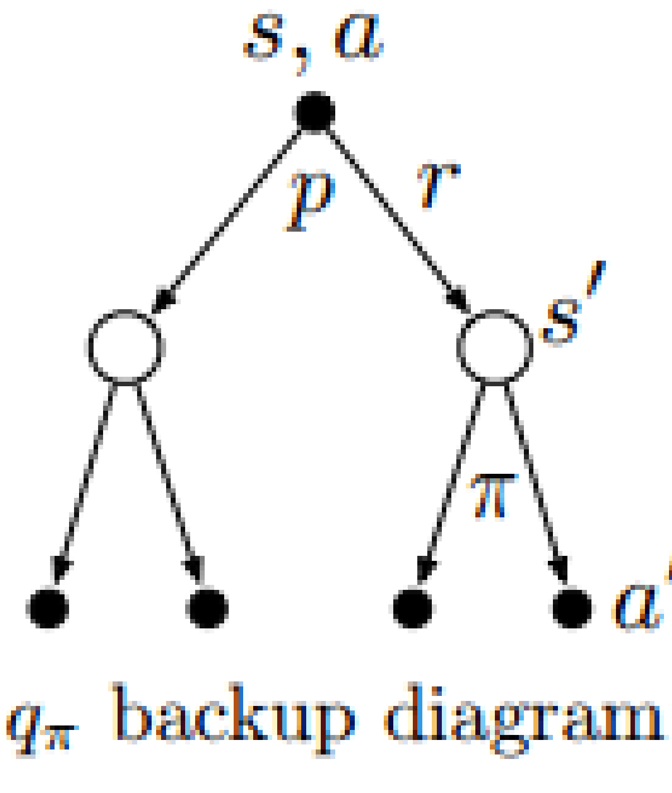


Figure A.2: [tmp] Action-value function backup diagram

A.4 State-Value and Action-value functions cyclic formula derivation

(A.4)

(A.5)

Appendix B

Further overview for more scenarios

Explain more about algorithms and other works, maybe mention more about game theory and RL formulation

Maybe add more interesting stuff about this problem, other scenarios and so on, mention similar problems that are relative

Explain more about similar problems and the big picture of RL and the levels of explanation and the general trends in this area now in part of self-play and similar

(More into the roadmap of the field not just continuation for this work but what is there in the world related to this in more point of view, more about the similar problems)

Appendix C

Deeper look on the implementation

Explain the code and the framework. And how to use it and the utils.

Appendix D

Thesis' ups and downs

More into the research process (reading papers, collect papers, implement, think, ...etc)

what did I try and did not work, how to debug, practice that I follow, what I did wrong, what I should not do and should have changed it

write drunk, edit sober

notion

use rl hp based from similar work or tasks

nuts and bolts for Rl

start writing a paper check accepted similar paper in a similar conference and follow their structure somehow

How to find an idea?

Tools and workflow

Twitter and other stuff

If time returned, what would I change in the thesis: