BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION Computer Science

DIPLOMA THESIS

Creating a competitive multi-agent environment for reinforcement learning

Supervisor
Conf. Dr. Mihai-Alexandru Suciu

Author
Pamfil Cătălin-Costel

2022

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA Computer Science

LUCRARE DE LICENŢĂ

Crearea unui mediu competitiv multi-agentic pentru învățare prin întărire

Conducător științific
Conf. Dr. Mihai-Alexandru Suciu

Absolvent
Pamfil Cătălin-Costel

2022

ABSTRACT

This paper was written as a study of reinforcement learning, and it lead to the creation of a multi-agent environment. The environment is an antagonistic one, where the agents try to eliminate each other, and it is written to respect the API provided by the library PettingZoo[27]. This library was used as it was created with the goal of aiding and standardising multi-agent environments, and it provides wrappers to aid in interaction with other reinforcement learning libraries. The first chapter represents a summary of this paper, highlighting the main aspect to be discussed and my own contributions to the subject at hand. The second chapter denotes a brief theoretical introduction of the main concepts used in the creation of my application.The third chapter proceeds to paint a general picture about the history of reinforcement learning, the main fields that used it, and what the future brings for this vast domain.The fourth chapter highlights the objective of my application and the main sources of inspiration for it. I also discussed some potential uses, especially in the evolutionary domain.The fifth chapter comprises of an in-depth presentation of my application, and the ingenious use of a somewhat new library in its creation.Last but not least, the final chapter provides an overview of the paper, and also discusses some implications derived from each individual chapter of the paper.

Contents

Chapter 1

Introducere

In this world, there are numerous instances where one entity hunts another, while the latter hides and tries to elude its own hunter. As a result of the struggle for survival in the real world, the animal kingdom has developed instincts and intelligence. The goal of this paper is to use artificial intelligence to understand and simulate this type of evolution, described above. This is done by creating a custom multi-agent environment with the help of an already existing library and training it. Moreover, I created this environment with the hope it may help others in the pursuit of developing their own environment, be it by improving this one or by studying it as an example.

The second chapter is an introduction to the basic notions needed for this paper. It includes general notions about machine learning and reinforcement learning, together with a description of the used algorithm, Proximal Policy Optimization, which is employed in training the agents. The chapter also presents the libraries and the necessary API utilised to create the environments used for reinforcement learning and those used for the training of the models.

The third chapter presents a view of both the past and present situation of reinforcement learning. It contains a brief look at the history of reinforcement learning and the current advances made in other domains using reinforcement learning. In this chapter there are also shown in detail cutting edge technologies created to solve and excel in games. All four games presented are of significance, as their complexity lead to great innovations being made. Two of them are classical games,namely chess and go, well known for their difficulty, due to the enormous number of valid positions required to be checked at every step. The other two, Starcraft 2 and Dota 2, are video games renown for the high skill ceiling at which they are played at professional level.

In the forth chapter, a more detailed explanation of the problem definition is given. It also maps the problem to real world situations, both in video game environments and natural ones, giving examples of how real world agents resolved it

and how the virtual environment can better replicate its real counterpart.

The fifth chapter is reserved for the accompanied application, presenting and in depth view of the design, implementation and testing, both for the environment and for the artificial intelligence. It provides a documentation of the code, giving an explanation of the functionalities and showing the code flow for each step taken while training.

Chapter 2

Theoretical notions

To understand the thesis fully and with ease the reader must be familiar not only with a number of theoretical notions pertaining to machine and reinforcement learning, but also with the main libraries used.

2.1 Machine learning

Machine learning has undeniably piqued the curiosity of many researchers and programmers in recent years. It is, however, not an unusual tendency. The Big Data phenomena grew in tandem with it. These two concepts are strongly intertwined and, in today's world, cannot be separated. Having said that, how can we define such a diverse field, which continues to expand every day? We can say with certainty that machine learning is a collection of methods that can create artificial intelligence. When we talk about machine learning, we're interested in how we can create computer systems that can learn from their own experiences and adapt and improve over time[12]. The algorithms are essentially simulating human intelligence in this way [10].

Machine learning has a wide range of applications, from autonomous vehicle control and spacecraft engineering to speech processing and pattern detection. It's clear to see how machine learning has drastically improved our lives. It has, for example, been successfully employed in the identification of spam email [30], rendering the problem (almost) obsolete. In other words, thanks to machine learning, nowadays we can solve problems that require learning (such as how to identify which plants in a field are diseased) much faster. In the past, this problem-solving process would have been much more costly, both in terms of wasted time and money.

Machine Learning ca be classified into three main branches of algorithms[25]:

1. Supervised Learning - it involves teaching a system, using labelled data as ex-

amples, to predict an outcome; the correct input is first provided by an external supervisor, somehow like a professor; the agent's main objective is to learn to extrapolate or generalize its responses; the biggest disadvantage for this type of learning is the fact that working with big data can often be an impossible task;

2. Unsupervised Learning - it involves uncovering a hidden pattern from a collection of unlabelled data; in this case, the machine tries to classify objects according to their similarities and differences; in some way, we can say that the agent's main task this time is to categorize and organize the inputs;

3. Reinforcement Learning - the main difference between this type of learning and unsupervised learning is that the former tries to maximize a reward;

Next, I will try to define, briefly, reinforcement learning and its underlying process.

2.1.1 Reinforcement learning

Reinforcement learning (or, in short, RL) is a method of machine learning, and it's concerned with the way natural (for example, animals) and artificial systems learn to predict the consequences of their behavior[8]. Based on these positive or negative consequences (rewards and punishments), the machine learns and optimizes itself. Thus, we can safely affirm that at the heart of reinforcement learning are the concepts of learning and decision-making [11].

The beginnings of this broad field of reinforcement learning lie in the field of behavioural psychology and in the early works of Pavlov and Skinner. Their studies of classical conditioning have laid the foundations for a domain which, in time, will acquire a very high level of interdisciplinarity. Reinforcement learning operates with ideas from areas like computer science, neuroscience and machine learning.

The basic principle of reinforcement learning is that humans and animals engage more often in behaviors that are reinforced (either through positive or negative reinforcement), and avoid doing those behaviors that were previously punished. This universal law of behaviour has diverse applications. Many of society's institutions and laws (e.g. fines and taxes) are built on this principle.

In simple terms, reinforcement learning is a process by which a given agent explores and interacts with an initially unknown environment. Through trial and error, the agent learns to select which actions to perform so as to maximise the rewards and minimise the punishments it receives. Over time, the agent's behaviour improves. According to one author [25], reinforcement learning can be characterised by three essential aspects:

1. Problems that are closed-loop, meaning that the agent's inputs have an impact on his later actions (it influences what courses of actions the system will take);

2. No person directly or indirectly directs or guides the actions that the agent takes. In other words, the agent is self-taught through trial and error;

3. In some cases, the consequences of the agent's behaviour have a long-term impact.

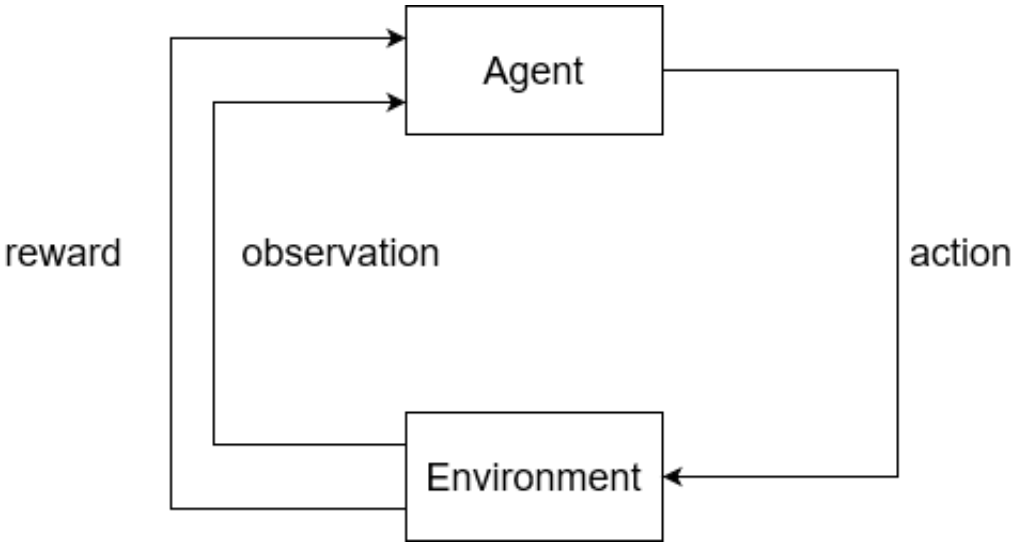

Figure 2.1: Reinforcement learning visual guide to the flow of a step

There are many intricacies through which we can describe the process of reinforcement learning. However, the scientific literature [25] mentions four elements that define a reinforcement learning system.

1. Policy — It is essentially a strategy that the agent adopts at a given moment; it connects the perceived states of the environment with the action the agent will take; it is, in psychological terms, the agent's reaction to perceived stimuli.

2. Reward signal — Mathematically, it is a numerical value. In practice, it represents the feedback provided by the environment to the agent as a result of its behaviour. The goal of the agent is to maximise the rewards it gets (and the corresponding numerical value) in the long run.

3. Value function — Values represent the total of expected rewards; the main feature is that values refer to something long-lasting, as opposed to rewards, which are short-lasting;

4. Model of the environment – The states that characterises an environment, from which predictions can be made (for example, about possible rewards). The

model has a very important role in reinforcement learning, because it's used by the agents for planning.

At this point, it is necessary to distinguish between model-based and model-free methods. Model-based methods allow us to identify a given model that influences the agent's policy in some way, whereas model-free methods rely on trial and error. As a result, one is more concerned with planning, while the other is more concerned with learning.

2.1.2 Proximal Policy Optimization

This family of algorithms, of policy gradient methods, was first introduced in this scientific paper[21]. At the time it came out, it was novel idea, as the algorithms performed were much simpler to implement than their counterparts, while not sacrificing performance as it compared favourably or even better than other policy gradient methods. As such, PPO has become the default reinforcement learning algorithm for OpenAI.

2.2 Machine learning Libraries

2.2.1 OpenAI Gym

Gym[4] is a python library that provides users with a huge number of environments created with the scope of standardising the training of artificial intelligence using reinforcement learning. It provides a standard API to facilitate the interactivity between environments and learning algorithms. The library was created with the goal of encouraging cooperation and peer review, as it provides users with means of comparing their algorithms and it encourages them to share the code and describe their approach.

An important improvement brought by the API for custom environments is the creation of a standardised API. Every environment must provide two attributes, namely env.action_space and env.observation_space, to ensure the validity of the actions and observations generated.In reinforcement learning, an agent performs actions in the environment, from which it then extracts observations and may even get a reward for the action taken. This is represented in the Gym API in the form of the step() function. This function receives as parameter an action, that must be valid when checked with env.action_space, and after it finishes operating in the environment must return 4 variables:

- observation - that must valid when checked with env.observation_space, and may be an array of values containing distances and velocities.

- reward - as in the reward that was resulted from the action taken.
- done - a boolean to signify the ending of the iteration, which can be returned for many different reasons, such as the completion of the situation or for exceeding the timeframe set.
- info - a dictionary that may be returned empty, as it pertains to debugging and logging. It may provide information about the state of the environment, variables that are not returned by the observation or whatever else the developer considers necessary.

A Gym environment must also provide implementations for the render() function, which can be used to display it in a format capable for human to understand it or in rgb_array form, which can be turned into video form. The reset() function also has to be implemented, as it resets the environment back to the initial state so it can be reused for training.

2.2.2 PettingZoo

PettingZoo[27] is a Python library that can be described as a multi-agent version of Gym, as it provides a standard API for multi-agent environments. It provides a number of already prepared environments, and it has wrappers that allow it to interact with most of the learning libraries that support Gym.

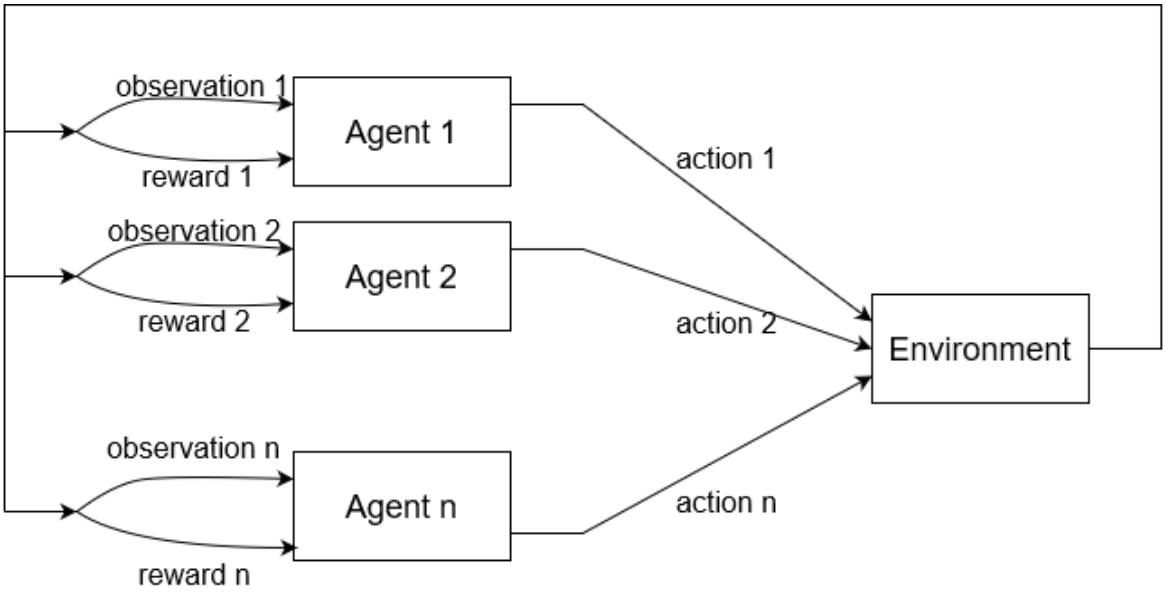

Figure 2.2: Multi agent reinforcement learning visual guide to the flow of a step

As it is also a reinforcement learning library for environments, the PettingZoo environment can be interacted with in a similar way to Gym. The reset() and the

render() functions have the same tasks as mentioned above, while the values returned by step() in Gym are returned in this case by the last() function, with step() only executing the given action and being tasked with switching control to the next agent. A small difference between the two libraries in the values returned by last() is that "done" does not signify that the environment is done, but only that the individual agent, as well as the returned reward is the sum of all rewards received by the agents since its last turn.

PettingZoo environments are considered as Agent Environment Cycles, where each agent takes its turn sequentially. Due to its nature of having more than one agent, the developers included lower level attributes and functions that can help with the implementation, with some of the more important ones being:

- agents - a list of all current agents. It can be modified as the environment progresses.
- agent_selection - this can be used as an index in the step function to know which agent takes the next step.
- observe(agent) - returns the observation of a specified agent, this function is usually called by last()
- dones - a dictionary that keeps track of the agents to see which one is done. It can be modified as the environment progresses.
- rewards - a dictionary that keeps track of the rewards earned by each agent in the last step, even if it was not its own. It can be modified as the environment progresses.

2.2.3 Ray

Ray[15] is an open source project created to ease the creation of any compute-intensive python workload. It aids in the implementation of several workloads, including reinforcement learning and hyperparameter tuning. Ray comes with a plethora of libraries that make it easier to implement distributed frameworks.

RLlib, the reinforcement learning library of Ray, provides a simple API that can be used not only for production level, highly complex reinforcement learning workloads, but also for starter projects, as you don't need to know the inner working of Ray or any of its libraries. This library provides a suite of features, as it supports both TensorFlow and Pytorch, the most popular deep-learning frameworks, has implementations for the most used algorithms, such as PPO and DQN, and provides support for environments made in Gym and PettingZoo, while also providing the chance to create them using this library's API.

Another useful library is Ray Tune[14], which allows for hyperparameter tuning at any scale while training. The algorithms provided by this library can maximize model performance while minimizing training costs. Its ease of use can cut down significantly on the time needed for hyperparameter tuning.

Chapter 3

Reinforcement learning - past and current trends

3.1 Brief history of reinforcement learning

To demonstrate how quickly the concept of reinforcement learning has evolved, but also to better understand what it is, I will summarize in the following paragraphs the stages that reinforcement learning has gone through to get to where it is today.

As stated before, the story of reinforcement learning begins in psychology, particularly in the field of animal learning. For the concept of reinforcement learning in computer science, Thorndike's ideas are of outmost importance. In 1898, the famous American psychologist formulated the Law of Effect, according to which a behaviour that receives an appetitive stimulus will be more likely to be repeated in the future, while a behaviour that receives an aversive stimulus will reduce in frequency.

Over time, reinforcement learning has also been associated with the theory of natural selection, according to which an organism with evolutionarily adaptive traits will be more likely to survive and reproduce (thus passing on their genes to their offspring) than an organism with detrimental traits [3]. Those traits that help the organism survive are 'conserved' through the mechanism of natural selection. Over the years, computational models have been created that draw their inspiration from Darwin's theory of evolution[20].

In tandem with the developments in psychology, in the field of mathematics another theory started to develop and catch the attention of following researchers, that is: optimal control. The concept of optimal control, introduced in the late 1950s[25] is based on the idea of achieving the best performance using the minimum amount of resources (including the shortest possible time) to achieve a particular objective[7]. Thus, a controller is created to provide input values to a system, in order to reach a

desired state. Numerous problems can be identified, both in the real world and in the research field, that can be analyzed using optimal control theory. Over time, several methods have been developed to solve such problems. One such set of methods refers to dynamic programming.

In an attempt to combine optimal control and dynamic programming on the one hand, and learning on the other, Werbos proposes the concept of heuristic dynamic programming[25]. It was the first attempt to integrate two branches of research that, until then, had been separated. Today, both dynamic programming and reinforcement learning are strategies for attempting to solve complex problems and optimizing artificial intelligence agents[5].

3.2 Current advances in reinforcement learning

With the emergence of reinforcement learning, millions of possibilities for optimizing existing systems or developing new applications to improve human life have emerged. The most significant advantage that reinforcement learning provides is the ability to identify the best solutions from an efficiency standpoint to a given problem. That said, what can we actually do with reinforcement learning?

A relevant example is provided by Mwiti[16]. Apparently, reinforcement learning was used in industrial automation, by a company named Deepmind. They used agents with artificial intelligence to reduce energy consumption at Google Data Centers. In short, the AI system makes several predictions per minute and determines the most effective actions it can take to reduce energy consumption while also meeting environmental constraints [6].

Another very interesting application was described by Barto et al. in 2017[1], and relates to the field of personalized web services and Internet marketing. This requires analysis of user activity and data collection. At the end of the process, a policy is created that is tasked with recommending content to the user. In time, the content the user receives will become highly personalized. Of course, the artificial intelligence can be optimised by implementing ways to receive feedback from the user.

Another frequent application of reinforcement learning concerns traffic control. There are many studies in the literature on this topic. Du and Ding (2020)[31] presents some of them. In summary, reinforcement learning can be used to reduce traffic congestions when it comes to the public transportation system, through the employment of adaptive traffic control systems.

Last but not least, I will talk about the application of reinforcement learning in robotics. Using artificial intelligence and the trial-and-error learning paradigm, agents (robots) get to perform complex behaviours. There are robots that can teach

themselves to walk, robots that can search and identify categories of objects, robots that can push and carry various boxes, and more. Robots learn to use a variety of skills in fields ranging from automation to medicine by exploring their surroundings and receiving rewards for appropriate and creator-encouraged behaviors.

As in any other area, there is much room for improvement and innovation. According to a Forbes article[26], reinforcement learning could be used in the following industries: entertainment, where characters (AI agents) will be able to improvise stories and interact with other characters; and medicine, where AI agents could be trained to recognize which treatment to give to which type of patient.

Of course, there are also challenges in reinforcement learning that will have to be overcome. According to Dayan and Niv [9], some of them include: the acquisition of hierarchical structures; integrating into reinforcement learning an idea that has long existed in psychology, namely that humans and animals do not "forget" (through extinction) what they have learned, but simply learn a new stimulus-response relationship; or the application of reinforcement learning algorithms in complex areas, such as cognitive neuroscience, which do not fully comply with known psychological principles and laws (investigated in highly stringent laboratory studies).

3.3 Reinforcement learning in gamified environments

Reinforcement learning is used in video games for two main reasons. On the one hand, environments designed for artificial intelligence training generally have many limitations and conditions. In other words, the environments are closed and do not allow for exploration, which is a critical compontent in reinforcement learning. Developing machines that can solve complex real-world problems necessitates testing and training in environments similar to those in which they will be used. Video games, particularly large-scale ones, provide chaotic, complex environments with many elements that must be observed and considered - in short, the ideal training ground.

Reinforcement learning, on the other hand, is becoming more popular in video game development. It appears to be a very useful tool in this regard. Reinforcement learning, for example, is currently used in game testing, where agents are taught to play the game and then identify potential bugs. There are limitations, of course, such as the fact that an AI learns to explore new maps with great difficulty (this is especially happening when the game has multiple levels). Professional players who want to improve their skills with the help of a strong opponent also use reinforcement learning.

Last but not least, another, more specific reason deserves to be mentioned. Human nature is incredibly competitive. As a result, we enjoy creating increasingly

difficult opponents to defeat, as well as breaking records and setting new upper limits to achieve. Reinforcement learning allows us to do just that.

3.3.1 Reinforcement learning in classical games

Throughout history, classic games have been ideal for testing new reinforcement learning methods and artificial intelligence systems. In the following sections, I'll go over a few games where artificial intelligence has made huge strides, breaking records and defeating some of the best players, masters of the games they play.

3.3.1.1 Reinforcement learning in chess

Chess is one of the oldest board games still played, the current form of the game dating back to the 15th century. It is an abstract strategy game with all the information being visible to both players, where the objective of the game is to place the other player in checkmate. The game is complex, but it has states and moves that are well-defined, and as such it requires creativity. This specific traits lead the game to be studied in depth since the start of the computer age[13], with many chess engines being created during the following years. In 1997 Deep Blue, developed by IBM, defeated the then reigning world champion Garry Kasparov in a six-game match, according to standard tournament rules. This marked the end of the era in which humans dominated chess matches, and it is now nearly impossible for a human to defeat a modern chess engine running on smartphone hardware.

2015 marks the year in which Matthew Lai proposes the first deep reinforcement learning model[13] to solve the game of chess. He acknowledges that computers are inefficient in the way they are designed to search for potential solutions, as Deep Blue had to search approximately 200 million positions per second, whereas Gary Kasparov could only search 5 positions in the same time frame. He argues this is due to the way the human brain works, as they can decide which branches don't need to be considered with a greater accuracy. He goes on further, saying that it is difficult to make attempts to code chess engines to be more selective while still retaining a high accuracy. This is because it is a highly complex and time consuming endeavour to convert human intuition and knowledge gained through experiences into concrete rules that can be written in code. Lai managed to train a network by giving it 175 million positions to train with, while also making Giraffe play against itself. He states that when tested on the Strategic Test Suite, the network reaches quickly a value of 6000, with a maximum of 9700 after training for 72 hours. While it wasn't the highest, he says it can be compared to the best chess engines at the time of publication, continuing that it " is remarkable because their evaluation functions are all carefully hand-designed behemoths with hundreds of parameters that have

been tuned both manually and automatically over several years, and many of them have been worked on by human grandmasters."[13]

In 2017 there was another milestone in reinforcement learning, as AlphaZero[24] was introduced. This algorithm was a generalization of the previously released AlphaGo Zero, with the aim of creating a model that can beat top level programs in different games, while starting with no knowledge and only the rules of said games. After testing, it was proved that AlphaZero was superior to Stockfish, the highest rated chess engine, as as it lost 0 games to it from a total of 100. While analyzed, it was shown that AlphaZero managed to discover on its own and frequently play the top human openings, with no prior strategy or knowledge being provided.

3.3.1.2 Reinforcement learning in go

Akin to chess, go is an abstract strategy board game, considered to be one of, if not the oldest board game still played today[33]. It is played with pieces named stones on a board that has a grid of 19 by 19 lines. This means there are 361 possible placements. The goal of each player is to surround and capture the pieces of the opponent or to capture spaces of the board. The difficulty of creating a competitive algorithm for go can be explained by 3 elements:

1. the total number of valid states for a board was calculated to be close to 10^{170}, which is even greater than the number of atoms in the observable universe (that is considered to be 10^{80}). Furthermore, to anticipate the next 4 moves a brute force algorithm would have to compute nearly 10^{11} possible combinations.

2. the placement of a piece can influence the flow of the game a hundred or more turns later, which would be impossible to exhaustively compute as a computer struggles to do so for only 4 moves as stated above.

3. compared to other games, in go there is no easy way to evaluate which player has a stronger position, as the number of pieces on the board does not take into consideration the territorial advantage, which in turn does not indicate who has an advantage granted by position and board influence.

According to their own article[23], the first computer program capable of competing with professional go players was AlphaGo. It was trained by exposing it to games between humans to create a baseline, then it was made to play against itself. Following this approach, in 2015 AlphaGo managed to beat the reigning European Champion, Fan Hui. In 2017 AlphaGo Zero was released, which trained itself solely by playing repeated games against itself. It managed to surpass its previous version, creating new strategies not discovered by human players and so it managed to

defeat the World Champions, Lee Sedol and Ke Jie. Later that year, AlphaZero[24] was released, managing to defeat AlphaGo Zero in 60 games out of 100.

3.3.2 Reinforcement learning in complex computer games

The goal of programmers and scientists is to create artificial intelligence that works in the most ecological contexts (environments similar to those found in real life) and solves problems close in complexity to those found in everyday life. Video games provide a suitable environment for training such AIs that match the above characteristics.

3.3.2.1 Reinforcement learning in Starcraft 2

Starcraft 2 is real-time strategy (RTS) video game, and one of the biggest Esports title. The game allows you to play with three species that have vastly different buildings, units, features and behaviours. From a gameplay perspective, at a macro level it provides a need for complex planning while at a micro level it provides an opportunity for fast paced action. It is both a game for resource management and strategy while also having battles influenced by the micromanaging of individual units. As such, it can be said that "from a reinforcement learning perspective, StarCraft II also offers an unparalleled opportunity to explore many challenging new frontiers" as noted in this article[29].

The authors that previously released AlphaZero and all of its earlier versions were the first to create an AI capable of reaching the grandmaster league in Starcraft 2, which is the highest. AlphaStar[28], as the artificial intelligence is called, is very remarkable, as it can play the game like a normal player, exemplified by the fact that it competes with other players on the game's official server, following the same conditions as human players. To achieve this it was needed to place constraints on it, to limit it to human capabilities. The constraints consists of making the observation space the same camera that a normal player sees, and limiting the number of actions per minute it can achieve to human levels. AlphaStar was also trained to be capable of playing as any of the three factions, each one being a different neural network, and it is capable of going against any of the three factions.

The paper states that simple self-play used in reinforcement learning has a big drawback in that as it improves upon itself, it may forget winning strategies learned against a previous version of itself. What is more, the authors discover that to make strong agents they need to employ a training behaviour used by real players when they are improving themselves. In the real world, a player trying to improve a specific strategy may call upon the help of a friend to help him in a friendly match, where the friend tries to expose the flaws of the player. To resolve this issues, the

network was trained both against older versions of itself and against agents that were fighting with the same strategy every time, not trying to improve. However, there was another problem that was identified, which was caused by the high number of possible actions the agent could execute at any step. Compounded to it, there was an issue that each game was pretty lengthy, which means the agent would execute thousands of actions until the end, where it would find out if it won. To solve this, they provided the network with knowledge of human strategies through the use of imitation learning.

3.3.2.2 Reinforcement learning in Dota2

Dota2, the biggest Esports title, is a multiplayer online battle arena (MOBA) video game [32]. A sub-gerne of the RTS, MOBAs represent the perfect arena for training game AIs [22], because it's focus is on managing your units, strategizing, planning and finding the best path to victory.

According to Wikipedia, in Dota2 there are two teams that start the game at opposite ends of the map. The goal of each team is to destroy a structure (called an "ancient") that is in the enemy team's base. Each team has 5 players, and each player chooses a "hero" to control throughout the game. There are several classes of heroes, each with unique abilities and specific strengths and weaknesses. At the heart of every round of Dota2 is cooperation and communication between members of the same team. Every minute, decisions need to be made and actions need to be taken.

OpenAI Five it's a team of five neural networks that managed to defeat Dota 2 world champion team. This victory represents a milestone for artificial intelligence and highlights the limitless possibilities of reinforcement learning. Because recurrent networks have an important limitation, in that such networks forget, after some point, what they have learned (in the process if sequence learning), OpenAI Five uses LSTM(long short-term memory) networks, a new and improved variant of recurrent networks. The input for each OpenAI single-layer LSTM network is received from Valve's Bot API. At first, according to this article[17], OpenAI learned from self-play – by playing against itself and against past versions of itself. In time, by using various randomization techniques and increasing them as needed, OpenAI Five started beating humans in one versus one. The learning process was accelerated by using many rewards obtained by taking diverse actions, from killing somebody, to assisting the teammates, and so on. In sum, the agent receives information from the environment, afterwards it takes an action that can be rewarded positively or negatively. Using this feedback, OpenAI Five updates itself and adapts in the long run. It's also worth mentioning that the policy uses Proximal Policy Optimiza-

tion (PPO) for training[2].

The results of OpenAI Five are impressive. It can register information at a much higher speed and has a much faster reaction time, for example. It's a very promising instrument, trained in a chaotic environment that, in some ways, mirrors the real world.

Chapter 4

My application

4.1 Problem definition

The program's goal is to create a multi agent environment that recreates a real-life situation in which agents try to find, identify and hunt each other. The environment is similar to a game of bumper cars, where cars try to hit other cars. However, the agents have to respect one condition. There is a restriction on the cars they can hit, as each car only has one target. Furthermore, there are walls on the field, that serve to prevent cars from passing or from seeing past them. The environment is also accompanied by a model that can be used as an example for training.

4.2 Projection of the problem into the real world

The concept for the environment evolved from a combination of various games. The initial concept was then generalized. The inspiration for the idea came from a YouTube video by CodeBullet[1], in which he trained a car to race down a track using reinforcement learning. This video motivated me to write a thesis about machine learning, specifically reinforcement learning. Specifically, I decided to recreate an environment akin to a racetrack. The next step was to incorporate bumper cars into the concept, as there have been numerous experiments with cars with the goal of achieving autonomous vehicles, but not as many with cars purposefully attempting to hit other cars. The final component that I integrated into the original idea was inspired by the main concept of a specific type of multiplayer game in which each player is assigned to find and remove another player from the game (very similar to a game of hide and seek). The end result is a multi-agent environment in which each agent must hunt a target, while also being hunted.

[1] link catre video

4.2.1 Gaming applications

A number of games have main gameplay loops that are structured similarly to the proposed environment (described above). Most of these games are classified as social deduction games, as usually, in these type of games, you are in a team that has to eliminate the other team members by identifying them and hunting them down. It is important to note that in this context, a "group" can mean different things. For example, it can also refer to groups of individual players who, in turn, may have more than one group as targets. That being said, I will now provide some examples of similar games that served as inspiration for the application I developed.

Assassin's Creed is a video game series in which you play as an assassin in a fictional history, tasked with eliminating various fictional or real-life figures. The Assassin's Creed: Brotherhood installment includes a multiplayer mode in which you are assigned a target and must find and kill him without blowing your cover. In the game, non-playable characters roam the maps as civilians alongside the players. The main gimmick of the game is that the models of the players and the non-playable characters are identical, making them indistinguishable. This allows the players to blend in and hide among the crowds. This puts you in the position of having to act like the artificial intelligence around you, while also analysing the characters around you in order to figure out who is another (real) player and what their intentions are. Murdering the wrong character exposes the player's identity, thus making him easier to recognize by either their pursuer or target. This ultimately leads to the development of sophisticated strategies and behaviours by the player, to help him avoid giving himself away.

Another game with a similar concept is The Ship, developed in 2006, which added an extra limitation by requiring you to murder your victim while no one was watching. This implies that the player must devise an even more complex strategy, in which he must avoid being killed by other passengers on the boat while also completing his own goal of killing his target.

Among Us is a multiplayer social deduction game that became extremely popular in 2020 as a result of its board broadcasting on social media platforms. Its gameplay is based on the party game Mafia, while the main theme and the atmosphere are inspired from the horror film The Thing. In this game, the players are divided into two groups, each with a different role: on the one hand – the crew members, on the other hand – the impostors. Each team is tasked with eliminating the other group. As in the previous games, the players have the same models, but the groups are balanced differently. Although the crew members outnumber the impostors, which gives them a slight advantage, they still start the game with an unknown variable and extra objectives that need to be achieved. More exactly, they

don't know the identities of the impostors and they have to complete tasks around the ship. Moreover, the latter (having to complete various tasks) make the crew members split. Being alone also means being more vulnerable to possible attacks from malicious players – the impostors. Also, while there are fewer impostors, they have two important advantages: they know the true identities of everyone, and they can quickly seize the initiative and kill members of the opposing team when it is in their favour. The crew members win by voting the impostors out or accomplishing tasks, which creates an artificial time limit for the impostor team to act in.

4.2.2 Real world situations

The proposed environment can be mapped to several simplified real world situations. In a broader sense, the presented problem can be used to represent any situation in which there are teams who are hostile to other specific groups. The agents that can be observed in the real world have evolved complex behavior to increase their own chance of survival. This was accomplished not only by honing the strategies and tactics used to bring down their prey, but also by honing the strategies and skills used to flee and confuse their hunter.

The natural world, and specifically the food web, is the best illustration of such an evolutionary environment. All life forms are constantly fighting for survival in order to obtain the energy they require to thrive, with many of them preying on others, placed lower on the food chain. Because every species' purpose is to live as long as possible, advantageous traits have evolved to benefit them and give them advantages when used in combination with their strategies. For example, numerous animals have developed the capability to camouflage in the environment and to blend in so that they are more difficult to spot, such as insects that resemble wood, or arctic animals whose fur evolved to be white to blend in with the snow. On the other hand, other animals, particularly predators, developed superior senses in order to increase their chances of being successful on a hunt. Furthermore, yet another example from the natural world can be found in the body of every living being. In the case of humans, the white cells protect us by hunting and neutralising viruses and bacteria, which in turn seek to reach their desired cells so they can infect them.

This type of antagonistic situation that drives the need for evolution can also be found in several other domains in which humans are involved. It can be said that the whole society is balanced between antagonistic and altruistic situations. The field of economics is a prime example for this, as we can observe situations where companies, especially at the level of corporations, try to get as large a share of the market as they can, even if it means driving another company out of business.

Another perfect example can be found in warfare. When talking about a war, we can usually identify two sides, the goal of both of them being to preserve as much manpower and to take as little damage as possible while making sure the other side is stopped.

4.2.3 Environmental features ideas for better specialization

As it can be seen from the previous examples, this simplified problem from which we started our analysis can be expanded to better reflect specific real world situations, especially if more variables that change the number of possible interactions between agents are included. While it is impossible to recreate an exact replica of the real world, virtual agents can depict a probable evolutionary route for their counterparts. Adding more types of agents with different behaviors and goals, establishing groups of the same type of agent, and modifying the characteristics of individual agents in a group that works together are just a few examples.

Chapter 5

Application code and design

5.1 Game environment

5.1.1 Design

For environments of PettingZoo[27] type created with the scope of training agents it is necessary to implement 5 functions which will be called when in training. The first 2 functions are the initialization and reset() functions, tasked with creating a start point for the game. The most important function is step(), which contains the logic that shows how the agent interacts and changes the environment. The action taken by the agent is derived from the observation it receives from the observe function(). Finally, the render() function is the method by which the user can observe the behaviour of the agents in the environment.

As the environment is a game with cars that can hit other objects, cars that try to hunt their targets, it needs to represent elements and behaviours the objects follow. There are two type of entities needed, cars and walls, which represent agents and barriers. As the agent is a car, it needs to respect the way a car moves in two dimensions when seen from above. The agent is expected to be able to accelerate forward or backwards, with the possibility of turning left or right at the same time. As in real life, cars in the environment can crash into the wall or other cars. In the case of an intersection with a wall, then the car will be stopped, while when a car hits another car they are both consider destroyed and must be reset. The agents need to be able to return observations made about the environment, such as distances to objects around it, the types of those objects, information about its state and information about its target in case it can see it.

5.1.2 Implementation

From an implementation perspective, the two types of objects, namely cars and walls, can be represented as classes. The wall can be considered as a simple line and implemented as such, while the car class requires more functionalities as it interacts with the environment in complex ways. Since the perspective of the field is from above, we consider the car a rectangle, defined by its center, angle, width and length. From this variables it is also possible to obtain the corners of the vehicle when needed for calculations.

The first important functionality of the car is the ability to move in both dimensions. This initially is done by receiving an action, expressed by an integer with values between 0 and 8, which can be then translated into the actual movement the car is to make. The car stores internally as variables its own velocity and angle. The velocity is modified by accelerating forwards or backwards, or by breaking when it tries to accelerate in the direction opposite to the velocity. When the turn action is taken the angle of the car modifies with the set turning angle. If the car is not moving when it receives the turn action then its angle will not modify. After the new velocity and angle of the car are calculated, the velocity is split in 2 vectors, one for each axis, so the new center position can be calculated.

A second functionality of the car is that it able to interact with the walls and the other cars by hitting them. The forms of the entities are simplified, the car being considered as a rectangle formed from 4 lines, while the wall is a simple line between 2 points. To check if the agent hit a wall the car checks if any of the four lines it is described by intersect with the line of any of the walls. If any wall was hit, the car is moved to its previous position where it did not overlap with any other object and its velocity will be reset to 0, as it is not considered completely destroyed. If the car hits a wall multiple times consecutively, then it will be reset to one of the possible reset points. To check if a car hit another car then, for all other cars, we check if any 2 lines of different cars intersect. If they did, then both cars have to be reset, and the rewards and punishments of the cars will be dealt depending on the targets of each car hit. When a car is reset following a crash, we remove a life, which brings it closer to being done and unable to get more points. It also gets moved to a new starting position from a pool of possible positions, after the new position is checked to ensure it does not intersect with another car.

A third functionality of the car is the ability to examine the environment and make observations about the objects it is surrounded by. It returns two sets of data, which are the can be divided into information about what it sees and information about the target it has. The way in which the car sees is similar to a lidar system, using elements akin to laser rays to detect distances between itself and other ob-

jects. As such the first step is to create segments with the starting point given on the car, and the end point being calculated from the expected angle of the ray and the maximum size of the ray, which represents the maximum range of vision of the car. After the ray is created, we check all walls and all sides of each car if we find an intersection, and if we do how close it is to the car. When we find the closest point we save the distance to it and the type of object it is, so that we can pass it when observe() is called. The car also keeps track of its target, as at each step it checks to see if the target is in its field of vision. It does so by taking the center and corners of the target car, checking if the points are in the visual cone and insight sight range. Then it checks if points are visually blocked by any walls. At the end it takes the center of the remaining points and returns the distance to that point and the angle it forms with its own direction.

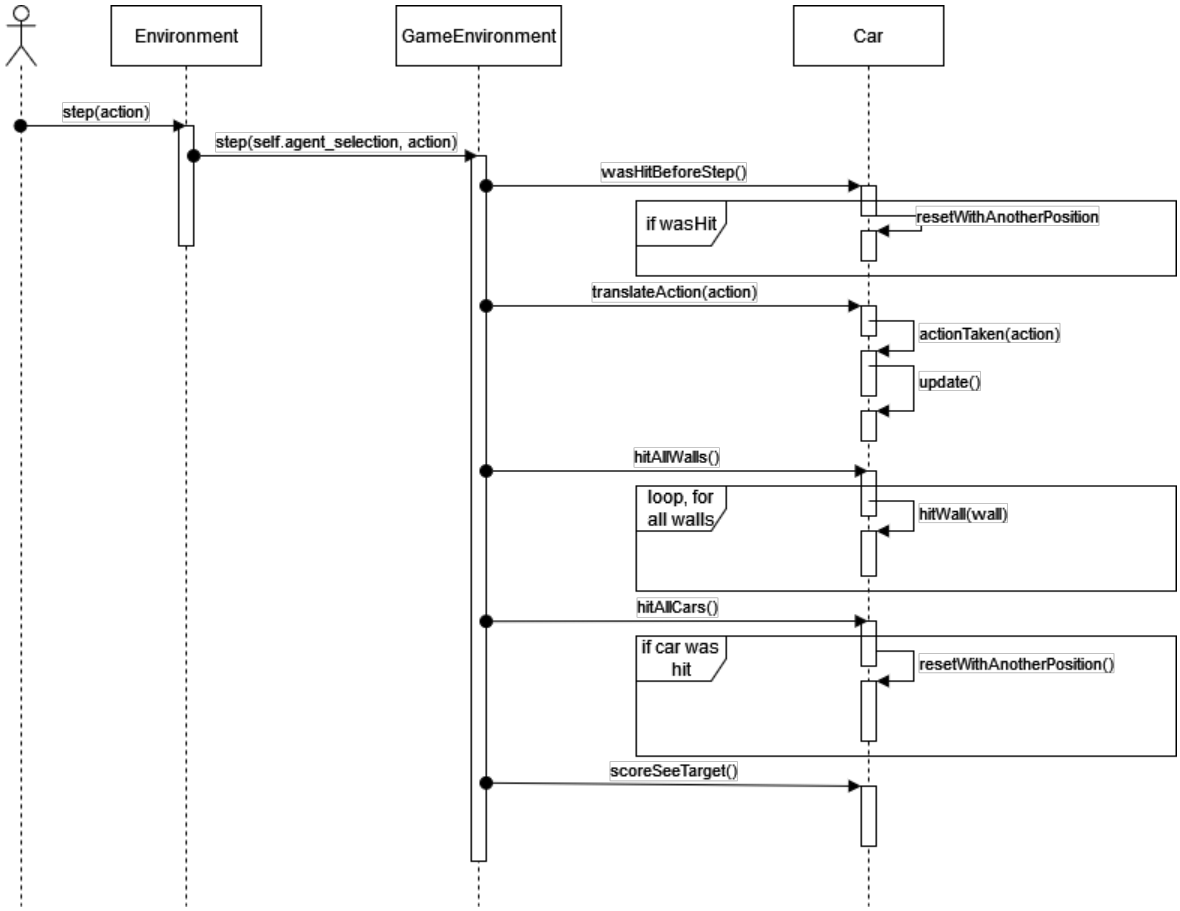

Figure 5.1: Sequence Diagram for step function

For the environment we need a class tasked with implementing the logic between the objects and the agents. It is implemented such that it can pass to the PettingZoo environment all the information it needs for the 5 functionalities used in training. The environment initialises the barriers and the agents, giving them their targets and keeping track of the possible starting positions and of the number of

cars that are done. For the observe function, when given an index of a car, it returns the velocity, angle and the observation made by the car. For the step function we combine the functionalities of the car. At the beginning it checks if the car was hit after its last step and before this step, then it moves the car with the given action. Since the position of the car corners was changed , it has to check if it hit any of the other objects and if it did, it follows the rules in such case. At the end it saves the score and checks if the car is still alive or it is done. Figure 5.1 shows a sequence diagram of the step function when called from the Environment class.

The render function has two modes, one that is more simplistic and one that shows the rays used by the cars to see around. The simplistic mode, as shown in figure5.2 only shows the field, the walls and the cars, each car being drawn from an image that is rotated depending on the angle of each agent. The second mode, as shown in figure5.3, not only shows the previous elements, but also the rays by which each agent can see. The color of the ray also indicates if at the end it sees a wall or another car, blue being for the former while red for the latter.

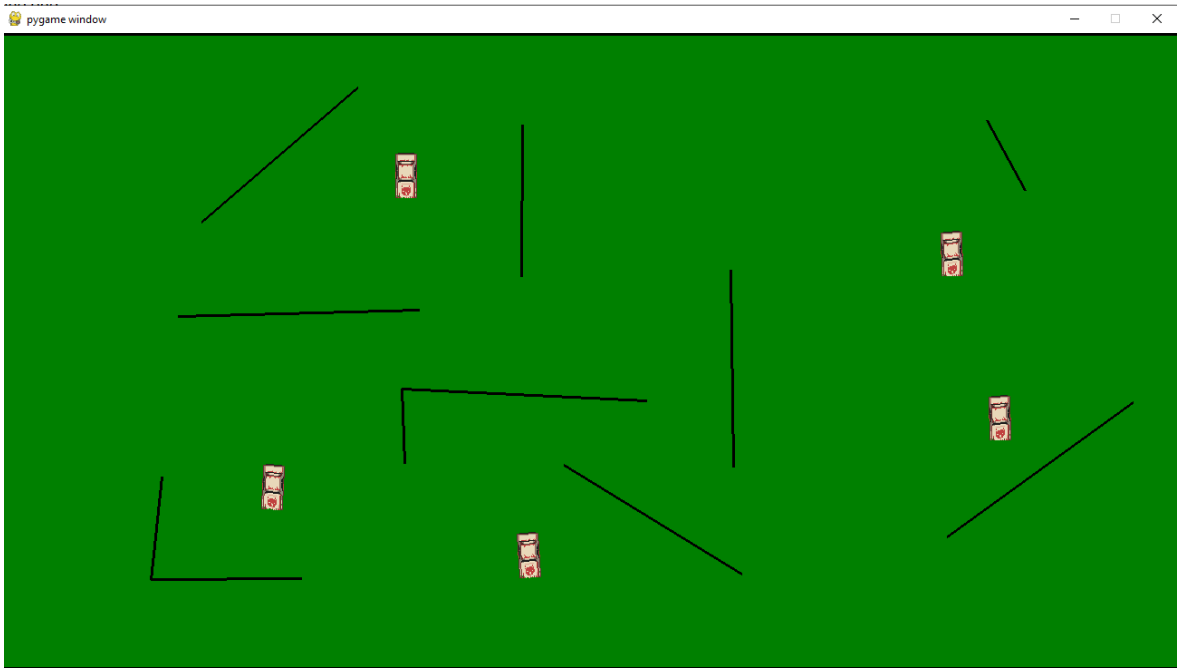

Figure 5.2: Simple render mode

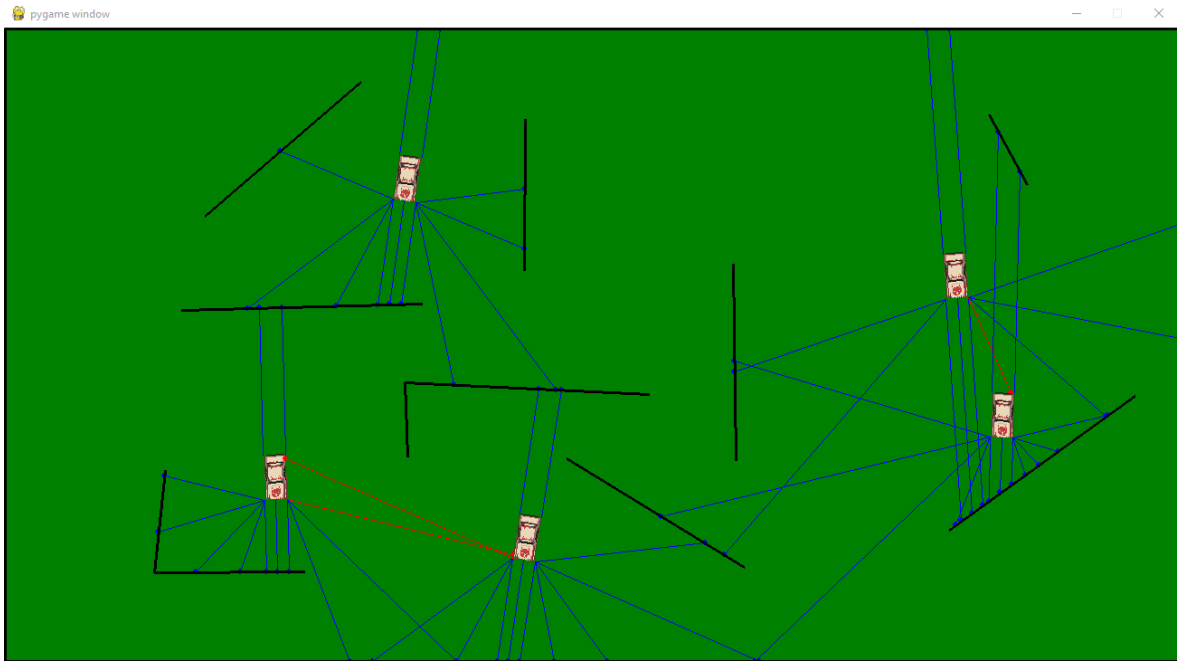

Figure 5.3: Sight render mode

5.1.3 Testing

Testing was done preponderantly through exploratory testing and using the API test provided by the PettingZoo[27] library. A new environment using the library's API has to follow a number of features and requirements. Given the need to ensure the consistency in rapport to the API, the library provides a test that can be made to run for a given amount of cycles that checks if the output is consistent with the API. Furthermore, the library provides a benchmark test which prints the number of cycles and steps took by the environment in 5 seconds. However, this provided tests only check the validity of the code, but not the correctness. As such, exploratory testing was needed to ensure that the functionalities are working as expected, and to find any bugs.

5.2 Artificial intelligence model

5.2.1 Design

From a design perspective, when creating the design of a model for an artificial intelligence it is necessary to keep in mind the form of the input data, the hyperparameters, the structure of the model and, specifically for reinforcement learning, the rewards and punishments given. In the case of reinforcement learning, the data cannot be gathered and given before the start of the training, as it is generated from the

environment upon the actions of the agent. As such the agent makes observations of the environment, from which an action can be obtained after computation from the model. As the proposed environment is a multi-agent one, each individual agent can be made to train with a different policy. However, in this paper the proposed model is used on all the agents, which can speed up the training.

5.2.2 Implementation

The proposed environment is written using PettingZoo[27] as its API is compatible with numerous reinforcement learning libraries, including Ray[15] and stable baselines3[19]. To show the simplicity of using the environment, the training was done using the Ray library, which also provides certain features useful for beginners to help them in their discovery of reinforcement learning.

The model used was created using a generic pytorch[18] model provided by the ray library, which already has implemented the neuron layers and the required forward() and value_funtion(). To teach the model in the provided environment, Ray not only provides a number of trainer functions which can be used with their default hyperparameters or given ones, but it also provides access to the tune()[14] function. RayTune is a hyperparameter optimization library, and as such it can optimize the hyperparameters of the model while at the same time training it, which can help to quickly increase its performance. The algorithm selected for training the models is Proximal Policy Optimization, or PPO for short, which has become of the most utilised reinforcement learning algorithms for its ease of use and good performance.

For the dataset, at each iteration the environment provides each agent with an observation, returning the position of the agent, the objects it can see, the distances from them and their types, and information about the target if it is in the field of view of the the agent. Since for the distance a car returns observations for up to the given max distance, while for the object type there are only two possible values, the model may learn to give too much weight to the distance in detriment to the type. As such, it was necessary to normalize the data by making all possible observations positive, and then adjusting them so they are all brought to the same scale, with values between 0 and 1.

Another important element of any reinforcement learning model is found in the reward structure provided by the environment that helps with the reinforcement of positive behaviour. For this program, the types of rewards can be split in three categories. The first category consists of the rewards and punishments awarded to the agents upon their intersection with other elements of the environment. When hitting a wall, the car receives a small penalty, which escalates to a bigger punish-

ment upon consecutive intersections with the wall. When two cars intersect, the rewards for each agent depend on their own target. In a crash, only the hunter receives points if successfully managed to find its quarry, and the reward in such an instance is significantly larger than any other possible reward. Any other instance deducts points for the agent, to encourage the behaviour of not hitting cars. The second category is directly connected to the identification of the target. An agent receives points for placing the target closer to the its direction and for closing the distance between them. The last category is similar to the one before it in that the agent receives a small reward for approaching the target, but the difference is in the fact that the agent does not know the position of the target. This reward has the goal to motivate the hunter to roam the map to find its prey.

5.2.3 Testing

The testing of the model training is done in two steps. The first step consist of the tune() function, which, for each iteration, it keeps track of the rewards received by each agent and creates statistics from them. It returns the maximum, minimum and mean reward attained in the iteration, together with the mean of the iteration length in steps and other information. In the table5.1 we can see the evolution of the model in 1000 iterations. To help with understanding the table, for this training there were 5 agents, each had a maximum of 2 lives and could make at most 201 steps. The reward for successfully hitting its own target for an agent is 50000 points.

Iteration	Max reward	Min reward	Mean reward	Mean length
1	-658	-658	-658	1005
100	250309	-650	79597	880
200	149970	46048	70157	850
300	151059	45559	87141	736
400	150294	46709	85046	790
500	1999761	47327	109650	708
600	203911	-3647	112210	642
700	199892	-2783	113974	602
800	200045	47742	116859	623
900	202047	-1239	118171	682
1000	199928	47887	112113	715

Table 5.1: Model progress in 1000 iterations

The second step of testing was manual, as I, the developer, had to interpret the returned statistics from the first step. This was done in tandem with a simulation of an iteration using the trained model, visualized with the help of the render() function found in the environment's API. This type of testing was necessary to help

with the development of the reward structure, as prior tests have shown that the agent were not motivated to hunt for their target.

Chapter 6

Conclusions

Finally, I'll summarize the main points raised in this paper. The goal of the program I designed for this thesis is to replicate a situation that occurs frequently in real life, particularly in the animal kingdom: the interaction between a prey and a predator.

In the first chapter of this paper, I briefly outlined the main topics that I have explored extensively throughout this paper.

The purpose of the second chapter was to introduce the reader to the key concepts and theoretical models employed in this paper. All subsequent arguments are built on these notions, which constitute the paper's backbone structure. They were first presented theoretically, and then implemented practically in the application I developed. As a result, I talked about the relationship between machine learning and reinforcement learning, Proximal Policy Optimization and its key applications, and the libraries I used to build the app: OpenAI Gym, PettingZoo, and Ray.

The third chapter attempted to provide an overview of the concept of reinforcement learning. The brief history offered in this section is crucial since it demonstrates the concept's rapid growth, and the many applications it had over the course of history. Reinforcement learning has been applied in a variety of fields since its development. It has been used to improve the performance of a variety of applications, including in fields such as robotics and economics. As such, it's essential to understand the major stages reinforcement learning went through. Moreover, I described some possible future avenues for growth. I also discussed the use of reinforcement learning in video games. I stated that video games are an important way for us to develop machines with artificial intelligence because they provide us with an environment similar to real life. Remembering history is important because it highlights the current advances we have in the present day. Moreover, based on this history, we can plan for the future.

The fourth chapter provides an overview of the application's concept. I talked about the program's goal and the main sources of inspiration for it. I also summarized some of the ways in which this application can reflect real life. I mostly talked

about evolution and the importance of adaptation and natural selection in real life. I believe that my application, particularly if improved in the future by adding variables that increase the complexity of agent interactions, has the potential to provide insight into the type of evolution found not only in the animal kingdom, but also in modern human-dominated domains.

The fifth chapter is the most practical. In it, I present my application in detail. I have described at length how I created the application's environment so that interested people can adapt and improve it further. Using the ideas presented in the previous chapter, the environment can be transformed to better reflect the needs of the developers that decide to use it. At its core, my application provides a basic framework for antagonistic environments where the agents try to eliminate each other.

Bibliography

[1] Andrew G Barto, Philip S Thomas, and Richard S Sutton. Some recent applications of reinforcement learning. In *Proceedings of the Eighteenth Yale Workshop on Adaptive and Learning Systems*, 2017.

[2] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[3] Matthias Borgstede and Frank Eggert. The formal foundation of an evolutionary theory of reinforcement. *Behavioural processes*, 186:104370, 2021.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2017.

[6] Jim Gao Chris Gamble. Safety-first ai for autonomous data centre cooling and industrial control, 2018.

[7] Guillaume Crabé. Optimal control, a preamble, 2021.

[8] Peter Dayan and Yael Niv. Reinforcement learning: The good, the bad and the ugly. *Current Opinion in Neurobiology*, 18(2):185–196, 2008. Cognitive neuroscience.

[9] Peter Dayan and Yael Niv. Reinforcement learning: the good, the bad and the ugly. *Current opinion in neurobiology*, 18(2):185–196, 2008.

[10] Issam El Naqa and Martin J Murphy. What is machine learning? In *machine learning in radiation oncology*, pages 3–11. Springer, 2015.

[11] Katja Hofmann. Reinforcement learning: Past, present, and future perspectives. In *Thirty-third Conference on Neural Information Processing Systems (NeurIPS)*, December 2019. Tutorial.

[12] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[13] Matthew Lai. Giraffe: Using deep reinforcement learning to play chess. *CoRR*, abs/1509.01549, 2015.

[14] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[15] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging ai applications, 2018.

[16] Derrick Mwiti. 10 real-life applications of reinforcement learning, 2021.

[17] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[19] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[20] Matthew Roos. Evolutionary approaches towards ai: past, present, and future, 2019.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[22] Victor do Nascimento Silva and Luiz Chaimowicz. Moba: a new arena for game ai. *arXiv preprint arXiv:1705.10443*, 2017.

[23] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search, 2016.

[24] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[25] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[26] Tom Taulli. Reinforcement learning: The next big thing for ai (artificial intelligence)?, 2020.

[27] J. K Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sulivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall L Williams, Yashas Lokesh, Ryan Sullivan, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*, 2020.

[28] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 2019.

[29] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John P. Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ek-

ermo, Jacob Repp, and Rodney Tsing. Starcraft ii: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017.

[30] Kiri Wagstaff. Machine learning that matters. *arXiv preprint arXiv:1206.4656*, 2012.

[31] Shifei Ding Wei Du. A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications, 2021.

[32] Wikipedia contributors. Dota 2 — Wikipedia, the free encyclopedia, 2022. [Online; accessed 8-June-2022].

[33] Wikipedia contributors. Go (game) — Wikipedia, the free encyclopedia, 2022. [Online; accessed 8-June-2022].