FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FC Portugal - High-Level Skills Within A Multi-Agent Environment

Margarida Santiago

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Luís Paulo Reis

Co-Supervisor: Nuno Lau

September 17, 2021

Resumo

Ao longo dos anos a RoboCup, uma competição internacional de robótica e da inteligência artificia, foi palco de muitos desenvolvimentos e melhorias nestes duas áreas científicas. Esta competição tem diferentes desafios, incluindo uma liga de simulação 3D (Simulation 3D League). Anualmente, ocorre um torneio de jogos de futebol simulados entre as várias equipas participantes na Simulation 3D League, todas estas equipas deveram ser compostas por 11 robôs humanoides. Esta simulação obedece às leis da física de modo a se aproximar das circunstâncias dos jogos reais. Além disso, as regras da competição são semelhantes às regras originais do futebol com algumas alterações e adaptações. A equipa portuguesa, o FC Portugal 3D é um participante assíduo nos torneios desta liga e chegou até a ser vitoriosa várias vezes nos últimos anos, no entanto, para participar nesta competição é necessário que as equipas tenham os seus agentes capazes de executar skills (ou habilidades) de baixo nível como andar, chutar e levantar-se. O bom registo da equipa FC Portugal 3D advém do facto de os métodos utilizados para treinar os seus jogadores serem continuamente melhorados resultando em melhores habilidades. De facto, considera-se que estes comportamentos de baixo nível estão num ponto em que é possível mudar o foco das implementações para competências de alto nível que deveram ser baseadas nestas competências fundamentais de baixo nível.

O futebol pode ser visto como um jogo cooperativo onde jogadores da mesma equipa têm de trabalhar em conjunto para vencer os seus adversários, consequentemente, este jogo é considerado como um bom ambiente para desenvolver, testar e aplicar implementações relativas a cooperações multi-agente. Com isto em mente, o objetivo desta dissertação é construir uma setplay multi-agente baseada nas skills de baixo nível previamente implementadas pela FC Portugal para serem usadas em situações de jogo específicas em que a intenção principal é marcar um golo. Recentemente, muitos participantes da 3D League (incluindo a equipa portuguesa) têm desenvolvido competências utilizando métodos de Deep Reinforcement Learning obtendo resultados satisfatórios num tempo razoável. A abordagem adotada neste projeto foi a de utilizar o algoritmo de Reinforcement Learning, PPO, para treinar todos os ambientes criados com o intuito de desenvolver a setplay pretendida, os resultados dos treinos estão presentes no penúltimo capítulo deste documento seguidos de sugestões para implementações futuras.

Abstract

Throughout the years the RoboCup, an international competition of robotics and artificial intelligence, saw many developments and improvements in these scientific fields. This competition has different types of challenges including a 3D Simulation League that has an annual tournament of simulated soccer games played between several teams each composed of 11 simulated humanoid robots. The simulation obeys the laws of physics in order to approximate the games as much as possible to real circumstances, in addition, the rules are similar to the original soccer rules with a few alterations and adaptations. The Portuguese team, FC Portugal 3D has been an assiduous participant in this league tournaments and was even victorious several times in the past years, nonetheless, to participate in this competition is necessary for teams to have their agents able to execute low-level skills such as walk, kick and get up. The good record of the FC Portugal 3D team comes from the fact that the methods used to train the robots keep being improved, resulting in better skills. As a manner of fact, it is considered that these low-level behaviors are at a point that is possible to shift the implementations' focus to high-level skills based on these fundamental low-level skills.

Soccer can be seen as a cooperative game where players from the same team have to work together to beat their opponents, consequently, this game is considered to be a good environment to develop, test, and apply cooperative multi-agent implementations. With this in mind, the objective of this dissertation is to construct a multi-agent setplay based on FC Portugal's low-level skills to be used in certain game situations where the main intent is to score a goal. Recently, many 3D League participants (including the Portuguese team) have been developing skills using Deep Learning methods and obtaining successful results in a reasonable time. The approach taken on this project was to use the Reinforcement Learning algorithm PPO to train all the environments that were created to develop the intended setplay, the results of the training are present in the second-to-last chapter of this document followed by suggestions for future implementations.

Acknowledgments

First, I would like to thank all the people involved in the FC Portugal team for the support given throughout this project, especially to Miguel Abreu and Tiago Silva for their patience and availability when I had many questions. I want to thank both my supervisor Luís Paulo Reis and co-supervisor Nuno Lau for the guidance and knowledge that they both offered.

Next, I need to go a few kilometers away from Porto, back to my home city. A huge thank you to everyone in my family, to my mother Ni and my father Nuno, thank you for all the love and support throughout the last five years, and for allowing me to live my best life. I also need to thank my sisters, Xana and Catarina, for always being there for me and for believing in me when I didn't. Remaining in Fafe, I would like to thank my friends, especially my school friends that I have known for many years now, some for almost 18 years. Thank you for staying in touch even after life sent us to different parts of the country, no matter where you are now, Fafe, Porto, Braga, Coimbra, or Lisboa know that I'm thankful for having all of you still with me and that I'm still questioning "Quem disse igreja ?". Last in my many thanks related to my home town, a huge thank you to a lovely lady and amazing teacher, Maria das Dores, a fundamental piece in my youth that made it possible for me to be pursuing an engineering degree.

It would not be possible to have an acknowledgments chapter without including the ones that walked this path side by side with me. My last thanks are for the 16 wonderful individuals that entered in Electro back in 2016 with me and that shared a similar lifestyle and dress code as me for the last years. Thank you for a friendship that is difficult to explain and even harder for others to understand, for the long nights of studying, having fun, barbecuing, or simply existing. Even though our year as " finalistas " was taken from us, I'm glad for the memories and stories that we get to share. I know that there are still many more adventures to come for all of us since " It's alright, cause we never say goodbye ".

Margarida Inês de Almeida Santiago

”The mind is left bereft when it is nothing more than a tool of regurgitation.”

Corey Taylor in *Seven Deadly Sins: Settling the Argument Between Born Bad and Damaged Good*

Contents

1 Introduction **1**
1.1 Context . . . 1
1.2 Motivation . . . 2
1.3 Objectives . . . 3
1.4 Thesis Structure . . . 3

2 Bibliographical Review **5**
2.1 Machine Learning . . . 5
2.2 Multi-Agent Robot Learning . . . 6
2.2.1 Multi-Agent Systems . . . 6
2.2.2 Cooperative Multi-Agent Learning . . . 6
2.3 Neural Networks and Deep Learning . . . 7
2.4 Reinforcement Learning . . . 8
2.4.1 Definitions and Terminology . . . 8
2.4.2 Deep Reinforcement Learning . . . 11
2.5 Deep Reinforcement Learning Algorithms . . . 11
2.5.1 Q-learning . . . 11
2.5.2 Policy Gradient Optimisation . . . 12
2.5.3 Combining Q-Learning and Policy Optimisation . . . 15
2.6 Multi-Agent Reinforcement Learning . . . 16
2.7 Evolution Strategies . . . 17
2.7.1 Covariance Matrix Adaptation Evolution Strategy . . . 17
2.8 Summary . . . 17

3 FC Portugal Skills and Tools **19**
3.1 Multi-Agent Architecture and Coordination . . . 19
3.2 Low-Level Skills . . . 21
3.2.1 Walk . . . 21
3.2.2 Kick . . . 22
3.2.3 Get Up . . . 23
3.3 High-Level Skills . . . 24
3.3.1 Coaching . . . 24
3.3.2 Player Types . . . 24
3.3.3 Dynamic Positioning and Role Exchange . . . 25
3.3.4 Situation Based Strategic Positioning . . . 25
3.3.5 Visual Debugging . . . 25
3.3.6 Intelligent Perception and Communication . . . 26
3.3.7 Setplays . . . 26

3.3.8 Reinforcement Learning Approaches 27
3.4 Support Tools and Technologies 28
3.4.1 SimSpark 28
3.4.2 RoboViz 31
3.4.3 NAO Robots 32
3.4.4 TensorFlow 33
3.4.5 Stable Baselines 33
3.4.6 FCP Gym 35
3.4.7 Full Learning Cycle 35
3.5 Conclusions 38

4 Experimental Setups **39**
4.1 Episode Layout 39
4.1.1 Description 39
4.1.2 Available Skills 40
4.1.3 Episode End Conditions 43
4.2 Action Space 43
4.2.1 Current Implementation 44
4.3 PPO2 Default Parameters 47
4.4 No Goalkeeper 48
4.4.1 Observation Space 48
4.4.2 Environment 1 48
4.4.3 Environment 2 52
4.4.4 Environment 3 54
4.5 With Goalkeeper 60
4.5.1 Observation Space 60
4.5.2 Keeper Description 60
4.5.3 Environment 1 61
4.5.4 Environment 2 63
4.5.5 Environment 3 67

5 Conclusion and Future Work **71**
5.1 Future Work 72

References **73**

List of Figures

2.1 Generic reinforcement Learning procedure [14]. . . . 9
2.2 RL algorithms, thicker lines represent different categories, the rest denotes the algorithms [16]. . . . 11
2.3 Actor-Critic architecture [20]. . . . 13
2.4 Comparison between A3C and A2C architectures [19]. . . . 14
2.5 Generic reinforcement learning procedure for MAS [27]. . . . 16

3.1 Simulated soccer field dimensions [63]. . . . 29
3.2 SimSpark's architecture during competition [65]. . . . 30
3.3 SimSpark's architecture during competition in more detail [66]. . . . 30
3.4 Setup alterations for training environment 32
3.5 Example of a still image of the RoboViz visualization [69]. . . . 32
3.6 Real NAO [72]. . . . 33
3.7 Simulated NAO [71]. . . . 33
3.8 Initial steps for creating n environments (*nenvs*) with two players for each training environment. . . . 36
3.9 Training episode cycle. . . . 37

4.1 Field division into quadrants [63]. . . . 40
4.2 Still image sequence of agent *R1* executing *passMediumKick(Vector kickTarget)*, top row represents the walk and the rest the preparation and execution of the kick. 41
4.3 *R2* moving to the chosen initial position. In the second row, from the first image to the second is possible to see the transition from skipping while waiting to chasing the ball. The next images shows the agent following the ball. Note that the pink circle outside of the penalty area is *R1*'s kicking target, this target is the central point of the circle that has a radius of 0.25 meters. . . . 42
4.4 *R2* kicks the ball without stopping and scores a goal. Note that the pink circle outside of the penalty area is *R1*'s kicking target, as for the circles inside of the goal, the pink circle (to the right of the second agent's front perspective) is the *R2*'s target with 0.15 meters of radius, and the red circle (to the left) its the previous target outputted by the network. . . . 42
4.5 In black possible targets taking into account the inputs of the *evaluateKickTargets19(...)* function and at yellow the best target that the strategy outputs. . . . 45
4.6 Alteration in the *evaluateKickTargets19(...)* function output. . . . 46
4.7 The pink circle and annotation represent the crossing agent's target, the left image shows a left corner and the adjacent a right corner. . . . 46
4.8 Available initial positions for the ball both agents are standing in their initial positions for the fourth ball initial position. . . . 49

4.9 Baseline implementation example. 50
4.10 Plot of the cumulative rewards of several models trained in Environment 1. Smoothing = 0.980. 51
4.11 Some details relative to the plots above. 51
4.12 Plot of the cumulative rewards of several models trained in Environment 2. Smoothing = 0.999. 53
4.13 Plot of the cumulative rewards of several models trained in Environment 3. Smoothing = 0.991. 56
4.14 Plot of the cumulative rewards of several models trained in Environment 3 with random ball initial position. Smoothing = 0.994. 57
4.15 Plot of the cumulative reward of a model trained in Environment 3 only calling the NN once. 58
4.16 Example of two models trained in Environment 3 with timestamps. 59
4.17 Opponent's goalkeeper, normal stand in the left and a defense example on the right. 61
4.18 Plot of the cumulative rewards of several models trained in Environment 1 with a goalkeeper. Smoothing = 0.995. 62
4.19 Plot of the cumulative rewards of several models trained in Environment 2 with a goalkeeper. Smoothing = 0.995. 64
4.20 Plot of the cumulative rewards of several models trained in Environment 2 with a goalkeeper and a custom policy network. Smoothing = 0.995. 65
4.21 Plot of the cumulative rewards of several models trained in Environment 2 with a goalkeeper and *lam* = 0.99. Smoothing = 0.997. 66
4.22 Baseline implementation example with a goalkeeper, the pink values above the pink targets are the ball's initial random coordinates and the$R2$'s target. 67
4.23 Plot of the cumulative rewards of several models trained in Environment 3. Smoothing = 0.990. 68

List of Tables

3.1 Stable Baselines RL algorithms and their characteristics. 34

4.1 Action space description with the limits of each action. 44
4.2 PPO2 default hyperparameters from the Stable Baselines. 47
4.3 Observation space for the environments trained without a goalkeeper. 48
4.4 Performance results from the baseline implementation. 50
4.5 Performance results from some of the models trained in Environment 1. 52
4.6 Performance results from some of the models trained in Environment 2. 54
4.7 Performance results from some of the models trained in Environment 3. 57
4.8 Performance results from some of the models trained in Environment 3 with random initial ball position, plus re-evaluation of the best current model. 58
4.9 Performance results from a model trained in Environment 3 using the NN once. . 59
4.10 Observation space for the environments trained with a goalkeeper. 60
4.11 Performance results from some of the models trained in Environment 1 with a goalkeeper. 62
4.12 Performance results from some of the models trained in Environment 2 with a goalkeeper. 64
4.13 Performance results from some of the models trained in Environment 2 with a goalkeeper and a custom policy network. 65
4.14 Performance results from some of the models trained in Environment 2 with a goalkeeper and *lam* = 0.99. 66
4.15 Performance results from the baseline implementation with a goalkeeper and random initial position for the ball. 67
4.16 Performance results from some of the models trained in Environment 3 with a goalkeeper. 68

Abbreviations and Symbols

γ	Discount Factor
μ(St)	Deterministic Policy
π(. | St)	Stochastic Policy
A, a	Action
A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor-Critic
AC	Actor-Critic
ACER	Actor Critic with Experience Replay
ACKTR	Actor-Critic using Kronecker-Factored Trust Region"
AI	Artificial Intelligence
ANN	Artificial Neural Network
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
COMA	Counterfactual Multi-Agent
CoM	Center of Mass
CPU	Central Processing Unit
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Q-Learning
DPRE	Dynamic Positioning and Role Exchange
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
FCP	FC Portugal
ES	Evolution Strategies
ER	Experience Replay
FEUP	Faculdade de Engenharia da Universidade do Porto
GAIL	Generative Adversarial Imitation Learning
GDB	GNU Project Debugger
GPU	Graphics Processing Unit
HER	Hindsight Experience Replay
IRL	Inverse Reinforcement Learning

LIACC	Laboratório de Inteligência Artificial e Ciência de Computadores
M3DDPG	MiniMax Multi-Agent Deep Deterministic Policy Gradient
m	meters
m/s	meters per second
ms	milliseconds
MARL	Multi-Agent Reinforcement Learning
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
MDP	Markov Decision Processes
MAS	Multi-Agent System
ML	Machine Learning
NN	Neural Network
PPO	Proximal Policy Optimization
R	Reward
RoboCup	Robot Soccer World Cup
RL	Reinforcement Learning
SB	Stable Baselines
S,s	State
SBSP	Situation Based Strategic Positioning
SAC	Soft Actor-Critic
TCP	Transmission Control Protocol
TD3	Twin Delayed DDPG
TRPO	Trust Region Policy Optimization
unum	Uniform Number
V	Value
$V(s)$	Value Function
VPG	Vanilla Policy Gradient
ZMP	Zero Momentum Point
ZMP-GPC	Zero Momentum Point - Central Pattern Generator

Chapter 1

Introduction

The promising development and progress in robotics and artificial intelligence (AI) lead to a growing interest by the scientific community in these two fields. These areas of research have been highly promoted by initiatives like the RoboCup, an annual event where several types of competitions and leagues occur. One of these leagues is the 3D Simulation League where teams, made by humanoid robots, interact with each other and the world in a simulated soccer game, this environment can be defined as a multi-agent system (MAS) that promotes the development of coordination, cooperation, and learning within each team. To achieve a likewise, real soccer team performance, several algorithms are used to teach the "players" all sort of behaviors starting with trivial ones as walking and kicking up to more high-level skills such as cooperation. In recent years, reinforcement learning algorithms became a subject of intense study having gain praised recognition after the outstanding performance of DeepMind's AlphaGo Zero which consist of a software capable of playing the game of Go[1]. This program remains unbeatable since it was able to surpass its previous versions (that have already beaten the Go's World champion) by only playing against itself and using a deep reinforcement learning (DRL) algorithm to master the game, whereas its old implementations were trained using data from previous human games combined with reinforcement learning.

In light of the description above, the pivot of this work will be to use deep reinforcement learning to train multiple humanoid robots in a multi-agent learning environment defined by the rules and restrictions of the simulated league in hopes of developing a advantageous cooperation between these.

1.1 Context

This dissertation emerges from a joint project between the University of Porto and the University of Aveiro with the intent on the participation of the FC Portugal 3D team in the Robot Soccer World Cup or simply, RoboCup. RoboCup is a global robotics competition that occurs annually, it

[1]Go game explained here.

started as an event-based around simulated soccer but has evolved into various types of competitions that include: RoboCupSoccer, RoboCupRescue, RoboCup@Home, RoboCupIndustrial, and RoboCupJunior. As the names suggest, each competition has its own "theme" and goals, nevertheless, focusing on the RoboCupSoccer, this competition is composed of several different leagues including a Simulation League with a 3D sub-league where matches are played between two teams of 11 humanoid robots each. Players resemble a simulated version of SoftBank Robotics's NAO robots and the games are played in a 3D Soccer Simulation environment as well [1]. The robots must play completely autonomously and as a team by interacting with each other and with the environment with the main objective being to score goals and not suffer them. This system can be classified as a multi-agent system (MAS), which, in a general view, can be defined as a group of intelligent agents that make decisions and take actions either in a cooperative or competitive matter to achieve a personal or global goal.

1.2 Motivation

The motivation behind RoboCup is to promote robotics and AI enthusiasm and research by exposing, in an appealing way, a challenge to those who are interested in participating [2]. With the various advances in robotics and artificial intelligence that have happened and that are still to come, the ultimate goal of the RoboCup initiative would be:

> “By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup [2].”

To promote the evolution necessary to achieve this objective, the rules for the competition are updated frequently so that a higher level of performance is demanded leading to an improvement in the methods and algorithms used and, consequently, the agents will converge into more realistic behaviors. The base of an agent's behaviors is considered to be low-level skills, which are a set of singular actions necessary to play a game of soccer, such as walking, running, kicking, and getting up. Currently, these skills are at a satisfying point of efficiency which enables the opportunity to explore more complex implementations, besides, resorting to simulation leagues (2D and 3D) enables the possibility of abstraction from hardware-related problems exiting in real robots. A major improvement happened in 2013 where teams, from that point on, were able to use heterogeneous robot types, a variation of the standard NAO robot where it is possible to have agents better adapt to a certain player type, for example having a good kicker for agent intended for scoring goals and a different type of robot for the goalkeeper. With these advances, in the current time, the 3D Simulation league can start to shift its focus into the development of high-level skills where agents use artificial intelligence, setplays, and strategies to outperform their opponents [1].

1.3 Objectives

The simulated robots games obey realistic physics, behave autonomously, and use low-level behaviors to engage high-level decisions. As mentioned before, given that the former keeps being improved and is, presently, at a suitable level of efficiency, implementing high-level skills, supported by robot's cooperation and state of the art low-level skills, is where teams should focus to surpass the rest of the competitors. The main goal of this dissertation is to design, implement, and test a multi-robot environment where the agents' base behaviors come from previous implemented low-level skills, and by using a deep reinforcement learning algorithm, these robots should learn the best way to follow a setplay designed with a pass and kick, in which the final result is scoring a goal. This setplay is meant to be used in specific game situations such as corners kicks and throw-ins, the latter is also referred to as kick-ins in the context of this dissertation.

1.4 Thesis Structure

This dissertation is divided into 5 chapters, the outline of this document is as follows:

- **Chapter 1: Introduction** This chapter exposed the dissertation's theme, with contextualization, motivation, and goals to be achieved.
- **Chapter 2: Bibliographical Review** Next, an introduction to the scientific field of this thesis's theme is performed, beginning with a brief introduction to machine learning, followed by the main subjects. Here a description of multi-agent learning is presented as well as a reinforcement learning deconstruction including some algorithms. This culminates into a depiction of the multi-agent reinforcement learning field. As a means of comparison, this chapter closes with a short resume of evolution strategies.
- **Chapter 3: FC Portugal Environment** Chapter 3 presents relevant information about the FC Portugal team. First, a summary regarding the multi-agent architecture and coordination in previous implementations followed by a description of the low-level skills available to use and some context about their development. Regarding high-level, former implementations in the team relevant to the field of coordination and cooperation are included. A short overview concerning fundamental tools used for the team's development and agent training is also made.
- **Chapter 4: Experimental Setups** This chapter includes some of the experiments made on the course of this dissertation with episode's structure, reward attributions, and results.
- **Chapter 5: Conclusion and Future Work** To conclude this dissertation, the last chapter includes a brief resume of all document's chapters as well as an evaluation of the results obtained and future improvements to be considered.

Chapter 2

Bibliographical Review

The following chapter aims to present a summary of the study made relative to the various fields and sub-fields of AI on which this dissertation focuses. At the beginning of this project, this field of science was relatively new and unknown, resulting in an extensive chapter that includes several fundamental topics in machine learning and later in reinforcement learning. Concerning multi-agent systems, there is an emphasis on cooperative agents since that is the environment within a soccer team. This chapter terminates with an alternative to machine learning algorithms, the evolution algorithms, and, in detail, the Covariance Matrix Adaptation Evolution Strategy used before to train robot agents as it will be mentioned in the succeeding chapter section 3.2.

2.1 Machine Learning

Machine learning is a branch of artificial intelligence that allows systems to learn from example data or former experience. Its methods are used to solve problems related to vision, speech recognition, and robotics [3]. There are different types of machine learning systems that can be implemented depending on the nature of the problem, these can be classified by the amount of human supervision given during training [4]. The major categories in machine learning are as follows:

- **Supervised Learning** is based on an established set of input data and an understanding of how to classify it by giving the algorithm the desired solutions referred to as "labels".
- **Unsupervised Learning**, similar to what is stated in the item above, in unsupervised learning the algorithms also receive input data, however, the desired solution is not provided [3].
- **Semi-Supervised Learning** combines supervised and unsupervised learning resulting in a division of inputs as labeled and unlabeled [5].
- **Reinforcement Learning** differs from the rest since the training does not require a sampled data set. Rather the system learns through trial and error, and when a sequence of successful decisions occur, the process is "reinforced" into that positive behavior.

2.2 Multi-Agent Robot Learning

Multi-agent systems are used to solve several types of problems in many domains such as robotics, distributed control, telecommunications, and economics. These fields present degrees of complexity too demanding for their tasks to be executed with pre-programmed behaviors. Therefore, the agents must discover the solutions on their own by appealing certain methods in hopes of learning to overcome the challenges at hand.

2.2.1 Multi-Agent Systems

Wooldridge, M. in “An Introduction to Multiagent Systems”[6], defines Multi-Agent System (MAS) as systems composed of several autonomous elements called agents. These are capable of interacting with the environment and each other by exchanging data as well as having "social activity", similar to what humans experience in daily interactions, such as cooperation, coordination, or negotiation. Simulated soccer robotics consists of a group of connected autonomous agents, a team, that act in an environment and through their interaction try to achieve a common objective, win the game. From this description, it is clear that simulated soccer embodies the characteristics that define a multi-agent system. As mentioned, in these systems it is possible to have agents either cooperating or competing. Bearing in mind that this thesis seeks to improve a team’s behavior, a cooperative MAS is the focus of this analysis.

Although multi-agent systems are of great complexity, they motivate distributed solutions that are more efficient than centralized single-agent systems in which a lone agent makes all the decisions. Nevertheless, the multi-agent systems present some critical challenges, thus, the use of ML methods to overcome these difficulties has been a recurring theme. In soccer robotics, the use of reward-based learning is the most appropriated given the highly dynamic and uncertain environment provided in the RoboCup games. These methods can be divided into [7]: reinforcement learning methods based on the estimation of value functions/policies or stochastic search methods the behaviors are learned directly without estimations. Stochastic hill-climbing or evolutionary computation, or simulated annealing are some examples of these methods.

2.2.2 Cooperative Multi-Agent Learning

Cooperative multi-agent learning approaches can be split into: team learning and concurrent learning [7]. The former is based on a unique learner responsible for the behaviors of all the participants in the team, however, with the increase in the number of agents, scalability becomes a problem, i.e, when multiple agents are present, the dimension of the search space will quickly increase with the amount and complexity of agents and agents behaviors. Scalability is considered to be one of the major problems when dealing with MAS. The alternative to team learning, concurrent learning, consists of each agent having its own training process, rather than learning behaviors for the entire team, thus reducing the state-action space. Nonetheless, since the environment is not static, agents must learn to adapt to a mutating world, as the other agents are also learning and changing their policies, this becomes the most considerable complication when using this method.

Within the concept of team learning, there are different categories to classify a team. Homogeneous team learning consists on the development of a single-agent behavior that every player on the team uses, as a consequence, it is not possible to have agents specialization since all players will act the same way. However, in heterogeneous teams, agents can develop unique behaviors specific to certain players, creating a stronger and more capable formation. There is also the possibility to have a compromise between these two concepts: a hybrid team can be constructed by dividing the team into squads where agents belonging to the same squad share the same behavior (homogeneous learning), although the assigned behaviors differ from one squad to another, having a heterogeneous team. Both heterogeneous and hybrid teams have been researched for the RoboCup competitions [7].

Regarding concurrent learning, usually, each agent has its particular learning process, however, it is also possible to split the team into squads similar to what was mentioned in the last paragraph, having a learned per squad. When working with multiple learners, a major inquiry stands: which agent should the reward be attributed to? The most elementary answer is to split the team's reward uniformly between all players. The downfall of this global reward is that it leads to lazy behaviors being rewarded. An alternative would be to only award the learners that actively participated in the demanded task and punish the others. Several approaches by various authors to solve the reward attribution paradigm are explained in [7].

In cooperative multi-agent systems, agents can communicate with each other to refine coordination, provide precise models of the environment, and learn subtask solutions from one another. However, communication can expand the size of the search space of the learning method by adding, to each agent, foreign states belonging to the other agents, as well as by increasing the agents' range of options. Thus, even when communication is needed for optimal performance, numerous times it is disregarded to simplify the learning process [7]. Nonetheless, there are two types of communications possibilities: direct communication and/or indirect communication. In the latter, the agents leave marks in the environment to pass information. In terms of direct communication, this method employs radio signals or light waves to send messages.

2.3 Neural Networks and Deep Learning

Neural networks (also known as connections models or parallel distributed processing) were first introduced in 1943 as a simplified model of biological neurons. These artificial neural networks (ANN) possess properties that allow them to identify patterns and make associations, resulting in a high capacity to adapt, generalize, cluster, and organize data [8].

The architecture of each network is based on neurons connected by weighted synapses. The use of a three-layer network is commonly applied in trivial problems and consist of: an input layer, one hidden layer, finishing with an output layer. In more complex ANN's there can be several hidden layers and not just one, likewise, each layer can have different amounts of neurons.

Input neurons have no predecessor neurons, it is from here that the NN is fed with data [9]. The hidden layer, located between the input and output, maps internal information patterns between the input and output layers. After the output layer collects the information, a solution is presented, having no successor neuron.

Considering the connection between layers, the output of a neuron can be the input of another, and the strength and meaning of these connections between them characterize each link weight. The stronger the weighting, the greater the influence a neuron can exert on the connection to another neuron [10]. To adjust and modify the weights of these connections, to favor a certain output for a given input, a function called the learning rule is used.

Finally, to compute the input of a neuron from the outputs of its previous neurons, it is necessary to use a propagation function [9], the input values are added up and passed to an activation function, which generates an output [11].

Deep learning is a sub-field of machine learning based on artificial neural networks, as for **Deep neural networks**, these are neural networks with plentiful hidden layers, which can be used to extract features from the inputs and to compute complex functions [12].

2.4 Reinforcement Learning

The evolution of computational resources and the constant emergence of new reinforcement learning algorithms led to a growing interest in using automated learning applied to robotic tasks. In summary, reinforcement learning consists of having an agent (or agents) learn by interacting with the environment. It uses the formal framework of Markov Decision Processes[1] to define these interactions in terms of states, actions, and rewards, allowing a plain representation of essential features in the AI problems [13]. While learning, at each step, the agent receives information from the environment relative to its current **state**. Based on an algorithm, the agent must assess and decide an **action** to take. After execution, the agent receives a **reward** representing the feedback related to the quality of its action. This reward can be positively promoting the agent's decision or, "punishing" it for the action taken. The goal of the agent is to maximize its cumulative reward, called **return value**. A generic scheme of the RL procedure for a single-agent learning can be found in figure 2.1, where *t* represent present events and *t+1* future ones.

2.4.1 Definitions and Terminology

The overall terminology that defines the reinforcement learning methods is as follows [2]:

- **Agent** the entity that is learning and who takes actions.
- **State (*S*)** gives a description of the state of the world that the agents as access to.
- **Action Space (*A*)** is the set of all valid actions that an agent can take.

[1]The MDP framwork is defined by $(S,A,\mathrm{P},R,\gamma)$ detailed at section 2.4.1.
[2]Key concepts in RL based on OpenAI.

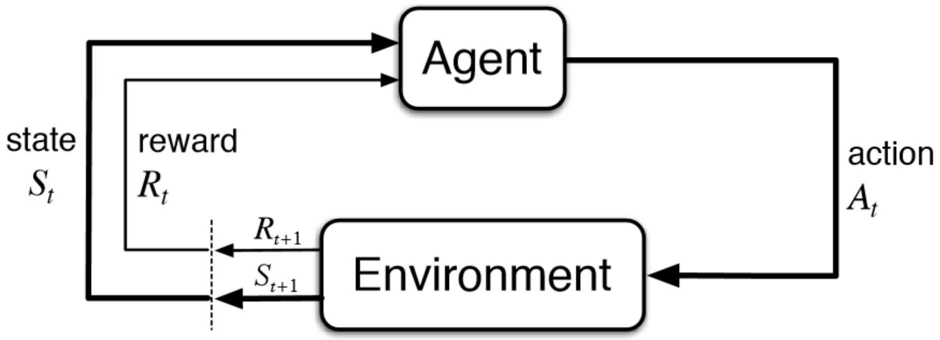


Figure 2.1: Generic reinforcement Learning procedure [14].

- **Observation** is a partial description of the state.
- **Episodes or Trajectories** define a sequence of states and actions.
- **Reward (*R*)** is the feedback provided to evaluate the last action.
- **Discount factor** (γ) determinants how much the agent values rewards in the distant future relative to the achievable ones in a "closer" future. Besides, this factor is useful to force the infinite-horizon sum of rewards to converge to a finite value.
- **Value (*V*)** contrary to the reward, this value represents the expected long-term return with discount.
- **Policy** (π / μ) is the strategy that the agent employs to chose what action to take based on the current state. It can be deterministic, in which case it is denoted by μ or stochastic denoted by π.
- **Exploration and Exploitation** in RL, the agent must find a proper balance in executing random actions, even if it has already found a good set of actions that result in a satisfactory reward. With exploration, the agent can find an even better solution than the best one found at the moment, or not at all. Thus, at some point, it should exploit the actions that favor the final reward.

2.4.1.1 Discrete and Continuous Actions and States Spaces

The world of reinforcement learning problems can be characterized either by discrete or continuous actions and states that impact the algorithm used. Earlier ML algorithms only had discrete features adequate for scenarios such as teaching an agent to play tic-tac-toe. In this game, the player has a maximum of nine concrete play options in a 3x3 environment that are either free or occupied (either the player or the opponent have already crossed a position). However, this implementation is far from representing real-world scenarios that mostly belong in the continuous domain such as the ones in robotics. Even if only a part of a robot is being controlled, for example, a robot arm, this agent has some degrees of freedom defined by how many joints it has. Each joint can be moved to a position with a certain velocity, with the only constraints being the maximum and minim limit values of each joint.

2.4.1.2 Deterministic and Stochastic Policies

Deterministic policies map states to actions defined by $A_t = \mu(S_t)$, where the policy will always return the same action for a given state of the observation space. In case of the **stochastic policies**, defined as $A_t \sim \pi(. \mid S_t)$, these are often represented as conditional probability distributions, where a function assigns a probability for each action to be taken given a certain state.

2.4.1.3 Off and On policies

For **off-policy** algorithms, the policy is independent of the agents' actions, resulting in getting the optimal policy regardless of the agent's motivation. In other terms, these algorithms evaluate and improve one policy based on greed to enable the agent to learn the optimal one, having a different policy to determinate the agent's behavior. Opposite to this in **on-policy** algorithms, the policy that is used and improved is the same policy that the agent is already using for action selection with determinants of its behavior [15].

2.4.1.4 Model-Free and Model-Based

In deep learning, a model is represented by a function that can predict state transitions and rewards, these models are used for planning by taking into account future situations, that have not yet happen, they define the course of action. When an agent's algorithm uses models and planning then it is called a **model-based** method, which opposes the **model-free** method, based on learning from the environment through trial-and-error [13]. The difference between these models is if, whether or not, the agent has the model beforehand or if it learns the model (or dynamics) of the environment [16]. In the context of this dissertation, the highlights are the **model-free** methods, given that learning through a model in a highly dynamic soccer match would lead to an unrealistic depiction of the environment given the constant alterations caused by elements unrelated to the agent like its teammates or opponents' actions.

2.4.1.5 Policy-Based and Value-Based

The classification of policy optimization methods can be defined as a: **value-based** optimization or **policy-based** optimization or even actor-critic. For **value-based** methods, the goal is to maximize a value function $V(s)$ to reach an optimal value. In addiction, these must oscillate between estimating the value function based on the current policy and improving the policy with an estimation of the value function. In the case of **policy-based** optimizations, these directly update the policy without estimating or learning an action-value function, the main focus is to find the

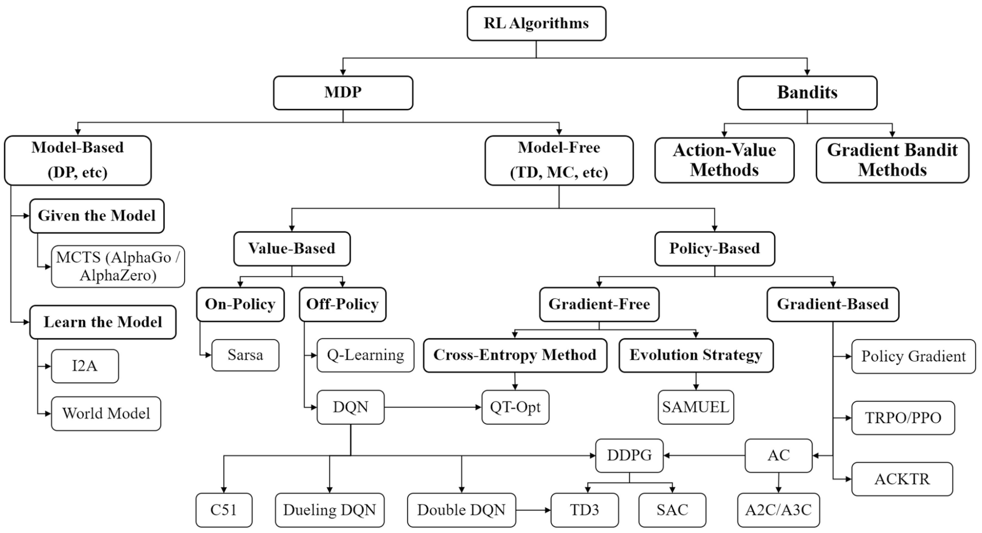


Figure 2.2: RL algorithms, thicker lines represent different categories, the rest denotes the algorithms [16].

optimal policy [16]. In terms of convergence the latter is preferable in situations with continuous high dimensional data, and are more effective when dealing with deterministic policies [17].

2.4.2 Deep Reinforcement Learning

DRL is a combination of reinforcement learning and deep learning as the name suggests. On account of the sizable action and state spaces present in robotic environments, it can be challenging for policy optimization. Accordingly, incorporating NN in the models along with adaptive reinforcement learning techniques leads to better performances in these systems. The use of neural networks is extremely effective to approximate optimal value functions and has been widely used to predict the most favorable control policy [17].

2.5 Deep Reinforcement Learning Algorithms

Figure 2.2 presents the taxonomy of reinforcement learning algorithms divided by model-based or model-free as well as value-based methods, policy-based methods, or the combination of the two methods, respectively. This scheme was obtained from chapter three of “Deep reinforcement learning: Fundamentals, research, and applications” [16]. As mentioned in section 2.4.1.4, the focal point of this work are the model-free algorithms.

2.5.1 Q-learning

Q-learning algorithms belong in the **off-policy** category and are also referred to as **value-based** methods. The ’Q’ stands for quality and it aids in the search for the next action that will grant an agent a state with higher quality. In short, these algorithms estimate the Q-value of each state-action pair (s,a) in a greedy logic where the goal it to maximize 'Q' at each step [16]. A particular feature of these algorithms is the use of a Q-table. This table shape shows the relation between the possible states and actions and the quality value of their paring. Usually, this is initialized with arbitrary values and suffers updates as the agent goes through the episodes. Using a Q-table requires a high demand of memory and computational resources, however, the results with these methods are more sample efficient compared to other on-policy alternatives.

2.5.1.1 Deep Q-Learning

The article "Deep Reinforcement Learning: A State-of-the-Art Walkthrough" explains that **Deep Q-Learning** "has the purpose of approximating the optimal action-value function Q$*$(s,a) through the use of deep neural networks" [18]. Hence, the foundation for all Deep Q-Learning methods is the use of **Deep Q-Network** (**DQN**), due to the fact that for more complex environments with vast possibilities for both states and actions, it is unreasonable to use a Q-table. Another disadvantage in elemental Q-learning is the lack of stability, when the approximation of Q-values uses non-linear functions. To overcome this issue, **DQN** has a clone of the original Q-Network (referred as a target Q-Network) that is updated intermittently to match the online network that is constantly up to date. Furthermore, the use of Experience Replay (ER) improves the algorithm's convergence into a solution, this technique uses mini-batches containing experiences that can later be sampled in the training. This technique was implemented primarily to combat the possible divergence of the policy caused by the use of NNs to approximate the Q-values [18].

2.5.1.2 Double DQN and Dueling DQN

Improvements to the DQN method came in the form of the **Double Q-Learning** algorithm (**DDQN**). This implementation improves performance by being able to remove the overestimation bias introduced in some cases given that the same network would both evaluate and select the agent's actions [18].

Another extension of DQN is the **Dueling DQN** this algorithm proposes that the Q-value is split into: state value, the value of being in that same state, and action advantage, which represents how beneficial is it to take a specific action A in state S compared to all the other available actions for that state [16].

2.5.2 Policy Gradient Optimisation

The policy-based algorithms can be classified as either gradient-based or gradient-free methods[3], a summary explaining most of the policy-gradient concepts relations and interactions, as well as the several algorithms is available at [19]. The theoretical foundation for several policy gradient

[3]The optimization methods' objective is to find the finest parameters that optimize a given function. When a method uses a gradient to get these parameters it is referred to as a gradient-based method, otherwise is considered to be a gradient-free optimization.

algorithms is based on the Policy Gradient Theorem and their purpose is to optimize the policy directly. As opposed to DQN methods, the policy-based methods support continuous action and state spaces [18], as mentioned before in section 2.4.1.5.

2.5.2.1 REINFORCE

Referred to as **REINFORCE** (also known as Monte Carlo policy gradients) or **Vanilla Policy Gradient** (**VPG**) [18] is an on-policy algorithm with a stochastic policy meaning that: exploration occurs via sampling actions based on the latest version of the policy. In this algorithm, the actions only get feedback of their impact on the world (reward) when the episode ends, as oppose to DQN, where the reward is immediately obtained after an actions is executed. As such, the **REINFORCE** algorithm praises every action (even if some were bad) if the experience is considered good, which is not desired.

2.5.2.2 Actor-Critic

Actor-Critic (**AC**) methods improve the policy update by reducing the gradient variance[4]. These algorithms are defined as temporal difference (TD) learning methods. The actor is the policy structure that proposes the action for a given state and the critic represents the estimated value function that learns the policy and evaluates the new state, i.e, the result of the actions taken by the actor chosen based on the given policy.

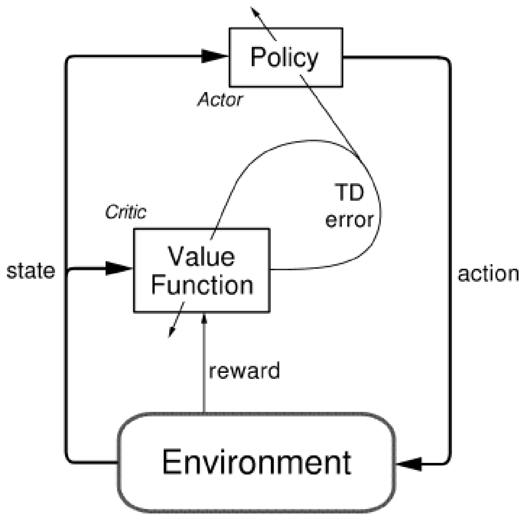


Figure 2.3: Actor-Critic architecture [20].

Figure 2.3 displays the architecture of the **actor-critic**. The TD methods can be used to estimate value functions, the *TD error* present in the figure is the error of an estimate for the following step, meaning that given a step *t*, the error in this step depends on the following reward and state,

[4]Policy Gradient methods can have issues given their high variance and low convergence [18].

however, these are only available at the *t+1* step, thus an error might exist when comparing the estimate with the value of the next step.

2.5.2.3 Asynchronous Advantage Actor Critic

A3C the **Asynchronous Advantage Actor Critic** main characteristic is the focus on parallel training, while the critic learns the value function, multiple actors are trained in parallel environments and independently update a global value function. Due to this asynchronous architecture, this method was revealed to be considerably faster than DQNs [21].

2.5.2.4 Advantage Actor Critic

Advantage Actor Critic (**A2C**) is the synchronous deterministic version of A3C, where trajectories are created by a coordinator who waits for all the parallel actors to finish their work before updating the global network. After this, the next iteration can begin, where each agent has access to the same version of the network [18]. This implementation arose considering that A3C has agents using outdated parameters since the network update is asynchronous.

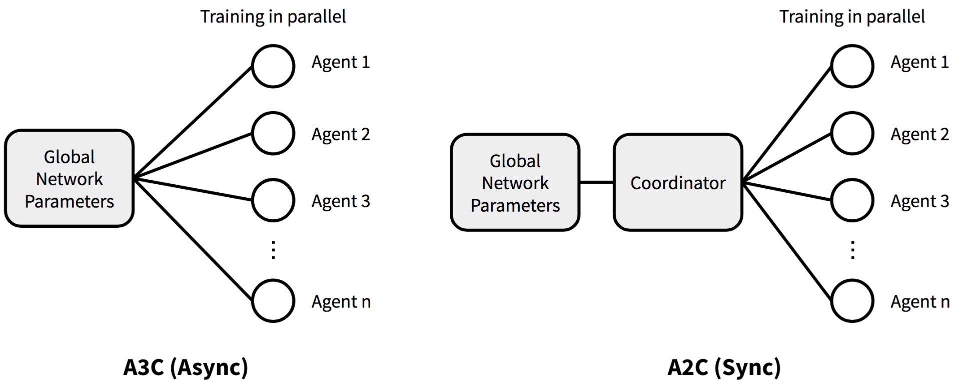


Figure 2.4: Comparison between A3C and A2C architectures [19].

2.5.2.5 Trust Region Policy Optimization

The **Trust Region Policy Optimization** (**TRPO**) improves stability during training by avoiding parameter updates that alter the policy in excess in one step [19]. The policies updates occur by taking the largest step possible to improve performance while satisfying the limitation, inherited by the use of trust region[5], of how close the current and older policies are allowed to be [23]. Nonetheless, **TRPO** is somewhat complicated and suffers from excessive computation needed for gradient approximation [18].

[5]The major optimization methods are considered to be the line search (gradient descent) and trust region. In the latter, first, the maximum step size is calculated and then exploration happens to locate the optimal point within said trust region [22]. Gradient descent is an optimization algorithm to find the local minimum of a function by consecutively moving in the direction of the fastest descent as determined by the negative of the gradient, a more detailed explanation can be found at the online Machine Learning Glossary.

2.5.2.6 Proximal Policy Optimization

Even with the complications of TRPO, a similar constraint could be desired when choosing a RL algorithm, hence the development of **PPO**. Unlike TRPO, **PPO** can simultaneously be less complex, reduce computational expense, and can maintain satisfying policy update sizes by incorporating first-order methods, opposed to the use of second-order methods in the Trust Region algorithm [18]. TRPO has a major problem with oversize the step size, however, in **PPO** the algorithm is able to maximize the gradient step size without letting it become too massive. At every step, the update that leads to a minimal cost function is calculated, while doing so, it guarantees that the newer policy does not deviate too much from the previous one [24]. This method is significantly simpler to implement and its performances showed results at least as good as TRPO's. The OpenAI Baselines (set of implementations of reinforcement learning algorithms) released a GPU-enabled implementation of **PPO**, called **PPO2** that runs approximately three times faster in comparison to **PPO** [24].

2.5.3 Combining Q-Learning and Policy Optimisation

It is possible to combine elements of both policy-based and value-based methods available in the model-free family to create algorithms that inherit the best features from each branch.

2.5.3.1 Deep Deterministic Policy Gradient

Policy gradients have a stochastic nature, however, there is a type of algorithms, based on these methods, called Deterministic Policy Gradient (DPG) where the actions are selected in a deterministic way. The **Deep Deterministic Policy Gradient** (**DDPG**) algorithm is the most primitive of DPG's. This algorithm is an off-policy actor-critic implementation that combines DPG with DQN (value-based), resulting in better performances when compared to the typical DQN model. Nonetheless, this algorithm has a significant problem inherited by the DQN regarding the overestimated Q-values (contrary to the actor-critic methods, A3C and A2C, **DDPG** estimates the Q-values instead of optimizing the policy), besides it also requires a great amount of tuning to reach a satisfactory setup [18].

2.5.3.2 Twin Delayed DDPG

To address the issues stated above, the implementation of **Twin Delayed DDPG** or simply **TD3**, emerged with a series of alterations to its predecessor. To begin with, it uses two different Q-functions for reducing the overestimation bias. The update of the policy (and the target network) occurs with less frequency than the Q-value function. Lastly, it introduces noise to the target action to smooth the Q-value function and avoid overfitting [16].

2.5.3.3 Soft Actor-Critic

Another improvement to DDPG is the **Soft Actor-Critic** algorithm (**SAC**), a compromise between Q-learning and DPG methods. This algorithm introduces a maximum entropy term, where the entropy regularized reward is being maximized by the optimal policy instead of the discounted cumulative reward which encourages the exploration of the policy [16]. The implantation of the maximum entropy framework is based on an off-policy methodology combined with a stochastic actor-critic. In terms of sample efficiency and performance, in high-dimensional continuous control tasks, this method can outperform the state-of-the-art on/off-policy algorithms [18].

2.6 Multi-Agent Reinforcement Learning

Figure 2.5 shows the application of RL in a multi-agent environment having evident similarities with the generic single-agent RL. In fact, the framework when using **MARL** (**Multi-Agent Reinforcement Learning**) can resemble the single-agent given that multi-agent environments can be trained by having each robot learn individually. Each agent sees the others as part of the environment and each one must learn their own policy to maximize their payoffs [25]. In terms of goals, the **multi-agent reinforcement learning** problems for cooperative agents can be approached with the intent to either achieve individual goals, accomplish a global team objective, or even, in the state-of-the-art **MARL**, maximize both rewards. However, the latter scenario is still considered to be an open challenge. Earlier works in this scientific field were limited to discrete states and actions, more recently, approaches based on techniques from single-agent deep reinforcement learning have been developed with successful results creating algorithms for MAS environments with high-dimensional continuous actions and states [26].

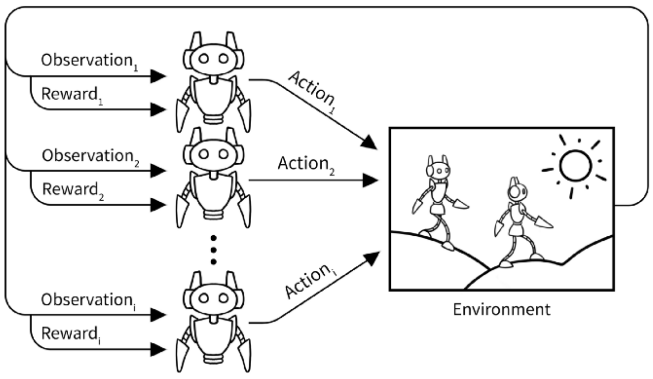


Figure 2.5: Generic reinforcement learning procedure for MAS [27].

Currently, there are some algorithms in **MARL** specifically to maximize the team's reward such as **COMA** and **QMIX** where the group performance overlaps the individual. Other algorithms such as **MADDPG** and **M3DDPG** focus on optimizing local rewards, although there are various algorithms for both approaches, the "Survey of Recent Multi-Agent Reinforcement

Learning Algorithms Utilizing Centralized Training" presents several **MARL** methods as well as a summary of this reinforcement learning sub-field.

2.7 Evolution Strategies

Evolution Strategies (ES) are direct search and optimization methods that belong in the family of evolutionary algorithms. These algorithms are based on the division of population inspired by natural selection, where it is believed that individuals with traits that allow them to survive and prosper will thrive and pass the beneficial characteristics to the next generation. Evolution happens by a selection process where the "fitness" of the population samples is evaluated, with the intent that future generations only have the most adept and better the equipped individuals [28].

2.7.1 Covariance Matrix Adaptation Evolution Strategy

The **CMA-ES**, used in continuous domain challenges, is an evolutionary algorithm able to achieve finer and faster results when compared to other stochastic optimization techniques. This method has some beneficial characteristics, it has the capacity for autonomous adaptation of the step size, has efficient covariance updates that incorporate the current samples in addition to the evolution path and its invariance properties. Even though **Covariance Matrix Adaptation Evolution Strategy** is not a reinforcement learning algorithm, it is still able to find the minimum value for a fitness function, making it useful when learning which parameters result in the best fitness function [29, 30].

2.8 Summary

Upon concluding the analysis of this chapter, it is clear that there was a greater emphasis on single-agent algorithms compared to MARL. This is explained given the high complexity of using the specific multi-agent algorithms and the time constrain to complete this dissertation. In 2020 a student at FEUP presented a thesis approach using MARL with decentralize learning, i.e, Ventura in "FCPortugal Multi-Robot Action Learning" [31] trained a multi-agent environment by having each agent learn individually. However, this project will focus on having **two agents learning at the same time by sharing a neural network that they both influence and alter, resulting in a shared policy**, and so, this approach will focus on **using a single-agent algorithm**.

Chapter 3

FC Portugal Skills and Tools

The FC Portugal team has shown their potential and quality through the years, namely being World champions in 2000, consecutive European champions in 2000 and 2001 in the 2D league (created before the appearance of the three dimensional league). Exclusive to the 3D league, in 2006, the team was World Champion and at various times achieved the status of European champion (2007, 2012, 2013, 2014, and 2015) among several other noteworthy participations and awards. The success of this team comes from the continuous work put into improving the learning framework by applying new techniques and algorithms to the agents' learning process. Concerning the low-level skills, the focus is on developing behaviors that, with minimal adaptation, could be applied to real robots. As for the high-level, the starting point was the adaptation of previously developed methodologies for the 2D team into the 3D [32]. Some implementations were able to be transported to the higher dimension team. Nevertheless, giving the added complexity of using a humanoid version for players and the evolution and adaptation of the 3D league rules, new challenges arose, leading to new objectives and leaving some tools previously used no longer of service in the present time.

3.1 Multi-Agent Architecture and Coordination

An introduction to cooperative multi-agent made in section 2.2 reveals that to enable a team to perform cooperative multi-agent tasks involves many factors, fundamentally, the recurrent update of the agents' world state in a somewhat accurate manner in a complex dynamic environment. To do so, the FC Portugal team states that agents must acquire three levels of knowledge: individual action execution, individual decision-making, and cooperation [33]. Individual action execution focuses on specific and necessary commands that allow the execution of a certain low-level skill. In terms of decision-making knowledge, this covers how agents choose which action to execute. Lastly, the third level of knowledge is related to the use of tactics, situations, dynamic formations, roles, dynamic plans, and communication protocols (referred in more detail in section 3.3).

These stratified steps of knowledge are inspired by P.Stone and M.Veloso's "Layered Learning and Flexible Teamwork in RoboCup Simulation Agents" [34], where the authors introduce the use of hierarchical layered learning into the robotic soccer behaviors. This learning method has four main principles:

- **Principle 1** mentions that for more complex behaviours a task should be decomposed into smaller sub-tasks instead of learning it directly from the agent's sensory inputs.
- **Principle 2** refers that there needs to be a hierarchical task decomposition, starting with low-level behaviors climbing until the high-levels where there are considerable domains of complexity such as cooperation and coordination.
- **Principle 3**, the ML methods used are selected according to each sub-task.
- **Principle 4** refers that by feeding each layer with a portion of its predecessor layer in respect to either the behavior used for training, features used for learning, and/or available and determining actions, ensures that each learned layer has a direct impact in the next learning layer.

Considering the previous principles, the authors propose a division of layers according to the behavior type to be learned, similar to what is described in the first paragraph of this section regarding the FC Portugal approach. As an example of the application of the principles above, the authors in publication [34] present the "RoboCup Synthetic Agents Challenge 97" which consists of a 2D competition with the intent of promoting individual learning, team planning, and execution, as well as rivals modeling. The first layer of the layered learning process reveals the individual learning of a low-level skill where each agent effectively learned to intercept a moving ball. Next comes the creation of multi-agent behavior where agents use their previously learned skill as a tool to evaluate the execution of a pass. When in this situation, it is convenient for the player to have an idea whether or not the ball will reach the other agent, this outcome depends on both teammates' and opponents' positions and their abilities to receive or intercept a pass. As the final layer, the pass-evolution learning process follows. When an agent has the ball and intends to pass it, it can see both its teammates (possible pass lines) and its opponents (possible cause for ball interception). To conclude this training framework, the agent needs to learn to select which teammate is the most likely to guarantee a successful pass. With this description, it is clear the application of the **Principle 1**, **2** and **4**. **Principle 3** was considered as well since the individual skill was learned using a neural network, the next layer's learning method was a decision tree and to conclude, the pass selection used TPOT-RL a multi-agent, reinforcement learning method.

In terms of coordination, generally in the RoboCup leagues, teams with highly efficient decision-making mechanisms are the most successful [33]. Good coordination techniques are advantageous for players to partake in cooperative behaviors, although their success is highly dependent on agents' low-level skills. Assuming these individual skills are not a concern, a basic principle for effective coordination is the ability to communicate, agents can use this to exchange their plans as well as their perception of the world with each other since it is common for an agent to have limited vision of the environment. In simulated soccer, the most important information that teammates may transmit and know is the location of the ball. For different RoboCup leagues, a special agent called Coach can be used, this agent has a full and exact view of the world. However, the use of communication in these competitions is usually constrained, making it impossible to depend only on the coach's messages to define the world state [33]. Moreover, with leagues having frequently its rules updated, some alterations to the communication procedures had to occur (at section 3.3.6 can be found a more detailed description on this subject). Returning to Stone's and Veloso's publication [34], it mentions the use of a mechanism called Locker-Room Agreement as the base of the agents' coordination mechanism. This agreement specifies how agents should behave when in low-communication, time-critical, or adversarial environments. This enables players to coordinate with each other without communication or reducing it to a minimum. On "Coordination in Multi-robot Systems: Applications in Robotic Soccer" [35] it is referred that the FC Portugal team wanted to resolve the issues that came with using communication without dismissing it. This was possible by implementing the following strategy: evaluate the importance of each piece of data wanted to be sent. This is done through utility metrics based on the current match situation and the estimated knowledge of teammates, and so, the dilemma was surpassed leading to the development of the Situation-Based Communication framework explained at section 3.3.4.

3.2 Low-Level Skills

To develop high-level skills, it is useful to understand the fundamental skills that can be used to construct them. As such, the first approach must be to probe the available low-level skills developed by the FC Portugal 3D team. Knowing the strengths and weaknesses of these skills makes it easier and more accurate to decide which implementations are better suited for the desired high-level skill.

3.2.1 Walk

The humanoid robot's motion consists of bipedal locomotion where the robot's movements are supported by its two legs presenting a balance and robustness challenge where the goal is for the robot to move in the most "human-like" way possible. To overcome this problem, earlier developments appealed to the use of a balance control based on the Zero Momentum Point (ZMP) stability indicator as well as the model based on the inverted pendulum. In the FC Portugal team, walking is modeled through the movement of ZMP and CoM (Center of Mass) trajectory that can be calculated using a model to approximate the bipedal robot dynamics, such as a cart-table, linear inverted pendulum, or, as mentioned before, an inverted pendulum model.

A **forward walk**, being a trivial type of locomotion, was one of the firsts to be developed, in this skill the robot only moves in a straight line. The research described in [36, 37] presents different approaches for this implementation.

Following this, it is advantageous, yet more complex, to generate a **omni-directional walking** behavior, this walk consists of giving agents the competence to move either in a straight, curved, sideways, or diagonal way, along with being able to change their trajectory direction while on movement. This improves the robot's ability to maneuver within a dynamic environment such as one in a soccer field. In “FC Portugal 3D Simulation Team: Team Description Paper 2019” [32] it is stated that the design of the controllers that grant an agent with the **omni-directional walking** skill rest on both the inverted pendulum and the cart–table models. This walk can be decomposed into different modules: ZMP Trajectory Generator, Foot Planner, CoM Trajectory Generator (generates the CoM by using the dynamics equations of inverted pendulum model [38], or cart-table model [39, 40, 41]), Feet Frame Computation (computes the position and orientation of both feet relative to the CoM frame) and Active Balance (related to maintaining stability).

A great advantage to be had in soccer is high-speed players, in robotics the faster a robot moves the more energy it consumes. Given that humanoid robot energy resources are limited, optimizations were made to previous methods with low energy consumption in mind. Some publications show a successful development of an energy-efficient human-like stable walk [42, 43]. To enable a robot with a fast walking skill, i.e.**running**, research reveals the importance of optimizing the hip height trajectory of the robot’s legs [44], and also that maintaining a fixed height during walking makes a robot slower [38]. The results of these optimizations show that a robot with a variable height while using the inverted pendulum can, at the same time, walk faster and with more energy efficiency [32]. It was also explored and tested a walking skill using a hybrid ZMP-GPC framework that allows for a **fast omni-directional walk** while still being robust to perturbations [45]. Despite promising initial results, the speed of this new walking engine was still beneath the team’s previous work [32].

In 2019 the use of reinforcement learning techniques brought a new perspective on developing skills from scratch, having achieved satisfactory results in a matter of hours, capable of replacing years of work and refinement. With the use of the PPO algorithm, the **running** skill was upgraded surpassing previous versions [46, 47], and promising results were shown in the integration of **dribbling**. This skill presents a challenging learning scenario since it involves a moving object not incorporated into the robot’s anatomy (the ball), revealing it to be an extra obstacle for the agent’s movement coordination. And so, this implementation still needs adjustments and is not yet ready to be used in the team, which, for now, uses a previous implementation of a **dribbling** skill developed without the use of ML. Considering that this recent alternative needs corrections to be able to run at full speed while maintaining the required equilibrium properties [47].

The most recent development in the team was a **sprint** skill briefly mentioned in [48]. The sprinting robots are extremely fast, with a maximum speed of 2,48 m/s. When comparing this to the **running** skill results, it is reveled a significant improvement in velocity, since the highest speed achieved until the **sprint** skill was 1,44 m/s. Nevertheless, the **running** has a better turning speed. The **sprinting** skill is very useful in situations when an agent needs to get to a position quickly, to achieve such high-speed it sacrifices the stability needed to perform turning movements without falling, i.e when robot sprints it should be in the same direction otherwise it might fall.

3.2.2 Kick

The main objective in soccer is to score goals to win the game, for this, a kicking skill is required. Similar to walking, throughout the years the kicking skill was developed/improved to be used in different game scenarios.

The most elementary kick consists in a **front kick**, this kick is based on static keyframes used to define the trajectory of the feet with a cost of a lack of flexibility. Furthermore, a preparation phase is needed for the robot to position itself to kick the ball forward in the desired direction. This setup is somewhat long and limited to a specific distance [49]. In a manner to optimize distance, a **forward kick** was developed using the Hill-Climbing method [50] as well as another optimization to get the ball to higher distances in the shortest time possible, this was done with the Tabu Search algorithm and is named as a **side kick** [50]. Given the inflexibility and the need for a preparation phase in the **forward kick**, the motivation for developing a **omni-directional kick** emerged. This skill grants the robots the ability to kick the ball in different directions and, by having no preparation phase, the kick/pass can be performed faster preventing the interception of opponents. A path planning module was used to create the trajectory for the foot to follow and be able to kick the ball in the intended direction. The movement was based on both an inverse kinematics module that computes the value of the joints to place the foot at a given position and the stability module responsible for the robot's stability [51]. By focusing on improving previous skills in terms of distance, the **long kick** was able to reach over 15 meters, ideal for scoring long-distance goals or for kicking the ball into the opponent's half in a counter-attack with the drawback of having a slow setup.

Nonetheless, these approaches focus on optimizing parameters for individual tasks, such as a kick reaching greater distances or being able to kick to a single and specific distance. Controlling and adapting a kick's distance allows the players to have more control and options regarding their next decision, and in a way, gain an advantage in the game's outcome. With this, the arise of **kick with controlled distance** was eminent [52]. This skill improves the player's flexibility allowing the ball to be kicked in a variety of distances, from 2.5m to 12.5m. This approach combines the update rules of Contextual Relative Entropy Policy Search (CREPS) with the stochastic search algorithm Covariance Matrix Adaptation (CMA-ES) that resolves the CREPS's premature convergence problem, resulting in the contextual relative entropy policy search with covariance matrix adaptation (CREPS-CMA). Regardless of being slower than the **omni-directional kick**, given that requires a setup, it is far more accurate.

Lastly, the FC Portugal team developed a **kick in motion** ability [48]. This skill was designed to enable players to move towards the ball and then kick it without stopping. The agent was trained using reinforcement learning techniques, namely the Proximal Policy Optimization algorithm. Additionally, from a static position and without using RL, a **fast kick** was also developed to move the ball within a medium distance range. The longest distance when using this behavior is greater than the maximum achieved using **kick in motion**, however, the need to stop the movement before kicking makes this skill more time-consuming.

3.2.3 Get Up

During games, agents fall frequently, being it from direct shock with other players or from spontaneous falls while executing certain movements, for example, after performing a long kick or after running and having to do an abrupt stop. As a consequence, a skill that allows agents to stand up after falling is needed. **Getting up** as soon as possible after falling is required, as an agent that is lying on the ground can not contribute to the team's success.

With the introduction of more realistic rules in the 3D sub league, this skill was recently updated to match the new requirements in the competition. These improvements were based on a black-box optimizer, the CMA-ES widely used in optimization problems [53]. This solution outperforms previous implementations by wide margins while simultaneously respecting the new rules.

3.3 High-Level Skills

The RoboCup Organizers' view on all simulation leagues is that the research done in these should center on the higher and more complex level modules of soccer robotics: the high-level decision and the coordination of, hopefully, heterogeneous robots [33]. Accordingly, the development and implementation of high-level skills precede the existence of a 3D Simulation competition. This sub-league was introduced in 2004 [1] and the FC Portugal 3D team of that year was strongly based on the previous FC Portugal 2D team. The high-level structure was identical, adapted to the 3D specificities however, all the low-level skills had to be rewritten [54].

In the 2D league, simulation is simpler, thus allowing developments to abstract from low-level details and shift research efforts to high-level skills. These games consist of two teams of eleven virtual players modeled as circles that play soccer in a two-dimensional field represented by a central server, called SoccerServer [33].

3.3.1 Coaching

A coach can have an impressive impact on a team's performance, the use of such a player type was introduced in a former existing league called Coach League. The competition in this league is based on the ability of each coach to recognize and predict patterns found in opponents' teams. The coach with more accurate patterns recognized wins. This agent was first developed for the two-dimensional league and later adapted for the 3D simulation league with the intent of being used to also coach this team [32]. Furthermore, the tips given by the coach agent to other players are done in a particular language, and so, the FC Portugal team created a language called COACH UNILANG [55] which can give high-level and/or low-level coaching instructions. The former can coach play modes into either offensive or defensive types, define game pace, the pressure made on the opponent's teams, and the formation selected for different situations.

3.3.2 Player Types

During the first years of each Simulation league, teams were made of homogeneous agents meaning that all of them were built in the same fashion and with equal abilities. Nonetheless, including players with different talents and thus having roles according to those abilities creates a more dynamic and challenging environment, closer to how real soccer teams operate. The concept of player types consists of assigning a specific role to certain players, making some agents play more aggressively (like midfielders), and others more defensively (defenders). This is achieved by allocating different characteristics for players including strategic, ball possession, and ball recovery characteristics [56].

3.3.3 Dynamic Positioning and Role Exchange

To improve flexibility, players can switch their position and role (player type) in the team's formation dynamically and in real-time with the support of the DPRE, Dynamic Positioning, and Role Exchange, system [33]. These exchanges can only occur if it is revealed to be a beneficial trade for the team [56].

3.3.4 Situation Based Strategic Positioning

The FC Portugal team strategy makes use of different player types and tactics applied over several formations. These can be used for distinct game situations as in defense, attack, or even transition from one to another. The management of formations and player types is based on SBSP – Situation Based Strategic Positioning algorithm [32]. There are two main types of situations that an agent can be [56]: an active situation (defined by ball possession or ball recovery) or strategic situation.

SBSP mechanism is used for the latter (situations where the agent trusts that it is not going to enter into an active behavior soon). Agents stop this strategic positioning when they get into a critical behavior mode, being it by having the ball in their team's possession or charging to recover the sphere. This arranges players to be in advantageous positions when they are not part of the occurring play, making the agents ready to transit to an active situation when ball circulation or recovery is necessary, enabling the team to move evenly keeping the field completely covered.

3.3.5 Visual Debugging

In light of the game's highly dynamic environment, agents have to quickly make decisions and take action in a limited and brief time as a response to the world around them. And so, it is believed that an offline analysis of agent's decisions, without the real-time pressure, is extremely practical and also provides a more thorough examination of agents reasoning and consequent action [33].

The FC Portugal team has developed many ways of debugging agent's behavior. The Visual Debugger, inherited from the 2D team to the three-dimensional team [54], is capable of describing the current state of the world based on the visual data saved in the server's record log files [56]. With this information, it can calculate the positions of all objects and by comparing their previous and actual positions, velocities can also be known. Furthermore, the last player commands (the most probable to be correct) can be recovered as well.

3.3.6 Intelligent Perception and Communication

The means that enable communication in the simulation leagues restrict its usage by limiting the amount of usable bandwidth, which leads to a narrow capacity for sharing information. In the Portuguese teams, communication is supported by the ADVCOM system [56]. Apart from low bandwidth, an update on the 2D simulation rules in 2002 imposed that messages had to have a maximum length of 10 bytes, which led to mandatory adaptations where only the fundamental information should be transmitted within the agents' team.

Moreover, the simulator used for the simulation leagues is only capable of delivering one message from an agent to another per cycle thus, the FC Portugal team developed a protocol [33] that makes all the players estimate the importance of their knowledge and the utility it has for the team through utility measures. Using the protocol forces a player to communicate only when its utility surpasses the others or if it is above a certain predefined threshold.

3.3.7 Setplays

As mentioned before, the modesty of the games played in 2D enables research to center on high-level skills instead of low-level behaviors. Setplays are commonly used in many team sports, and in the simulation leagues, represent the highest levels of coordination and cooperation to develop in a team. These pre-thought flexible plans are intended to be executed in real-time during games against an enemy team [57] similar to real soccer matches. To carry out these strategies all of the agents in a simulation team possess a playbook where all of the usable setplay definitions are. Whenever or not a certain setplay should be proceeded or even stooped needs to be supervised by an engine that continuously monitors the state of the match and world. Therefore, these decisions are commissioned by the coach/programmer who can rapidly define a new setplay to overcome a given obstacle without needing to alter or recompile the team's base code.

With the improvements made though time since the first use of these strategies, the application of setplays evolved from just being used in specific situations, like kick-ins, kick-offs, or corner kicks to be available during game times where agents are playing and not in a "stopped" situation as the one mentioned before, i.e, during a game if the world state seems suitable to use a setplay this can be immediately selected and executed [58] contributing to the improvement of the team's flexibility resulting in better performances. The coordination of the players' skills and execution of the setplays is based on a specific language domain used for their constructions [57]. However, writing all the needed definitions manually is harsh, time-consuming, and liable for errors. Thereby, a graphical solution, named Strategy Planner (SPlanner), was developed to allow users to escape from the complexity of designing setplay by hand. With this tool the constriction of these plans is very intuitive, reducing the errors and speeding up their definition.

The first RoboCup competition to test the usability of setplays in a competitive environment for the three-dimensional league happen in 2006. The FC Portugal 3D team won first place in its respective league that same year. Nevertheless, the use of setplays in 3D was applied in an older version of the game simulation where players were modeled as spheres. In the following year, the use of humanoids agents was introduced, and the complex dynamics of these models resulted in forcing the high-level implementation such as setplays to be put aside in favor of improving the low-level skills [58]. Since then, as already mentioned, low-level behaviors have evolved and the focus is once again shifting to high-level skills. At the time of writing this dissertation, research and testing related to re-including setplays in the 3D league are already ongoing.

3.3.8 Reinforcement Learning Approaches

In recent developments for the 2D RoboCup team, the evolution and expansion of computational power and new reinforcement learning algorithms led to a rising interest in using this sub-field of machine learning to enhance behaviors quality in harsh scenarios. Results regarding two high-level skills covered by the Portuguese team in "Learning High-Level Robotic Soccer Strategies from Scratch through Reinforcement Learning" [59] reveal interesting conclusions regarding the influence of the rewards given to an agent during its learning process and the relation between the reward with the episodes initial conditions. The paper mentioned before presents two different learning environments, the first scenario consists of an agent **approaching a moving ball and scoring** against a keeper. The other is defined by two cooperative players trying to learn to **pass and receive** the ball alternately between them. A third agent plays as an opponent to cut the other learning agents' pass line. To train both of these scenarios the PPO algorithm was used due to its great level of performance and ease of tuning. The scoring scenario exposed the importance of a target distance reward that compensates the robots for reducing the distance between either itself or the ball and a target, which could variate or be static. In regards to the passing scenario, this reward was revealed to be insignificant. Another important component of rewarding in the first scenario was the penalty that reprimands the agent for each step it takes until it executes the first kick.

The most recent high-level skill implementation for the FC Portugal 3D team consists of a kick-off. The publication "Learning Low-Level Behaviors and High-Level Strategies in Humanoid Soccer" [53] presents a distributed framework capable of learning both low-level behaviors and high-level strategies, like a kick-off scenario. Starting with the optimization framework of the higher skills, these can be based on a multi-agent reinforcement learning algorithm using strategies from the MAS to achieve agents coordination. In this particular scenario, the optimization was done with the use of an objective function (fitness or reward function) that optimizes the parameters that determine each agent's decisions. The used framework supports multiple reward-based learning algorithms, placed over a network of multiple workers, in this case, the reward-based optimizer used was CMA-ES.

Regarding low-level behaviors, developments followed the concept of layered learning with hierarchical task decomposition that allows learning to happen at all levels of the hierarchy, having each layer affect the learning of the subsequent superior level, this subject was discussed in the previous 3.1 section and is explained in more detail at [34]. For the high-level strategies, an extension of the layered learning is considered, the overlapping layered learning [60] which consists of learning both in series (similar to layered learning) and in parallel. After a layer has completed its learning process, only some elements of it become frozen allowing the rest to be accessed and modified during the learning of the following layers. The elements left open in the former layers are the “overlap” with the next layer. Moreover, overlapping allows for behaviors to be learned independently in parallel, afterwords they are combined by learning at the “seams” where their influences overlap. Results from [53] reveal that the complexity of the optimization problem can be reduced by starting the learning process with the low-level behaviors and afterward shift to the optimization of the high-level strategies. Additionally, due to overlapping, it is doable to optimize complementary skills together even if they are not in the same layer, such as a kick to the goal and a strategy that ends in a kick to the goal. With this optimization, the current kick-off strategy outperforms previously implemented solutions by wide margins. For future work, it is suggested the use of another algorithm, the Proximal Policy Optimization.

3.4 Support Tools and Technologies

Bearing in mind that this dissertation is based in a particular environment, specific for the simulated humanoid robot competition, it is fundamental to know the technologies used, in particular the server (SimSpark), the used interface (RoboViz), as well as the type, and characteristics of the agents used in these games (NAO robot). Furthermore, the FC Portugal team has its own tools to facilitate the training (FCP Gym) of agents as well as recommend software to do so (Stable Baselines).

3.4.1 SimSpark

SimSpark is a physics simulator for multi-agent systems for 3D environments, it is built upon the Spark application framework and it is the official simulation platform for the RoboCup competitions. However, it is also usable as a generic and flexible simulator for various types of simulations, thus for the RoboCup Simulation league, the rcssserver3d was created. The latter consists of a soccer simulation server running on top of SimSpark used with a specific 3D physical soccer simulation in mind. Nonetheless, is important to notice that the three-dimensional competition appeared from the efforts put into the 2D competition, and so, soccer games went from having players as circles to spheres (from 2004 to 2006), and only in 2007 was the humanoid player made available. These improvements made through the years were supported by adjustments to the simulator’s core architecture [61, 62].

3.4.1.1 Simulated Field

A full description of the soccer simulation can be found at the SimSpark Gitlab [63], a summary of its main characteristics regarding the soccer field follows:

- Dimensions are 30 by 20 meters.
- Goals have a height of 0.8 meters and are 2.1 m by 0.6 m in length and depth.

- Penalty area is 3.9 m by 1.8 m (full penalty area width includes the goal width and the penalty width extending to 6 m overall).
- Center circle has 2 meters of radius.
- The ball has a radius of 0.04 m and a weight of 26 grams.
- The filed is divided by two axes, the origin point (x = 0 and y = 0) is at the center of the center circle. The x-axis extends from -15 to 15 and the y from -10 to 10 as can be seen in the illustration below.

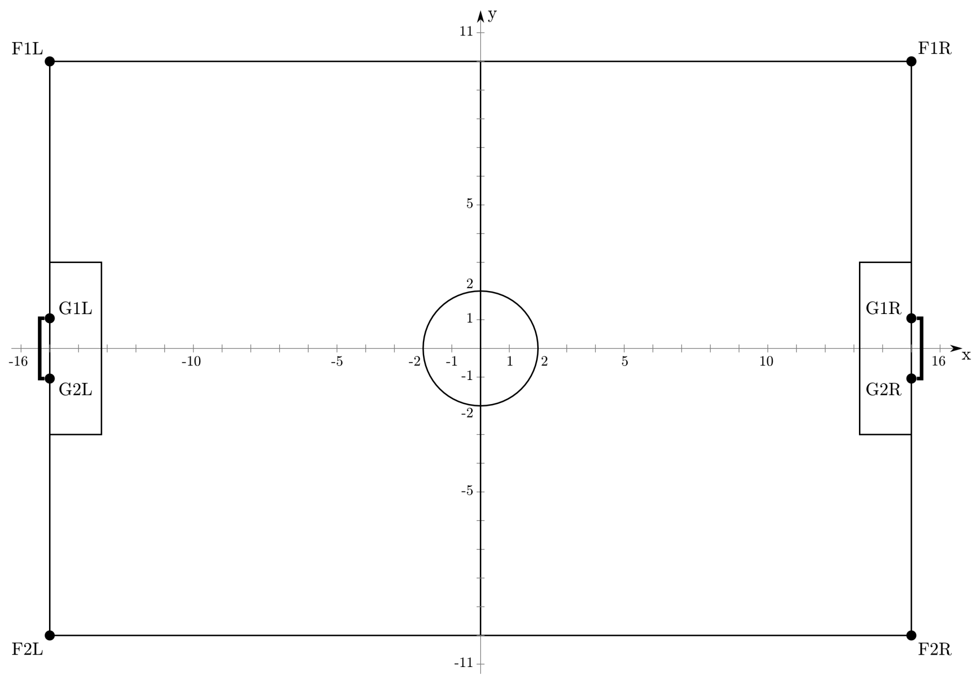


Figure 3.1: Simulated soccer field dimensions [63].

To aid the agents to know their relative position, all the goal posts and corners in the field have a distinctive static marker. Given that agents know the position of these markers along with having the field lines in their line of sight their global localization is easily known.

3.4.1.2 Standard Architecture

For the RoboCup 2021 3D Simulation league games, it was used the most recent version of SimSpark (at time of writing 0.3.2) and rcssserver3D (now 0.7.3) in Linux Ubuntu 18.04 64 Bits. The two figures below show the software and hardware setup for the competition's environment. Each team is assigned a client computer (Client 1 or Client 2), within each one, every agent needs to be connected to a proxy that will connect with the server. For visualization purposes, the server is linked to a monitor (RoboViz) [64].

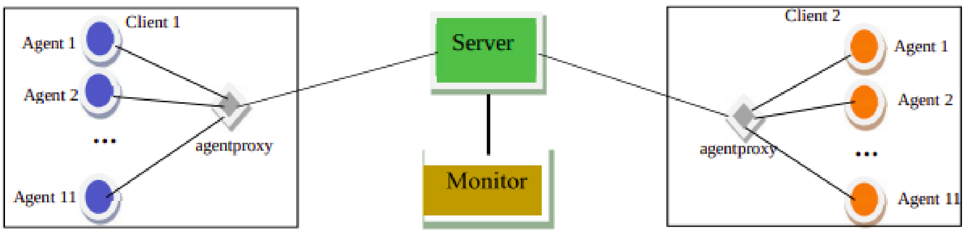


Figure 3.2: SimSpark's architecture during competition [65].

Agents can communicate with the server using an Agent Proxy. This proxy has two threads, one for forwarding server messages and for keeping sync-times, and other for sending agent action messages. During games, the simulation runs in a real-time mode having all players in each team in sync to guarantee fairness. This is done by having the forwarding thread wait 20 ms after sending all messages to the agent, then it sends a "sync" command back to the server [66].

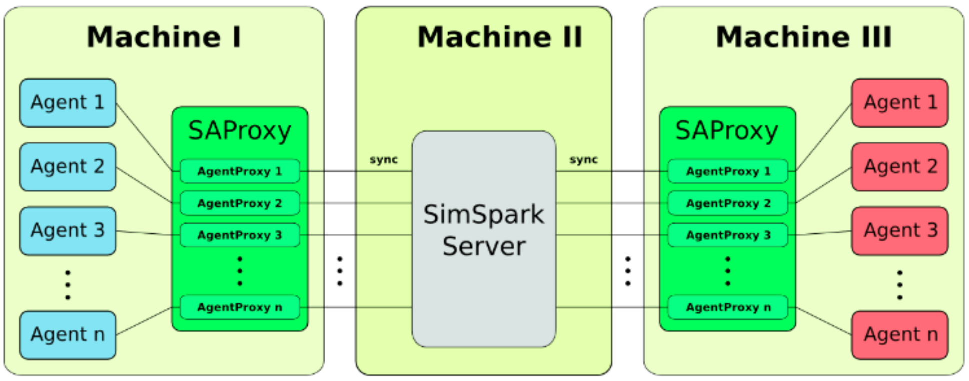


Figure 3.3: SimSpark's architecture during competition in more detail [66].

When an agent connects to the server, it is initially invisible and not able to affect the simulation, thus each agent must send a "*CreateEffector*" message to the server after establishing the connection. This message contains a scene description that instructs the server on how to construct the physical representation and all further effectors[1] and perceptors[2] of the agent. After an agent is visible on the simulation, the server sends groups of messages containing the output of the agent's perceptors and, as a response, the agent can influence the simulation by sending effector messages [67]. Thus, the cycle that repeats every 20 ms goes as follows:

1. The server sends individual messages with sensations to each respective agent;

2. Based on their knowledge of the world, agents can decide to take new actions;

[1]Effectors allow agents to engage with the simulation, by performing actions that could lead to its alteration. The SimSpark Gitlab offers a more detailed explanation on effectors.

[2]Perceptors represent the senses of an agent, this is how the agent knows its model and the state of the environment. The SimSpark Gitlab offers a more detailed explanation on perceptors.

3. Following, agents can then send their messages to the server for aimed actions;

4. The server compiles the agent's messages and calculates the resulting new environment (poses and locations, ball movement, and many others) according to the laws of physics and the game.

As for the communication between the SimSpark and the monitor, the latter connects to the server using the monitor protocol, and the next exchange follows [67]:

1. The server sends a message containing world description and subsequently the full scene graph;

2. Then it sends a game state message, followed again by the full scene graph;

3. The server continues to transmit partial game state segments followed by a full or partial scene graph, depending on what has been updated in the meantime.

For either receiver of the server's messages (agent or monitor) the communication is done via TCP in their respective ports.

3.4.1.3 Training Alterations

Regarding skill learning, a few alterations need to be made in the setup files to speed up the agent training. For starters, when the games run in real-time, it forces the server to wait 20 ms for the agent's effector message. If the agent does not respond in this time window, its behavior will degrade by having its message ignored. Nonetheless, if the full team responds before the server's waiting time terminates, then the CPU is not utilized at its maximum. Hence this mode can be deactivated in favor of the Agent Sync Mode, which supports the full use of the CPU's resources, making the training faster. In addition, it guarantees that no server cycle is wasted by forcing it to wait for the agent's commands before proceeding to the next cycle. This will result in a rapid optimization in situations that require an extensive amount of iterations, decreasing the time needed to reach the desired results [68]. Other alterations can also benefit the training environment such as, disabling the soccer rules and noise as well as letting the agent know its and the ball positions with high precision.

3.4.2 RoboViz

RoboViz is the official display interface for the RoboCup 3D Simulation League. It was developed and is still maintained by the *magmaOffenburg* team, an active participant in the simulations competitions. This tool allows the visualization of results during training, being helpful for analyzing the agents' learning process [69].

```
/usr/local/share/rcssserver3d/rcssserver3d.rb
        $enableRealTimeMode = false
/usr/local/share/rcssserver3d/naosoccersim.rb
        addSoccerVar('BeamNoiseXY',0.0)
        addSoccerVar('BeamNoiseAngle',0.0)
        #gameControlServer.initControlAspect('SoccerRuleAspect')
/usr/local/share/rcssserver3d/rsg/agent/nao/naoneckhead.rsg
        (setViewCones 120 120) ;
        (setSenseMyPos true) ;
        (setSenseMyOrien true)
        (setSenseBallPos true) ;
        (addNoise false)
~/.simspark/spark.rb
        $agentSyncMode = true
```

Figure 3.4: Setup alterations for training environment

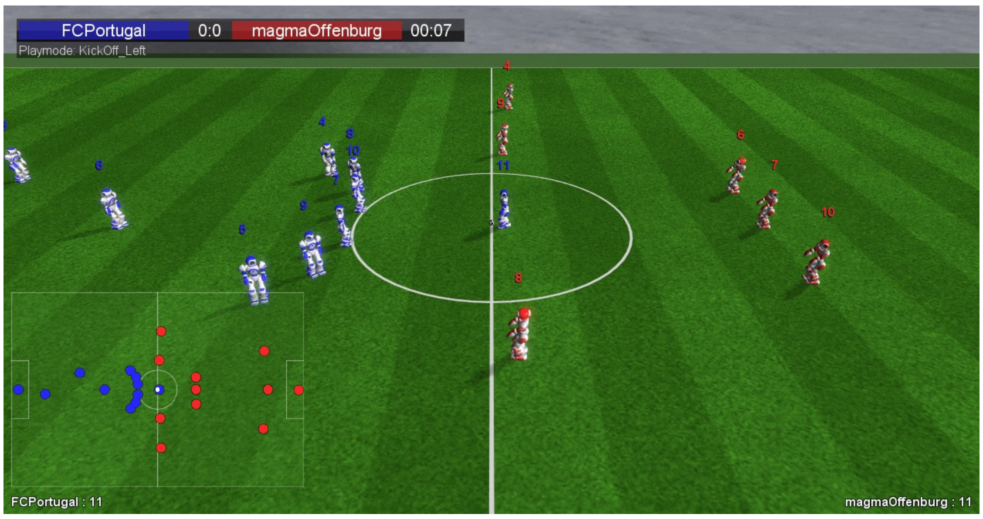


Figure 3.5: Example of a still image of the RoboViz visualization [69].

3.4.3 NAO Robots

The NAO robot seen in figure 3.6, was developed by SoftBank Robotics and, it is the model used for the agents in the 3D simulation [70]. The simulated NAO in 3.7 has 22 degrees of freedom resulting in 22 joint perceptors[3]. It is equipped with a gyroscope and accelerometer in its torso to monitor its radial and axial movement in the three-dimensional space. To detect contact with the ground or other objects, in each foot the agent has a force resistance perceptor, for visual detection and communication it also includes a restricted vision perceptor at the center of its head and a hear perceptor [71].

[3]There are five types of heterogeneous players available, each with different purposes. These are distinguished by their number (e.g., *nao0*, *nao4*. The type 4 agent has two extra joints in each foot used as toes, having a total of 24 degrees of freedom.

Figure 3.6: Real NAO [72].

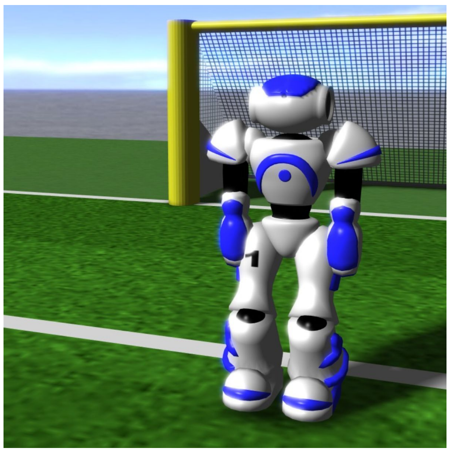


Figure 3.7: Simulated NAO [71].

3.4.4 TensorFlow

TensorFlow released in 2015 by Google consists of an open-source library used for building machine learning models in dynamic environments, its base language is Python. It uses dataflow graphs to represent computation, shared state, and the operations that alter that state. It offers several levels of abstraction so users can choose which one best suits their needs without sacrificing speed in execution and being a highly flexible [73].

This software includes a tool called TensorBoard that can collect and display data relevant during training, which can be useful to be analyzed later or even while the learning is happening. TensorBoard presents information, such as the reward of all the episodes, the model losses, the observation, and other parameters unique to certain models [74].

3.4.5 Stable Baselines

Due to the complexity of implementing reinforcement learning algorithms by hand, the use of existing implementations is quite common. The most well-known set of algorithms is from OpenAI Baselines, however, the use of the Stable Baselines (SB) implementations has been gaining popularity over time since it has proven to give high-quality results and was designed to be easier to use, adapted, replicate and refine. SB is a fork of the OpenAI Baselines, its environments follow the gym interface[4], thus allowing custom environments. It supports TensorFlow and TensorBoard and has been used in previous FC Portugal 3D implementations with great results.

3.4.5.1 Available Algorithms

The table below presents the available reinforcement learning algorithms implemented by Stable Baselines as well as some characteristics that may be determinant when choosing one to use.

[4]Gym is a toolkit with an open-source Python library developed by the OpenAI used to create and compare reinforcement learning algorithms. It provides an API to communicate between learning algorithms and environments https://github.com/openai/gym (last accessed 14/08/2021)

The first column shows the available algorithms, followed by if said algorithm supports recurrent policies. The "Box" and "Discrete" columns describe if the algorithm can be used with a continuous and/or discrete action space. The last column informs if the algorithm can have multiple environments all training in parallel.

Table 3.1: Stable Baselines RL algorithms and their characteristics.

Algorithm	Recurrent	Box	Discrete	Multiprocessing
A2C	Yes	Yes	Yes	Yes
ACER	Yes	No	Yes	Yes
ACKTR	Yes	Yes	Yes	Yes
DDPG	No	Yes	No	Yes
DQN	No	No	Yes	No
HER	No	Yes	Yes	No
GAIL	Yes	Yes	Yes	Yes
PPO1	No	Yes	Yes	Yes
PPO2	Yes	Yes	Yes	Yes
SAC	No	Yes	No	No
TD3	No	Yes	No	No
TRPO	No	Yes	Yes	Yes

Most of the algorithms present at table 3.1 have been introduced in the past section 2.5. Those that were not, have a brief description below.

- **ACER**: "Actor-Critic with Experience Replay" combines different aspects of multiple algorithms: it uses an actor-critic deep reinforcement learning agent with multiple workers similar to A2C, as in DQN, it relies on a replay buffer, and it also uses retrace for Q-value estimation, importance sampling, and a trust region for optimization [75].

- **ACKTR**: "Actor-Critic using Kronecker-Factored Trust Region" combines actor-critic methods, the trust-region optimization where the curvature approximation uses Kronecker-factored which improves sample efficiency compared to A2C [76] and is less computationally expensive than TRPO, plus it is an improvement to scale larger deep neural networks [18].

- **HER**: "Hindsight Experience Replay" wrapper that performs along with off-policy methods [77].

- **GAIL**: "Generative Adversarial Imitation Learning" is a model-free algorithm that learns a policy after recovering a cost function using expert trajectories. It learns by imitating complex behaviors in high-dimensional environments, and by learning the cost function from expert demonstrations makes this algorithm part of the IRL, Inverse Reinforcement Learning implementations (only implemented for TRPO)[78].

3.4.5.2 Vectorized Environments

Vectorized environments are a method that allows training of the same agent to happen in **n** environments per step instead of training only in one per step. As a consequence, the actions, observations, and rewards are passed to the environment as vectors with dimension **n**. Stable Baselines offers two possible options to create vectorized environments, *DummyVecEnv* and *SubprocVecEnv*, both work in continuous and discrete action space, however, the former can not be used as multiprocessing (runs all environments in the same process), the latter runs each environment in a separate process. The use of this method makes the collection of experience happen faster and grants a more diverse range of states improving exploration [79].

3.4.6 FCP Gym

The FCP Gym is a tool exclusive to the FC Portugal team, it is a framework for implementing reinforcement learning algorithms similar to the popular OpenAI Gym. Despite the Gym from OpenAI being highly used and recommend for its flexibility, the cost of communication between the FCP agent code, done in C++, and the Python code used for the optimization is too high, even for small changes. As a way to bypass this problem, the FCP Gym was developed by Tiago Silva, a member of the FC Portugal 3D team, that keeps similar aspects regarding the execution of RL algorithms with OpenAI but was able to solve the communications issue by using a third-party library (pybind11). Thus, it became possible for various aspects of the reinforcement learning processes to shift to the C++ code, such as the definition of rewards, action, observation spaces, and others.

3.4.7 Full Learning Cycle

As mentioned before, this project is based on the implementations provided by Stable Baselines resulting in the training models being available in the same language as the RL algorithms, in this case, Python. Figure 3.8, shows a diagram that intends to facilitate the comprehension of the communication between the Python side and the C++ gym. Assuming that the intended environment to learn was created beforehand[5] (in C++), the Python code starts by launching *nenvs* (*n* environments), and given that the desired learning scenario demands two agents, each pair of environments, one per agent, will share the same SimSpark instance, i.e when training two agents at the same time, *nenvs* must always be an even number. Furthermore, each agent has to run in a unique process and, to be able to share the same world as its corresponding teammate, the pair of players are launched with the same agent and monitor ports resulting in $\frac{nenvs}{2}$ SimSpark instances to train.

[5]The FCP team provides default environments as examples, one of these is a MAS available since the begging of this project, considering that the use of multi-agent in the FCP Gym was not explored until now. Given that the purpose of this dissertation is the use of multi-agents, the diagram at 3.8 refers to these systems. In the case of a single agent per training, the diagram would be more direct and simple. An example of the Python code using the OpenAI Gym can be found at the Stable Baselines documentation.

To launch a complete team, each agent enters the simulation at a different time to give the necessary data to the server without overstepping the others. When launching only two agents, the first one to connect to the server will have the uniform number equal to one (*unum = 1* or *R1*), and the next player will have a uniform with the number two (*unum = 2* or *R2*).

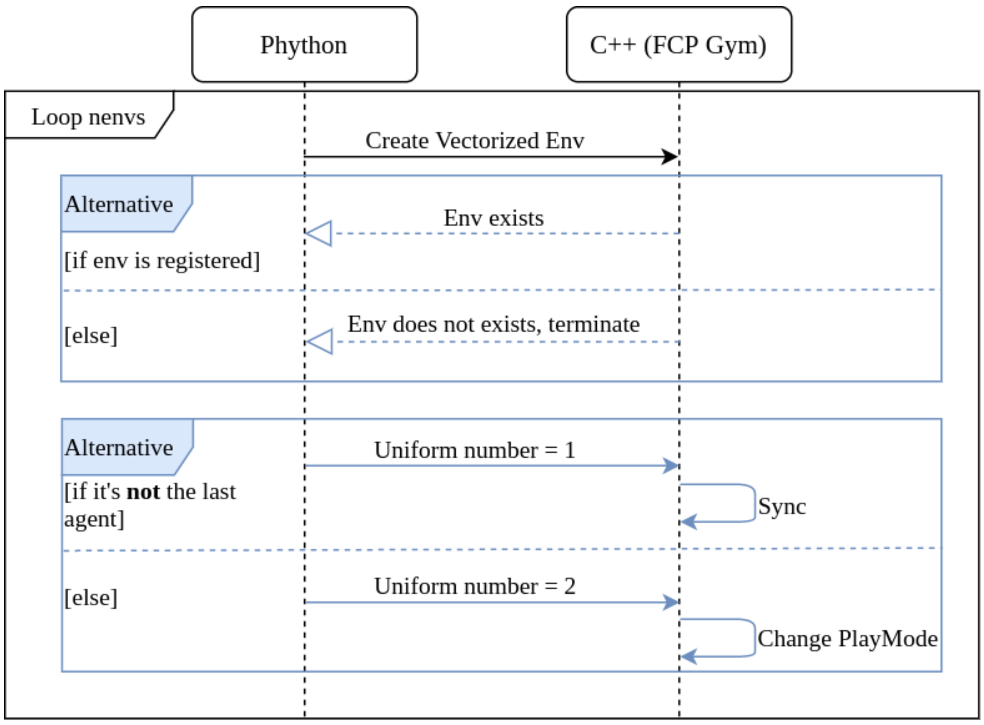


Figure 3.8: Initial steps for creating n environments (*nenvs*) with two players for each training environment.

As mentioned at the section 3.4.1.3, during training, the Agent Sync Mode is put at *true* resulting in all the agents having to Sync at the same time[6]. Given that the agent *unum = 1* will have to wait for its teammate to connect to the SimSpark, it will keep doing Syncs until the other one arrives, seeing that at this point, it is acceptable that the training procedure does not advance. Afterward, since each agent runs in its own process, a simple way to communicate to *R1* to stop doing Syncs and proceed to the learning phase, is by having the last agent that joins the instance, in this case, *R2*, change the *Game Mode*[7]. This shows that if *nenvs* is an odd number, one of the SimSpark instances will not continue to the training cycle, given that the last agent to connect to the server will never exist and, the *Game Mode* is never altered (flag that signals that both agents are present in the simulation), and so, this instance will not proceed resulting in the model to freeze in its entirely.

[6]When Sync Mode is activated, the simulation will only run when it receives a Sync message from all the agents, if this does not occur, the simulation freezes, waiting for the missing agent(s) message.

[7]*Game Mode* defines which soccer game situation (kick-off, corner-kick, penalty, etc) is the simulation in. This attribute is broadcast by the server when altered, and thus, all the agents see it concurrently. The alteration here happens from *Before Kick-Off* to *Play On*. Note that using this approach is the most efficient and straightforward way of communicating with both agents without recurring to outside communication or having to alter server specifications.

Regarding the training cycle itself, it is based on three main functions present in the FCP Gym: *reset()*, *observe()* and *step(action)*. Before an episode begins, it is necessary to place the players and ball in their respective position and orientations to create the intended environment. As an example, to train an agent to perform a penalty kick, its environment should have the ball in a specific placement, it should be placed in the position that the rules of the game dictate as the standard position for this game situation. This way, the agent will be trained in a proper and realistic setting, all of these placements happen in the *reset()* function. After the elements are in their respective positions and orientations, the environment is considered ready, and the code proceeds to the *observe()* function where the observations previously defined[8] for the agents are updated. What follows is a cycle that moves between the *observe()* and *step(action)* functions until the episode ends. The latter is where the agents' actions are applied as an output of a neural network and where the interactions and consequent alteration of the world take place. The program goes back and forward between these two functions, every time it goes back to the *observe()* and returns, the output of the neural network is updated since it received new information from the observations. In a sense, the NN is only being used when the code leaves the *step(action)* to the *observe()* and then re-enters the former. Finally, there can be many reasons to terminate an episode, however, this will always be signaled by altering the flag *done* that is set as *false* for most of the duration of an episode until the moment when it must end, then it is set as *true*.

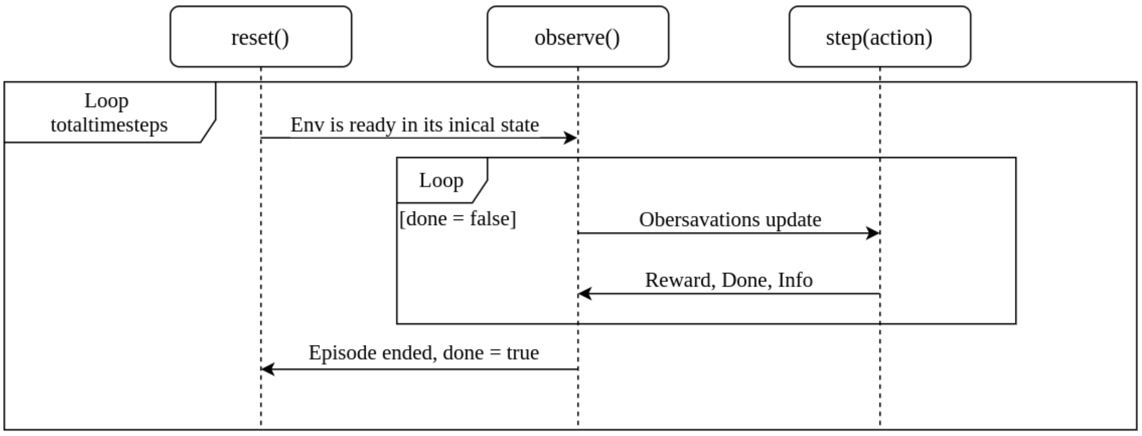


Figure 3.9: Training episode cycle.

Figure 3.9 shows a diagram of the entirety of the training cycle. Episodes will start and end until the *totltimesteps* (defined in the Python code when creating a training model) is achieved. As explained in the last paragraph, before each episode starts, it goes through *reset()* to set its elements, and then advances to the episode cycle. The diagram consists of a loop that alters between the *observe()* and *step(action)* functions where the neural network outputs values for the actions and receives the observations and reward as feedback to construct the output values. Once

[8]Both observation and action spaces are defined before the *reset()* when each agent first connects to the SimSpark, this happens at a unique instance done only once during all training session. It happens prior to either, doing Syncs while waiting for the other player, or changing the *Game Mode*.

the *done* flag is set at *true*, the episode ends, passing through *observe()* to update the network one last time and returning to the *reset()* to begin a new episode.

3.5 Conclusions

Before proceeding to the next chapter, some key ideas should be made clear. First, the agent type that will be used for both robots is the *nao4* given that kicking is a major aspect in the setplay meant to be created. Regarding the kicks already existing in the team (presented in section 3.2.2), the use of the kick with controlled distance and the fast kick, seem appealing for the passer[9] and scorer respectively, however, in recent years, the simulation 3D league imposed a rule regarding agents self-collision. This rule states that it is considered a foul when a body part of a player trespasses another body part of the same agent. The two kicks mentioned previously were developed before this rule and normally result in self-collision when used. Therefore, training with these kicks is not an ideal scenario.

While training, **both agents will be launched in the same monitor and server ports**, then in different ports, there can be other pairs of teammates in order to speed the training process, thus the ***SubprocVecEnv*** will be used to create multiple environments. In order to **use FCP Gym to solve multi-agent problems**, the succeeding guidelines must be followed: first, **every agent needs to run in its own process** (not thread), the agents sharing the **same training environment** need to connect to the **same SimSpark instance**, and have a **unique *<uniform number, team name>* pair**, and lastly, every agent needs to **sync at the same time**.

Regarding the algorithms implemented by Stable Baselines described at the table 3.1, since it is desirable to use multiple training environments the algorithms that do not support multiprocessing will not be considered. Furthermore, the initial implementations are most likely to be based on continuous actions spaces so, the chosen algorithm must be able to support this actions type. The options that fulfill these needs are A2C, ACER, ACKTR, GAIL, PPO2, and TRPO. When reviewing each algorithm, it was concluded that PPO is more adequate than TRPO so the latter will be dismissed. From the original PPO paper [80], when comparing this algorithm for “continuous control tasks...it performs significantly better in terms of sample complexity than A2C and similarly to ACER though it is much simpler”, thus these two other algorithms will also not be considered, as well as GAIL since it is used for Inverse Reinforcement Learning. The remaining algorithms are PPO2 and ACKTR, given that, in the team’s most recent reinforcement learning approaches presented at the section 3.3.8, it is recommended as future work the use of PPO, plus, both the **kick in motion** and **sprinting** skills were developed using PPO, leads to **the initial approach being based on the faster version of this algorithm, the PPO2** implemented in the Stable Baselines.

[9]Note that as of now in the 3D simulated games the throw-ins are executed with a kick similar to executing a sideline cross, contrary to what happens in real soccer where it is done using the hands and arms.

Chapter 4

Experimental Setups

This chapter presents some of the experimental implementations constructed over the course of this dissertation. Training happened both in a personal computer as well as at FEUP's LIACC (*Laboratório de Inteligência Artificial e Ciência de Computadores*) server, the Python version used was 3.6.9, along with Tensorflow 1.15.0 and Stable Baselines 2.10.1.

4.1 Episode Layout

Every environment has the same fundamental episode setup, the differences lay in what reward is being applied or in how much is the network being used during an episode, or how the ball's initial position is chosen.

4.1.1 Description

As a general description of the training scenarios, it can be said that these are meant to be a singular setplay for both throw-ins and corner-kicks within a multi-agent system environment. Episodes consist of two teammates with the objective of scoring a goal, the first agent has the uniform number equal to 1 (referred to as *R1*) and its task is to pass the ball from the sideline or corner to the other agent. The agent with uniform number 2, *R2*, is responsible for scoring the goal. The episodes will only occur in the quadrants of the field where the x-axis is positive, figure 4.1 shows the field divided into four quadrants, thus, only quadrants 2 and 4 will be used. Furthermore, *R1* will always start the episodes in the fourth quadrant, one meter being the ball outside of the field lines. As for the scorer, this agent will always start at the coordinates (11.5 ; 0). Although the episodes are restricted to a certain part of the field, the results of the training models can be used anywhere when the game situations, corners, and throw-ins, occur, the following section 4.2 explains how.

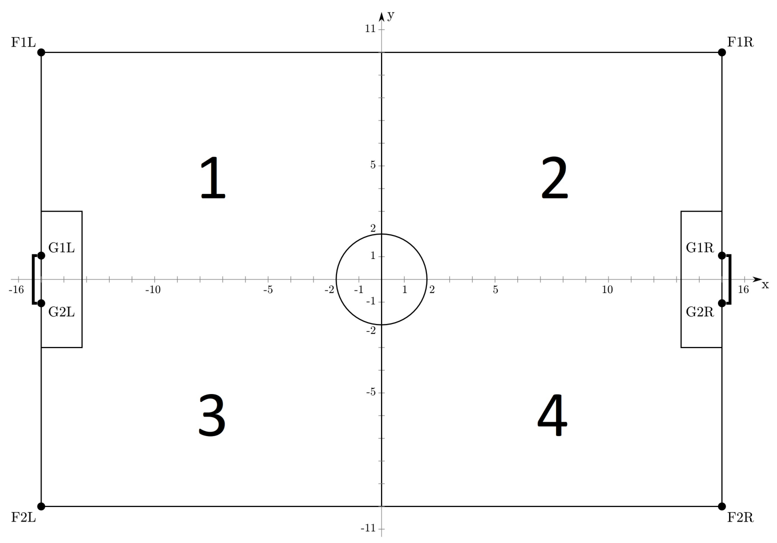


Figure 4.1: Field division into quadrants [63].

When an episode begins, *R1* will wait 4 seconds (200 time-steps) before moving closer to the ball to kick it and, in the meantime, *R2* moves to an initial position, decided by the network, where it will wait for the cross to happen. When the cross is executed, agent number 2 decides when to charge the ball leaving the "waiting" stage and proceeding to the kicking phase that hopefully ends in a goal. A final aspect to take into account is that in the simulated soccer games the corners are executed not in the corners of the field where y = 10 or -10 but at the end-lines with y = 6 or -6. For kick-ins, the minimum value for the x-axis will be 5, the area from 0 to 4.99 meters will not be applied in any environment.

4.1.2 Available Skills

Despite the extensive description of FC Portugal's previously implemented skills done in 3.2, some kicks were not mentioned in this section since there aren't any publications that include them given that some were created using the same techniques already presented in other publications, and so, only by having access to the team's code is it possible to know all the skills available. Nonetheless, the ones mentioned in the previous chapter make a good part of the complete implementation.

4.1.2.1 R1: Crossing Kick

A kick called *passMediumKick(Vector kickTarget)*[1] is used for the passing agent, this kick has a range of [3;12]m for the type 4 robots being one of the most accurate kicks in the team. This skill receives as parameters the x and y values of the desired final position for the ball (as do all the kicks implemented by the team). Comparing this one with the *slowLongKick(Vector kickTarget)*, the

[1]Some functions mentioned in this chapter have many parameters and, as a way to simplify writing and avoid massive text, when a function has more than two parameters it will be displayed as "*nameOfTheFunction(...)*" with the ellipses "..." symbolizing 3 or more parameters.

latter, as the name suggests, can achieve greater distances (maximum achieved during experiments was 21 meters although it is not always able to, 15 meters is the safest distance to consider as the maximum as referred in section 3.2.2 regarding the long kick). However, the long kick is not as accurate as the medium and, given the action space that was chosen, described in the next section, it makes more sense to go for better accuracy instead of longer distances. Both kicks are incorporate with a walking behavior that takes the agent close enough to perform a kick, this results in the agent having to stop the walk and then switch to kicking, which in itself takes a few seconds to prepare. Nonetheless, there is no problem using the *passMediumKick(Vector kickTarget)* for the *R1* agent given that in throw-ins and corners there is no risk of an opponent trying to move the ball during the pass preparation, thus, the agent can take its time to execute the kick without losing the advantage.

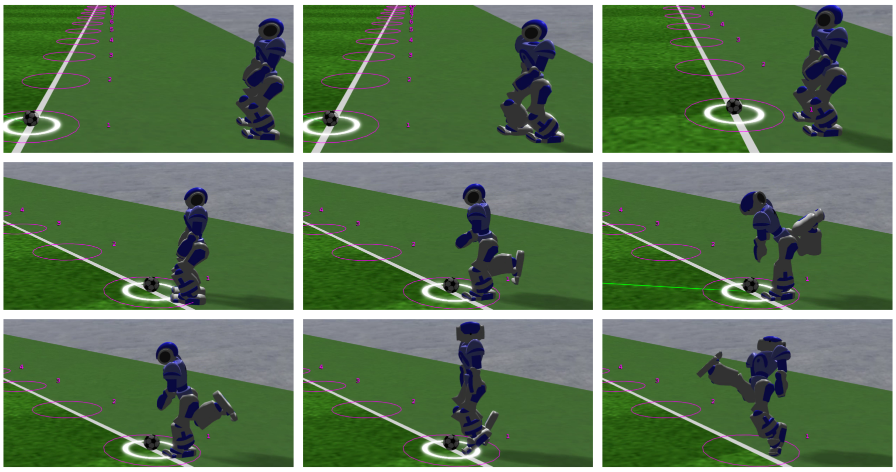


Figure 4.2: Still image sequence of agent *R1* executing *passMediumKick(Vector kickTarget)*, top row represents the walk and the rest the preparation and execution of the kick.

4.1.2.2 R2: Scoring Kick

For the player with uniform number 2, this agent is supposed to be close to the opponents' area ready to score. Taking into account the high activity in this zone, it is important that as soon as *R2* is close to the ball and has a target defined it should immediately kick it. With this in mind, the kick in motion, *kim(Vector kickTarget, float maxerr)*, is used to kick the ball to goal since the agent has the capacity of kicking without having to stop first. As mention above, this agent goes to an initial position while its teammate gets ready to cross, this movement is done with a previously implemented walk function the *goToTargetChallenge(...)*. This walk can be used for general motion since its highly flexible although not very fast, however, given its robustness in maintaining equilibrium during trajectory alterations this walk is used in most FCP's kicks to approach the ball including all of those mention in this chapter[2].

[2]All the kick functions have a walking stage before executing the kick itself, however, only in *kim* did the training incorporate the walking skill so that the agent learned kick without stopping its movement.

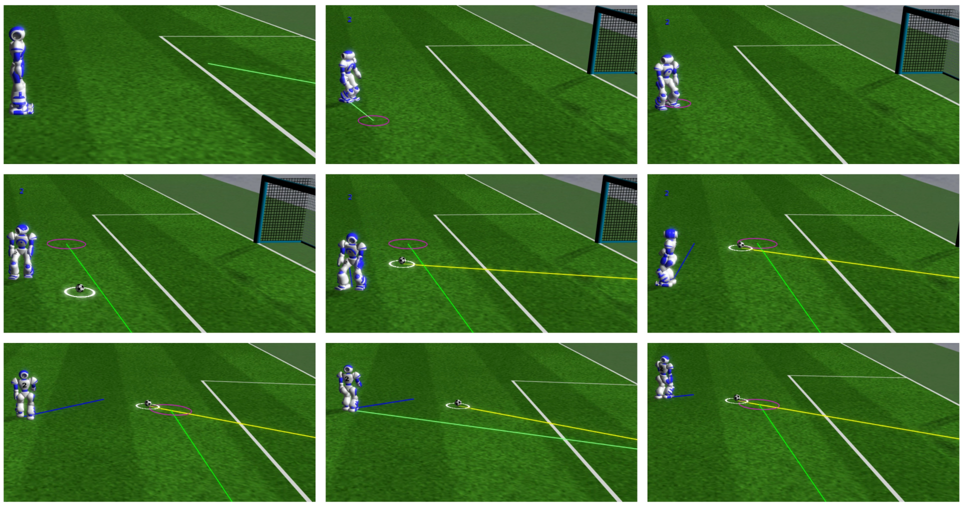


Figure 4.3: *R2* moving to the chosen initial position. In the second row, from the first image to the second is possible to see the transition from skipping while waiting to chasing the ball. The next images shows the agent following the ball. Note that the pink circle outside of the penalty area is *R1*'s kicking target, this target is the central point of the circle that has a radius of 0.25 meters.

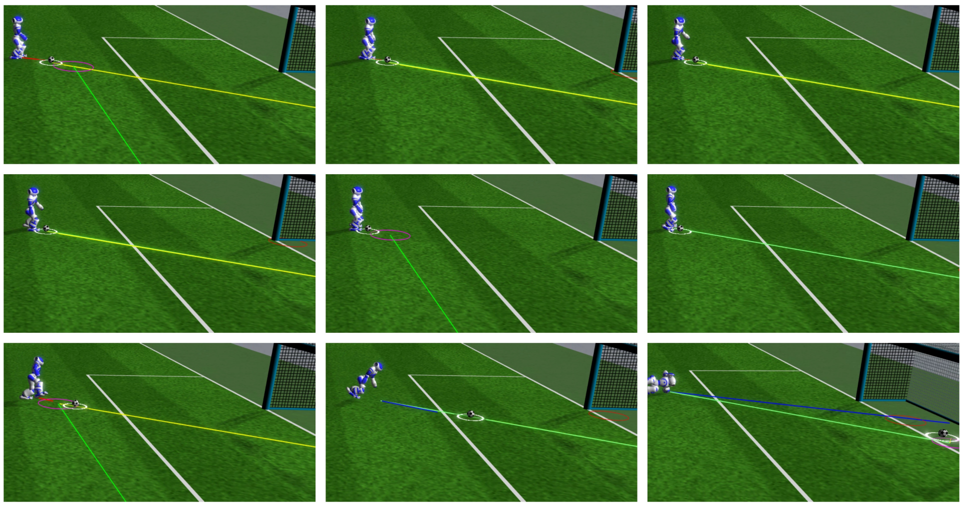


Figure 4.4: *R2* kicks the ball without stopping and scores a goal. Note that the pink circle outside of the penalty area is *R1*'s kicking target, as for the circles inside of the goal, the pink circle (to the right of the second agent's front perspective) is the *R2*'s target with 0.15 meters of radius, and the red circle (to the left) its the previous target outputted by the network.

In summary *R1* uses a kicking skill called *passMediumKick(Vector kickTarget)* that includes the *goToTargetChallenge(...)* skill which is being used until the agent is close enough to kick the ball, in the meantime, *R2* is using *goToTargetChallenge(...)* as well to move to an initial position and keeps this behavior until it starts to pursue the ball (this waiting movement resembles a skipping action since the *float* targets for the initial position are difficult to achieve with exact precision, thus, the agent keeps moving basically in the "same place" while it waits for the pass). Once the pass is executed by the first agent, the second one will chase the sphere using the *kim(Vector kickTarget, float maxerr)* that incorporates the *goToTargetChallenge(...)* to get near the ball, once it is close it kicks the ball to goal without stopping its march.

4.1.3 Episode End Conditions

There are four conditions in every environment that can terminate an episode, an extra one is applied when training involves a goalkeeper:

1. **Goal**: The agents were able to score a goal following the setplay.
2. **Ball kicked outside**: *R2* kicked the ball and it did not enter the goal, going outside of the field.
3. **An agent falls**: If either agent falls before executing its primary task the episode ends. For *R1* if it falls before kicking or when it tries to kick the ball, it falls and the cross does not happen the episode resets. After *R1* has done the pass, this agent can either fall or not, at this stage, no matter what happens, the episode will continue. In the case of the scoring agent other situations can trigger the end of the episode; if *R2* falls while moving to the initial position, or when it is chasing/kicking the ball, or if after kicking, the sphere stops inside the field.
4. **Exceed time-steps**: Every-time the *step(action)* function loops and Syncs, a variable called *timesteps* is incremented, if it gets to 1700 without any other condition being meet the episode still ends.
5. **Defense**: When training with a goalkeeper, if this player can defend the ball there is no need to continue the episode.

4.2 Action Space

The following table shows the actions being used in all of the environments, each action is defined by a continuous interval of *float* values that the network must manipulate. The first two lines in 4.1 set a vector that will be the input of the medium kick function, the next target pair, *a2* and *a3*, are used in the *kim* kick. As for *a4* and *a5*, these define the initial position that the scorer needs to be while waiting for the ball to be moved by *R1*. The final action chosen is used after the pass has been done, by evaluating the ball's speed *R2* has to decide the best timing to start chasing the ball.

Due to the fact that most of the action spaces are targets, all the models trained in this fashion can be applied anywhere on the field. If the throw-in or the corner happens at quadrant 2, seen in figure 4.1, it is enough to use the actions representing y coordinates as negative inputs since this quadrant is a mirror on the x-axis of the quadrant that is being used for training. For example, if the neural network outputs the target (10 ; -1) for *R1* to pass the ball too, then if the scenario occurs in the second quadrant it should become (10 ; 1), for quadrant 3 (-10 ; -1) and if it is at the first one (-10 ; 1). In contrast, the velocity does not need to be altered for any part of the field.

Table 4.1: Action space description with the limits of each action.

Reference	**Action Description**	**Interval**
a0	*R1*'s kick x target	[12.5 ; 15.0]
a1	*R1*'s kick y target	[-3.0 ; 3.0]
a2	*R2*'s kick x target	[15.0 ; 16.0]
a3	*R2*'s kick y target	[-1.05 ; 1.05]
a4	*R2*'s initial x position	[12.0 ; 14.0]
a5	*R2*'s initial y position	[-3.0 ; 3.0]
a6	Ball's Velocity	[0.0 ; 22.0]

As for the intervals chosen, it is important to know that each goal post is positioned at -1.05 and 1.05 in the y-axis and 15 or -15 in the x. Furthermore, the penalty area goes from -3 to 3 on the y-axis and from 13.2 to 15 on the x-axis. Thus, the target for *R1* is defined within the limits of the y values of the penalty area. Since the inside of this zone can sometimes be clustering by having multiple player in it, the x values of the target extend further from the limits of the penalty area in this axis, resulting in *a0* belonging between 12.5 and 15.0. For the kicking target of *R2* since its objective is to put the ball between the posts, the y values are limited by the posts positions, and the x values are defined by the end line coordinate as a minimum value, for the maximum value it was chosen 16 as the limit. Even though this value surpasses the depth of the goals nets (0.6 m), the idea is to see if this helps in kicking with an extra force since the target is further away. Nonetheless, if it is revealed that this is irrelevant or hurtful in training, then the network must learn not to use the values that exceed the net's depth. The targets for the scoring agent initial position were defined considering the interval defined for the passing agent target, as well as, the fact that the kicks implemented in the FCP's team tend to work better when the ball is at a small distance from the agent instead of right in front of them when calling these skills functions using the intended target. As for the ball velocity, this object is either not moving and its speed is zero or is moving and can have several values, the maximum value chosen for the *a6*, 22 m/s, was not the maximum value found during initial experiments with the kicks, however, it was a common value and thus was decided as the superior limit.

4.2.1 Current Implementation

To select a target for an agent that is executing a thorw-in, the team uses a function named *evaluateKickTargets19(...)* which receives several parameters as inputs: a *Vector playerPos*, *float kickMinDist*, *float kickMaxDist*, *float res*, *float fieldMinX*, *float fieldMaxX*, *float fieldMinY*, *float fieldMaxY*,*float weightFlux*,*float weightSafe* and*float weightEasy*. Once all the parameters are defined, a loop will then cycle through all the targets defined between the limits of [*fieldMinX*,*fieldMaxX*] for the x value and between [*fieldMinY*,*fieldMaxY*] for the y. The cycle starts with the minimum value defined by the inferior number in each interval and, at every new iteration, the constant defined by the variable *res* is added until the max value is achieved. Figure 4.5 shows an example of all the possible targets for a random throw-in that has *fieldMinX* = -13, *fieldMaxX* = 13, *fieldMinY* = -8, *fieldMaxY* = 8 and *res* = 1. The goal of this loop is to find the best target to kick the ball to considering the following criteria: how easy is it to get the ball from its current position to the target in question, how safe is the final position of the ball given the rest of team's players' positions relative to the opponents' positions and finally if the target for the ball belongs in a desirable part of the field.

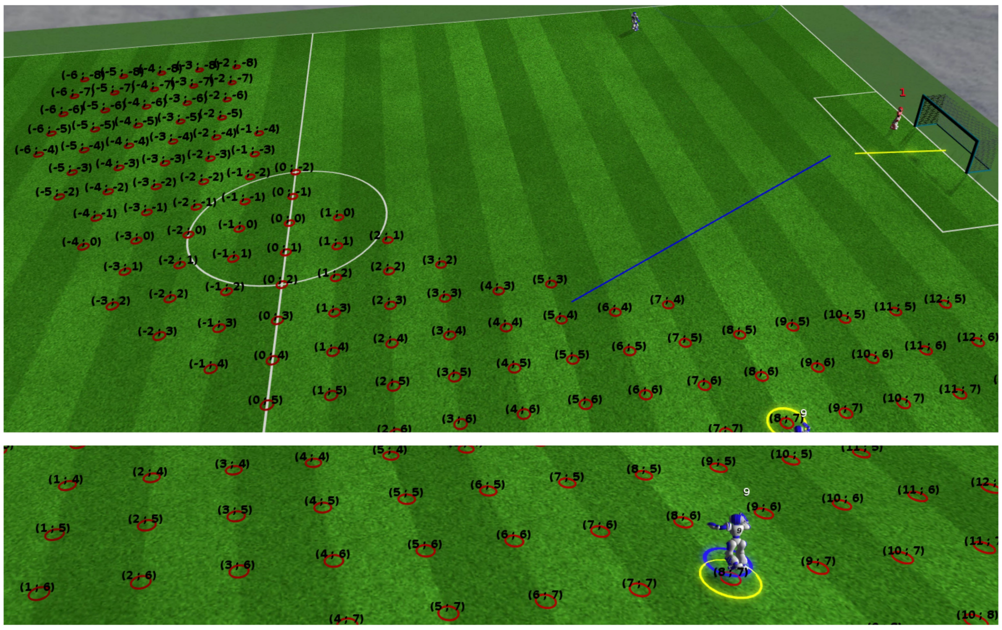


Figure 4.5: In black possible targets taking into account the inputs of the *evaluateKickTargets19(...)* function and at yellow the best target that the strategy outputs.

With the support of three other functions (*evaluateEasy19(...)*, *evaluateSafe19(Vector playerPos, Vector targetPos)* and *evaluateFlux19(Vector playerPos, Vector targetPos)*) a *float* value is outputted by each representing how well its respective criterion is met for a specific target, meaning that the greater the output, the better the condition is being satisfied[3]. Using the *weight* parameters, a criterion can be considered more important than the others since that the best target for the ball's final position is decided by a weighted average of all these factors.

The following figure 4.6 shows the best target changing given the alterations occurring in the game such as the players' positions.

[3]How the calculation of the values for each criterion is obtained is not a critical aspect in the context of this dissertation, and so, it will not be addressed.

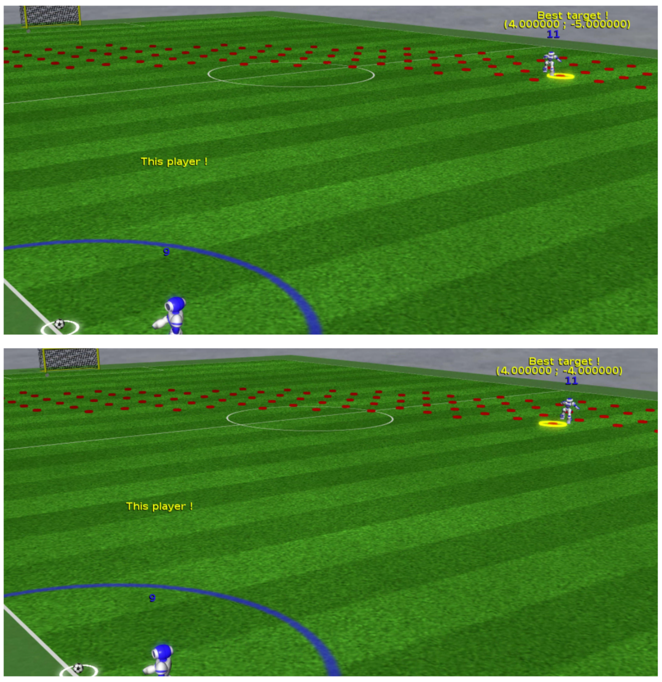


Figure 4.6: Alteration in the *evaluateKickTargets19(...)* function output.

As for the corners, no matter the game's condition, the agent that passes the ball will always kick the ball to the position (10 ; 4) if it is a right corner or to (10 ; -4) if it is a left corner as it can be seen on figure 4.7.

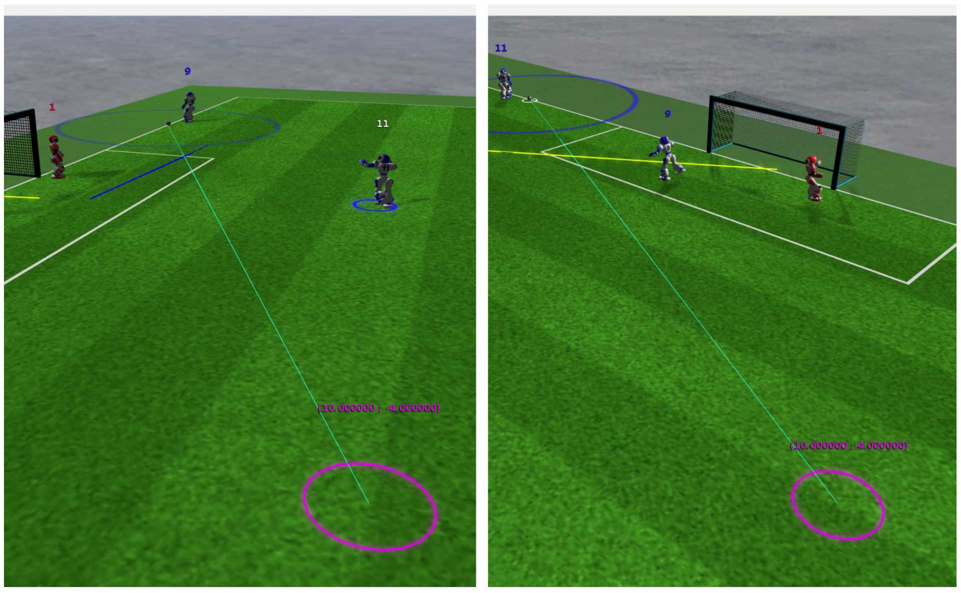


Figure 4.7: The pink circle and annotation represent the crossing agent's target, the left image shows a left corner and the adjacent a right corner.

For both situations, the rest of the agents on the field do not have a determined target position to be in, only general areas considering the ball's position. In terms of scoring goals, the function *simpleCalculateTarget(...)* outputs the best target to kick the ball to considering the agent's and ball's position and a prevision of the ball's final position. The calculations are based on a series of target normalizations taking into account the mentioned parameters.

4.3 PPO2 Default Parameters

The PPO2 implementation in Stable Baselines has several hyperparameters that can be altered to improve training. Since the tuning of these parameters is a complex subject, initial environments used the default values of these hyperparameters. The following table 4.2 shows the main parameters in this algorithm and their respective default values based on the documentation provided by the SB's page [81].

Table 4.2: PPO2 default hyperparameters from the Stable Baselines.

Hyperparameter	**Value**
Policy neural network(s):	2 layers of 64 neurons
float gamma (discount factor)	0.99
int n_steps (number of steps that each episode runs per udpate)	128
float (or *callable*) learning_rate (can also be a function)	0.00025
float vf_coef (value function coefficient for the loss calculation)	0.5
float ent_coef (entropy coefficent for loss calculation)	0.01
float lam (factor for trade-off of bias vs variance for Generalized Advantage Estimator)	0.95
int nminibatches (number of training minibatches per update)	4
int noptepochs (number of epoch for optimization)	4
float (or *callable*) cliprange (clipping parameter, it can be a function)	0.2

4.4 No Goalkeeper

The models trained within this section did not include a goalkeeper to simplified the interpretation of results.

4.4.1 Observation Space

The observations for the environments in this section are described at the table 4.3, the range of these is limited by [-INFINITY,INFINITY], this infinite value is defined by the C++ *math.h* library. The spaces set with "*3*" are meant to represent the observation in each of the axis x, y, and z. On the fourth row of the table, the prevision for the ball's final position is obtained using a function preexisting on the team that calculates a prediction for those coordinates.

Table 4.3: Observation space for the environments trained without a goalkeeper.

Reference	Observation Description	Space
o0	Ball's velocity	3
o1	Ball's acceleration	3
o2	Ball's current position	3
o3	Prevision for ball's final position	3
o4	*R1*'s current position	3
o5	*R2*'s current position	3
o6	*R1*'s distance to ball	1
o7	*R2*'s distance to ball	1

4.4.2 Environment 1

Depending on the type of kick that an agent intends to execute (small versus big distance for example), the joints are positioned in a certain way to better perform the kick, thus, on earlier tests during this dissertation, it was noticed that by varying the kick's target parameters constantly (every time the observations are updated) it normally results on poor execution in case of the *kim* kick, and a too-long preparation time for the *passMedium* that forces the end of the episode by exceeding the limit time-steps defined. As a solution, this first environment saves the network's output regarding each agent's action space pair for the kicking targets 25 syncs after each player starts using their respective kick. Until the 25 syncs are met, the input of the kick functions used is the current output of the NN. Even after the targets are saved both action and observation spaces are constantly updated. As mention before, the kick-ins happen from the sideline (y = 10), having the x limited from 5 to 15, plus, probably the best way to train this game situation would be, for every new episode, to have the ball starting at a random position in the sideline. Furthermore, when the random number resulted in 15, then the throw-in would be altered into a corner situation with initial ball coordinates at (15 ; 6). However, since each agent is launched in its own process, having a random number generator decide the value for the x-axis would result in each pair of agents seeing the ball and consequently *R1* (always begins the episode one meter being the ball) with a different initial position. This messes the observations, since both agents share the same NN, in addition to having the two agents seeing different versions of the episode. To solve this issue, initial experiments were based on a field division that has 11 possible initial positions starting at (5 ; 10) and increasing by 1 meter in the x-axis, ((6 ; 10), (7 ; 10), (8 ; 10), ...) until 15 then, as mention above, the situation turns into a corner resulting in the ball being at (15 ; 6). Figure 4.8 shows in pink the valid initial position for the ball and in red the ones not being considered in any environment in this dissertation.

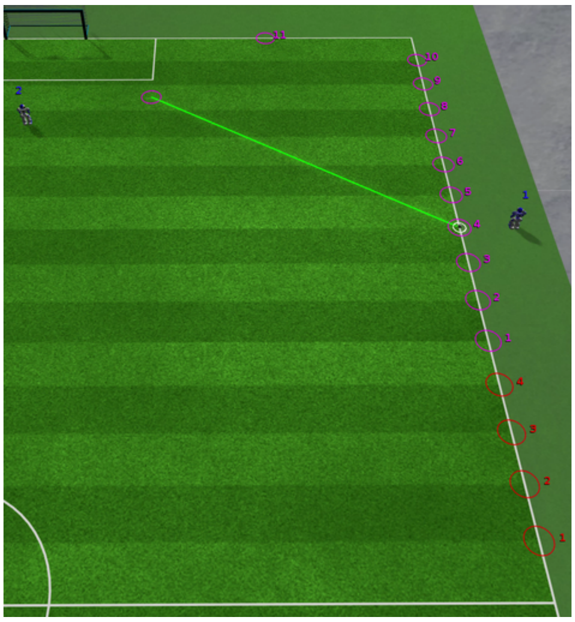


Figure 4.8: Available initial positions for the ball both agents are standing in their initial positions for the fourth ball initial position.

4.4.2.1 Reward

The reward is a *float* value attributed constantly, note that the variable *distBallToR2* represents the distance from the ball's current position to the agent *R*2, *distBallToCenter* the distance from the ball's current position to the coordinates (15 ; 0) and *rewardaux* is an auxiliary variable, the reward is shaped as:

$$R(t) = \begin{cases} \frac{1}{(distBallToR2\ +\ 0.25)} & \text{, ball is at a distance superior to 0.55 meters from } R2 \\ 25 + rewardaux + \frac{1}{(distBallToCenter\ +\ 0.25)} & \text{, goal scored} \\ rewardaux + \frac{1}{(distBallToCenter\ +\ 0.25)} & \text{, else} \end{cases} \tag{4.1}$$

Once each agent sees that the ball is at distance inferior to 0.55m from *R*2, the reward at that time-step ($R(t) = \frac{1}{(distBallToR2\ +\ 0.25)}$) is saved in the variable *rewardaux* and kept until the end of the episode.

4.4.2.2 Baseline Implementation

Before proceeding to the results of applying the RL algorithm to this environment, a baseline implementation was constructed and tested using various tools from the teams code to serve as the base results that the trained setplay should overcome. The baseline includes the two agents mentioned before. *R1* starts behind the ball and will wait 4 seconds before kicking it, similar to what happens in the trainable environments. During that time the *evaluateKickTargets19(...)* is used to select the kicking target of this agent (instead of using the action space pair). As for *R2*, this agent begins the episode at (11.5 ; 0) and then uses the *goToTargetChallenge(...)* to move to a random initial position (*Version 1*) or stays in the same place doing skipping (*Version 2*, both alternatives were tested). Once the 4 seconds are over, the first agent executes the pass with the medium kick using the target outputted by the *evaluateKickTargets19(...)* function. As soon as the second agent sees that the ball as moved it advances to the kicking stage using the *kim(Vector kickTarget, float maxerr)* skill with it's target decided by the *simpleCalculateTarget(...)* function. **This implementation can be defined as a scenario equal to the trainable environment described before without the elements related to reinforcement learning**. In addition, considering the alterations that can occur from one environment to another, mentioned in the beginning of this section (4.1), this baseline is comparable with most of the models trained without a goalkeeper. For the sake of fairness and approximation of the scenarios, the episodes ending conditions from the environments are applied here as well. This baseline was tested throughout the course of 330 episodes passing through the 11 possible initial position of the ball, in total each position was tested in 30 episodes. The average for each position considering the total sum of goals can be seen in table 4.4, as well a percentage of scored the goals in the 330 episodes. Each row on the table represents a version of the implementation considering the *R2*'s initial position.

Table 4.4: Performance results from the baseline implementation.

	Goals												
Baseline	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**Mean Total**	**Achievement**
Version 1	8	6	9	8	5	7	7	4	4	7	3	68/11 = 6	21%
Version 2	4	4	6	7	8	6	3	2	1	0	3	44/11 = 4	13%

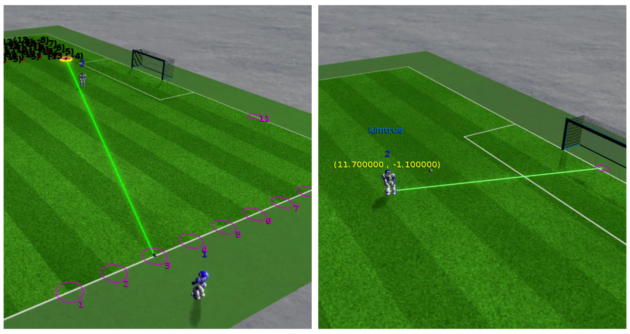


Figure 4.9: Baseline implementation example.

4.4.2.3 Results

Following, to get the results of training in Environment 1, the models obtained in this environment were applied in a total of 330 episodes ran in separated SimSpark instances having 110 episodes per instance to speed the process. In total, each initial position of the ball goes through 30 episodes similar to the baseline implementation. The TensorBoard tool provides a graphic called *Episode Reward* that includes the cumulative reward of all the episodes from all the environments used to train a model. The graphs, presented in the figure 4.10, have a high level of smoothness obtained using an exponential moving average.

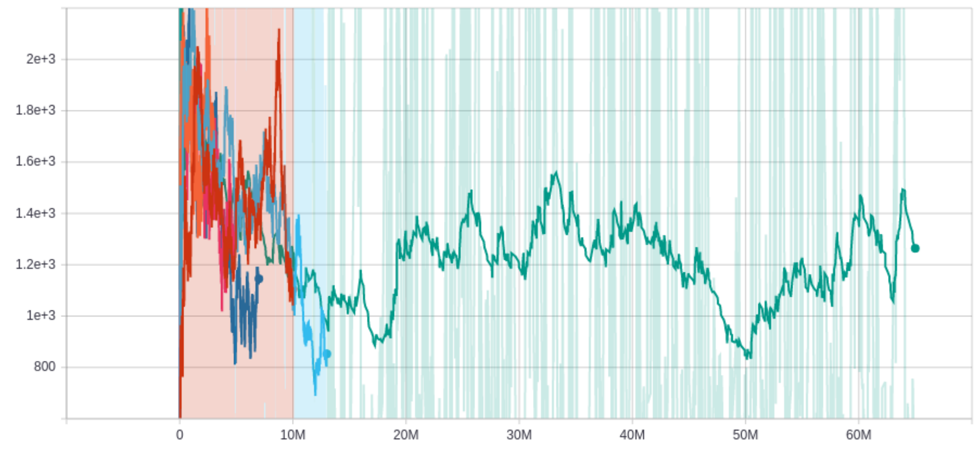
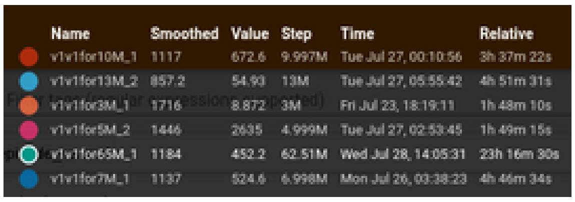


Figure 4.10: Plot of the cumulative rewards of several models trained in Environment 1. Smoothing = 0.980.



	Name	Smoothed	Value	Step	Time	Relative
●	v1v1for10M_1	1117	672.6	9.997M	Tue Jul 27, 00:10:56	3h 37m 22s
●	v1v1for13M_2	857.2	54.93	13M	Tue Jul 27, 05:55:42	4h 51m 31s
●	v1v1for3M_1	1716	8.872	3M	Fri Jul 23, 18:19:11	1h 48m 10s
●	v1v1for5M_2	1446	2635	4.999M	Tue Jul 27, 02:53:45	1h 49m 15s
●	v1v1for65M_1	1184	452.2	62.51M	Wed Jul 28, 14:05:31	23h 16m 30s
●	v1v1for7M_1	1137	524.6	6.998M	Mon Jul 26, 03:38:23	4h 46m 34s

Figure 4.11: Some details relative to the plots above.

To give a general perspective of how effective the training was, the following table 4.5 shows the results of applying some of the trained models. The first column has the name of the model followed by how many goals were scored in each ball initial position as the sum of the values in that position from each SimSpark instance, having 30's in every single one of these eleven columns would mean that the perfect model was achieved. The "*Mean Total*" column is the average of the sum of goals from all 11 positions, the next two columns "*Mean of Inst*" and "*Var of Inst*" represent the mean of instances and the variance of the instances respectively. The mean is the simple arithmetic average of the total of goals and thus is calculated by dividing this value by

3, the variance was calculated as $\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}$, where σ^2 is the variance, n is the number of instances (3), μ is the mean and x_i is the total of goals in all the position from each instance (values not present in the table). The "R1 *fall*" column exhibits how many times the passing agent fell before kicking the ball, and finally, a percentage of the results is calculated by dividing the total of scored goals by the 330 episodes.

Table 4.5: Performance results from some of the models trained in Environment 1.

	Goals															
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**Mean Total**	**Mean of Inst**	**Var of Inst**	**R1 fall**	**Achie-vement**
v1v1 for7M	12	10	9	15	10	12	11	11	10	9	8	117/11 = 11	39	8	10	35%
v1v1 for13M	15	13	11	7	9	14	10	17	9	22	9	136/11 = 12	45	3	11	41%
v1v1 for65M	11	10	6	8	7	5	10	10	8	10	4	89/11 = 8	30	14	11	27%

4.4.2.4 Conclusions

Comparing the achievement from the baseline implementation table 4.4 with the table above, 4.5, it can be seen that even the worst model scored more goals than the base implementation resulting in a superior percentage of achievement. By visualizing the trained models in practice using Roboviz and having the table results, it was concluded that in the worst model, *v1v1for65M*, the passing agent is kicking the ball too close to the end-line. Either making it go out of the field and as a consequence terminating the episode or, getting the ball really close to the end-line, without leaving the field, but limiting *R2*'s angle to kick the ball and making it harder to score. As for the best model, *v1v1for13M*, when *R1*'s cross goes behind its teammate's back, it is difficult for this agent to follow the ball without losing its balance and falling. In addition, the kick itself from *R2* failed a few times, possibly from deciding the target to close to the ball resulting in a bad positioning from this agent to kick the ball. As a final notice, a problem found in all models is the trajectory of the pass from the first agent to the second, after executing the pass, the ball might impact with *R2* at a high velocity making this agent fall, this occurs due to poor target selection by *R1* as well as bad positioning from *R2*.

4.4.3 Environment 2

The following environment is similar to 4.4.2 the only difference being the reward.

4.4.3.1 Reward

Here, instead of the inverse distance, the negative distance from the ball's current position to the scoring agent is used, plus, the distance from the ball to the center was eliminated.

$$R(t) = \begin{cases} -distBallToR2 & \text{, ball is at a distance superior to 0.55 meters from } R2 \\ 3 + rewardaux & \text{, goal scored} \\ rewardaux & \text{, else} \end{cases} \tag{4.2}$$

Once each agent sees that the ball is at distance inferior to 0.55m from $R2$, the reward at that time-step ($R(t) = -distBallToR2$) is saved in the variable *rewardaux* and kept until the end of the episode.

4.4.3.2 Results

The figure below has a higher level of smoothness given that with more models being displayed the bigger the cluster and amount of data present in these plots, to facilitate the visualization, the graphs were smoothed as much as possible. Note that only part of the models trained in each environment are displayed. Otherwise, similar information or data that does not bring anything new to the analyzes would be presented.

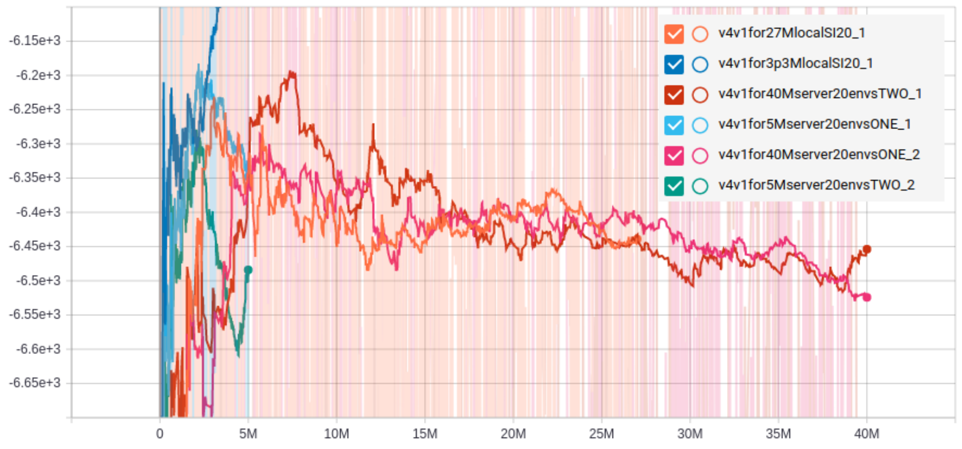


Figure 4.12: Plot of the cumulative rewards of several models trained in Environment 2. Smoothing = 0.999.

Table 4.6: Performance results from some of the models trained in Environment 2.

	Goals															
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**Mean Total**	**Mean of Inst**	**Var of Inst**	**R1 fall**	**Achie-vement**
v4v1 for3p3M local SI20	11	14	14	13	10	11	8	9	10	15	13	128/11 = 12	43	10	8	39%
v4v1 for5M server 20envs ONE	12	15	15	15	17	16	18	17	18	19	13	175/11 = 16	58	2	10	53%
v4v1 for40M server 20envs TWO	12	11	12	12	13	10	9	11	13	10	14	127/11 = 12	42	24	9	38%

4.4.3.3 Conclusions

It could be expected by looking at figure 4.12 that the *v4v1for3p3MlocalSI20* model would have the best results given that its plot shows a rapid increase in the value of the reward through the episodes. However, this is not the case. What could have happened is that during training this model had a series of "lucky" episode setups that privileged the outcome result or since the overall reward will have a negative value if the episodes end earlier then the final cumulative reward will be smaller giving the illusion of good training. As for the best model, it still presents some issues with poor kicking, nonetheless, possibly due to better passing and position this problem happens less as well as bad ball trajectory during the pass (less common but still happens).

4.4.4 Environment 3

As a way to perform better target selection, this environment was envisioned to use the neural network only when necessary to prevent noisy updates, i.e, by looking at the agent's number 1 kick target and the agent's number 2 velocity threshold, these actions are only used momentarily. Nevertheless, they are being influenced during the full duration of each episode. To solve this, it was decided to use communication to signal both agents to stay in the *step(action)* at strategic moments. Taking this into account, the episodes are now divided into five focal moments:

1. During the initial four seconds of the episode the standard procedure occurs with the network always updating its values taking into account the current observations. The major

difference at this point is that *R2* is also not moving contrary to what it was doing in the previous environments.

2. Once the four seconds have passed, the code will stop moving to the *observe()* function, at this point no communication is yet needed since the time-steps count is equal in both agents. *R2* then moves to the initial position last outputted by the NN and its teammate precedes to the kick with the last target given by the network as well.

3. After executing the kick, once *R1* sees that the ball has moved from its initial position, it will tell the other agent that by the next cycle, it should return to updating the action values.

4. Next, when the ball is close to *R2* (less then 1 meter) and its x coordinate is superior to the x of the agent, then *R2* communicates to its teammate to, once again, stay looping the *step(action)* function.

5. Finally any possible outcome that terminates the episode (goal, ball out or, agents falling) is communicated by *R2* to *R1* so that both have the *done* variable set as *true* at the same time and proceed to the next episode simultaneously. Before re-starting a new episode in *reset()*, the code goes through the *observe()* on last time, and so the network receives a final reward taking into account the outcome of that training session.

Communication is needed to stop/precede to functions given that the observations that each agent receives are not exactly the same, the further away from an element an agent is, the noisier the observations become. Using moment 3 as an example, *R1* could see the ball moving from it's initial position before its teammate that is further away, leading this first agent to advance to the *observe()* function without the other process doing the same, resulting in all training freezing.

As for the ball trajectory problem, to try and overcome this, alterations to the reward were made seen at 4.4.4.2 .

4.4.4.1 Communication

As a way to have fast communication without interfering with the server's game definitions, external communication was used based on *named pipes*. Given that, each pair of agents used for training have a unique server and monitor ports, the *pipes* are created at the beginning of each training model and are named as the monitor port that both agents are connected to, allowing each pair to not interfere with the rest of the players that are training in parallel. The signaling is done using flags, each flag is a number that signals the other agent in what "moment" of the episode is supposed to be in.

4.4.4.2 Reward

The reward for this environment was designed to avoid the second agent falling so many times and thus, the variable *ally2z* represents the height of the player *R2* (maximum height value is 0.4).

$$R(t) = \begin{cases} 5 & \text{, goal scored} \\ ally2z & \text{, else} \end{cases} \tag{4.3}$$

4.4.4.3 Results

The results of the models from this environment will be presented in a similar way to the previous ones, including alternative versions of this environment with slight changes that will be duly pointed out.

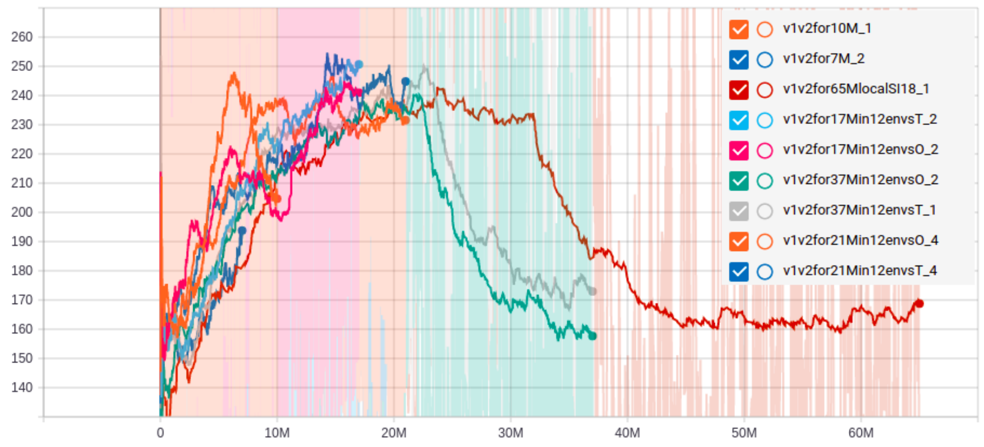


Figure 4.13: Plot of the cumulative rewards of several models trained in Environment 3. Smoothing = 0.991.

Table 4.7: Performance results from some of the models trained in Environment 3.

	Goals															
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**Mean Total**	**Mean of Inst**	**Var of Inst**	**R1 fall**	**Achie-vement**
v1v2 for7M	8	14	14	10	9	11	8	9	10	9	8	110/11 = 10	37	7	8	33%
v1v2 for17M in12 envsT	5	4	10	8	8	4	5	8	6	13	5	76/11 = 7	25	1	20	23%
v1v2 for21M in12 envsT	9	10	8	10	12	13	7	5	6	7	4	91/11 = 8	30	16	18	28%

Although the results are not famous, since this environment was already based on communication, it was used to test if training with the ball's initial position set as a random instead of the predefined 11 position brings better results. This is done by altering the environment so that *R1* sends its teammate the ball's initial position after having it randomly generated. The following figure and table show a few models trained with this setup. By comparing the table's 4.8 first two rows, with the previous results it can be seen that this alternative does not show improvements.

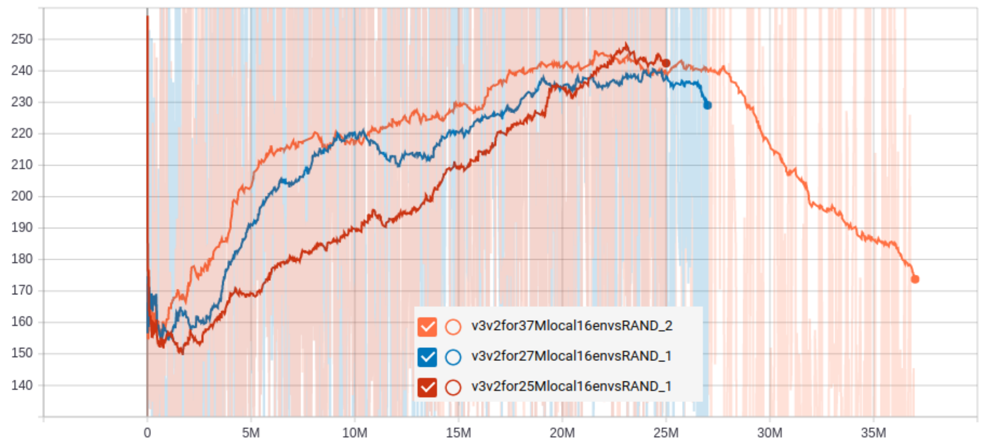


Figure 4.14: Plot of the cumulative rewards of several models trained in Environment 3 with random ball initial position. Smoothing = 0.994.

Note that the *v4v1for5Mserver20envsONE* on the last row from table 4.8 is a model trained in Environment 2 from section 4.4.3. To test the quality of the best model yet, the environment in the 4.4.3 section was altered to have the ball's initial position begin at random coordinates. Next, the model was tested in this new scenario and results are displayed at the following table. It was

Table 4.8: Performance results from some of the models trained in Environment 3 with random initial ball position, plus re-evaluation of the best current model.

	Goals				
Model	**Total**	**Mean per Inst**	**Var of Inst**	**R1 fall**	**Achievement**
v3v2 for25M local 16env RAND	60	20	3	11	18%
v3v2 for37M local RAND	80	27	4	15	24%
*v4v1 for5M server 20envs ONE**	132	44	1	9	40%

expected that the quality of the results decreased, nevertheless, this model still presents as the most accurate even in an alternative environment to the one it was trained with. Lastly, since the original 11 initial position for the ball were not used, the "Mean Total" is replace with the total sum only.

A final environment was tested where the neural network is only used in moment number 1 (recalling the enumeration made at the beginning of the Environment 3 description). In short, the network is updated for 200 time-steps then it is not used for the rest of the episode having all the actions being decided in the beginning and receiving the final feedback once the episode has finished.

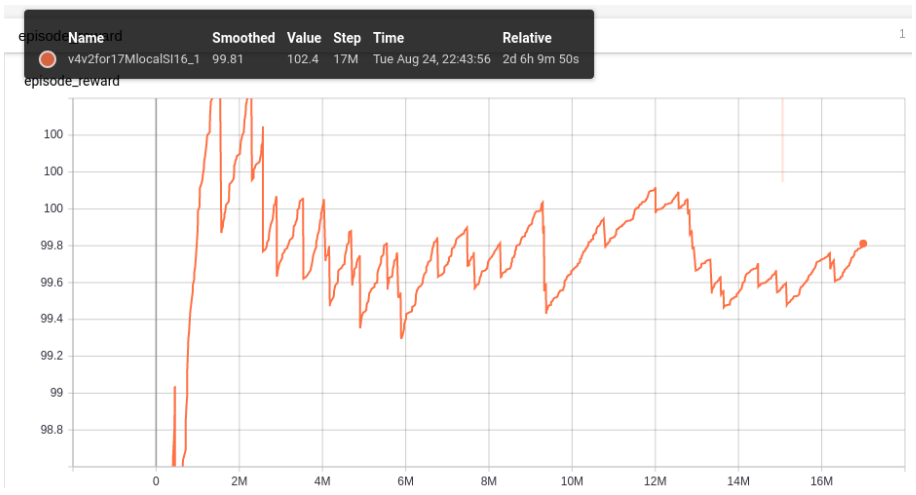
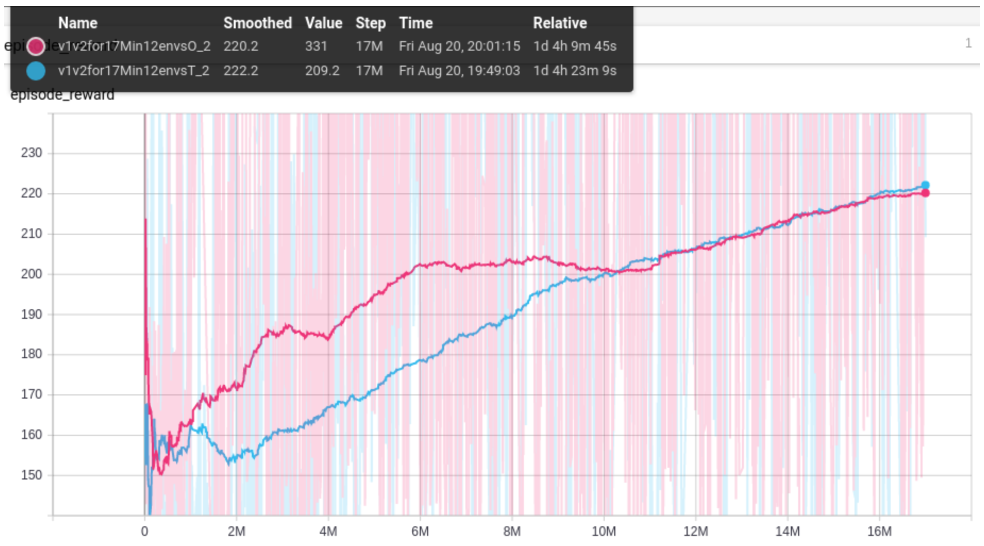


Figure 4.15: Plot of the cumulative reward of a model trained in Environment 3 only calling the NN once.

Table 4.9: Performance results from a model trained in Environment 3 using the NN once.

	Goals															
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**Mean Total**	**Mean of Inst**	**Var of Inst**	**R1 fall**	**Achie-vement**
v4v2 for17M local SI16	10	8	13	7	11	7	9	7	10	10	7	99/11 = 9	33	3	10	30%

In spite of this alternative showing promising results, it is important to notice at figure 4.15 the time it took to train 17M time-steps, compering with Environment's 1 figure 4.11, that revels that to train for 65M takes 23 hours and 16 minutes in contrast with the 2 days and 6 hours needed here. For perspective reasons, the figure below shows how much time it takes to run Environment 3 with the default setup for 17M steps as well.



	Name	Smoothed	Value	Step	Time	Relative
●	v1v2for17Min12envsO_2	220.2	331	17M	Fri Aug 20, 20:01:15	1d 4h 9m 45s
●	v1v2for17Min12envsT_2	222.2	209.2	17M	Fri Aug 20, 19:49:03	1d 4h 23m 9s

Figure 4.16: Example of two models trained in Environment 3 with timestamps.

4.4.4.4 Conclusions

With this environment is possible to conclude that using less the neural network has an increase in training time. The use of communication allowed to see that having the ball initially in a random position doesn't necessarily improve the results. Nonetheless, the environments trained with the predefined initial position will lose the quality of results when used in this alternative, although, it could be beneficial to train with predefined initial positions and then, retrain the same model with random coordinates. This way it is guaranteed that the agents have a general perception of the field but also that they do not get restricted to specific areas. Due to time consumption as well as lack of improving results, once the environments get more challenging with the addition of a keeper, the use of the NN throughout all the episode's duration will be considered instead of the scenarios present in this third environment.

4.5 With Goalkeeper

The models trained within this section all include a goalkeeper, and so the observation space is going to be increased to follow this change. It would be reasonable to retrain the best model achieved without the keeper now in the presence of one, however, that is not possible, given that both the action and observation spaces must keep the same size as the original model for retraining to be doable.

4.5.1 Observation Space

The action space will stay the same and, as mentioned, the observations will be altered by adding both the position of the goalkeeper as well as its distance to the ball. This is done to help the training since the episode terminates if the keeper defends the ball (when the sphere distance to this opponent is less than 0.5 m and the ball is in front of them).

Table 4.10: Observation space for the environments trained with a goalkeeper.

Reference	**Observation Description**	**Space**
o0	Ball's velocity	3
o1	Ball's acceleration	3
o2	Ball's current position	3
o3	Prevision for ball's final position	3
o4	*R1*'s current position	3
o5	*R2*'s current position	3
o6	*R1*'s distance to ball	1
o7	*R2*'s distance to ball	1
o8	Goalkeeper's current position	3
o9	Goalkeeper's distance to ball	1

4.5.2 Keeper Description

The goalkeeper utilized is the FC Portugal team's keeper, this player has its own behaviors being programmed to defend the team's goal and not be a "mindless" agent, this implementation is the one used in 3D simulation competitions.

This agent will be launched using the library "*subprocess*" available in Python in the same server port as each pair of trainable agents, even though this is the team's goalkeeper when launching this agent in the server, it is defined with a different team name (by default is "FC Portugal", here it is named "Opp"), resulting in the other agents seeing the keeper as an opponent.

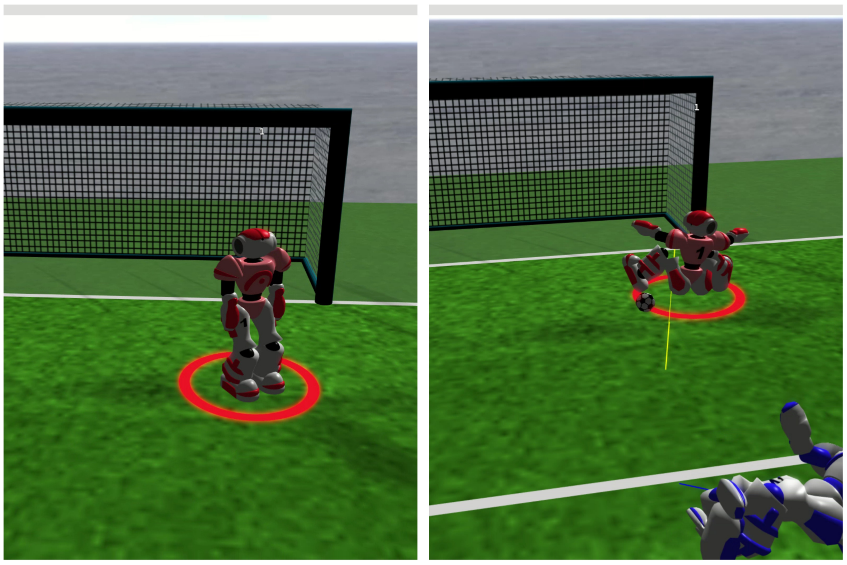


Figure 4.17: Opponent's goalkeeper, normal stand in the left and a defense example on the right.

4.5.3 Environment 1

The first environment tested was based on the environment from where the best model appeared, i.e, Environment 2 at section 4.4.3 with the extra condition that if the keeper defends the ball the episode is over.

4.5.3.1 Reward

The reward shaping is kept the same as the one in section 4.4.3.1 and will be for all the models with a goalkeeper, thus, the following environments will not have a reward description since it is the same in all of them.

$$R(t) = \begin{cases} -distBallToR2 & \text{, ball is at a distance superior to 0.55 meters from } R2 \\ 3 + rewardaux & \text{, goal scored} \\ rewardaux & \text{, else} \end{cases} \tag{4.4}$$

At the first moment that the ball is at distance inferior to 0.55m, the reward at that time-step (-*distBallToR2*) is saved in the variable *rewardaux* and kept until the end of the episode.

4.5.3.2 Results

The results presentation for these models maintains the same format as before.

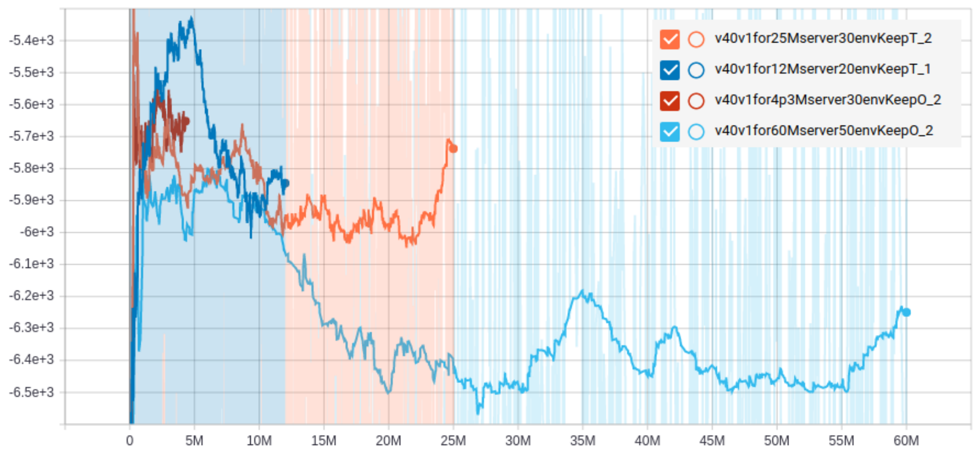


Figure 4.18: Plot of the cumulative rewards of several models trained in Environment 1 with a goalkeeper. Smoothing = 0.995.

Table 4.11: Performance results from some of the models trained in Environment 1 with a goalkeeper.

	Goals																
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**Mean Total**	**Mean of Inst**	**Var of Inst**	**R1 fall**	**Def**	**Achie-vement**
v40v1 for4p3M server 30env KeepO	8	8	10	9	9	13	11	11	14	10	10	113/11 = 10	38	6	9	45	34%
v40v1 for12M server 20env KeepT	9	10	8	17	8	6	10	10	12	12	15	118/11 = 11	39	34	14	53	36%
v40v1 for25M server 30env KeepT	8	5	9	9	12	9	8	13	11	9	6	99/11 = 9	33	18	18	54	30%
v40v1 for60M server 50env KeepO	9	8	9	3	13	5	4	9	9	14	5	88/11 = 8	29	8	11	32	27%

4.5.3.3 Conclusions

Once again, by visualizing the models in practice it was possible to conclude the source of some of the failures in this environment. The fact that the target for the second agent is decided 25 syncs after entering the kicking stage (that includes the movement that precedes it), results in hasty decisions that are not taking into account the dynamic obstacle that is the goalkeeper. The latter's active behavior can make it unlikely that the target chosen is still as good if some time passes since it was selected until it is executed. Nonetheless, the problems that exist in the original environment that has no keeper are inherited in these models as well. Finally, it can also be concluded that with more time-steps trained the lesser the defenses and falls, however, the total of goals is worst. This is due to the fact that both of the agent's targets decrease in quality, the passing agent is kicking the ball far too distant from the keeper making it harder to score, moreover, even if the pass lands the ball in an advantageous position, the scoring agent is kicking it as much as possible away from the opponent's position resulting in the ball going out not counting as a defense.

4.5.4 Environment 2

In an attempt to improve the accuracy of when a target should be locked before kicking, the following environment introduces some alterations to the usage of the kick's actions space. Similar to previous environments the *R2*'s target will be updated with the current network output after a certain amount of syncs, in this case, 50 syncs. However, the blocking of this output does not occur right away, this will only happen once a set of conditions occur, until then, every 50 syncs the target is updated in the kick function. The conditions that need to exist in order to lock the target are: the ball's x coordinate being bigger than *R2*'s (implying that it is in front of them), the ball's distance from this agent is under 1.4 meters and with a velocity below 0.8 m/s and finally, the orientation of the second agent is, as an absolute value, smaller then 75° (implying that this agent is facing the goal and not on it is back). Only when all of these circumstances are true is the last output from the NN saved and used as input for the kick in motion until the end of the episode.

4.5.4.1 Results

Analogous to the previous environments, what follows is a presentation of the results using a table as well as the "*Episode Reward*" plots for the models, in addition, results to alternative versions are also showed and explained.

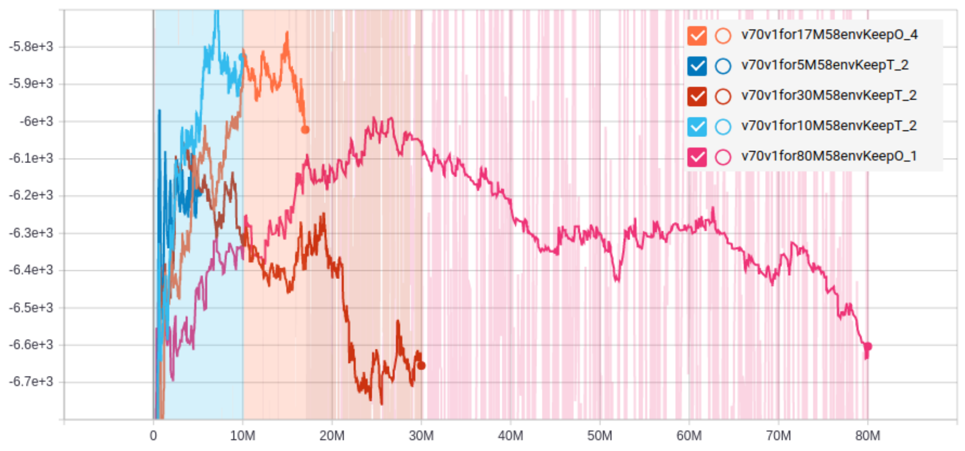


Figure 4.19: Plot of the cumulative rewards of several models trained in Environment 2 with a goalkeeper. Smoothing = 0.995.

Table 4.12: Performance results from some of the models trained in Environment 2 with a goalkeeper.

	Goals																
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**Mean Total**	**Mean of Inst**	**Var of Inst**	**R1 fall**	**Def**	**Achie-vement**
v70v2 for5M 58env KeepT	9	11	10	16	13	10	11	9	10	13	9	121/11 = 11	40	18	16	39	37%
v70v2 for17M 58env KeepT	8	10	7	6	10	9	8	13	12	17	11	111/11 = 10	37	5	19	32	34%
v70v2 for30M 58env KeepT	5	7	8	7	9	8	11	10	9	10	9	93/11 = 8	31	18	14	40	28%
v70v2 for80M 58env KeepO	9	10	8	9	8	2	6	9	6	15	5	87/11 = 8	29	8	16	36	26%

Given that the results presented do not show significant improvements it was decided to stop trying to adapt the environment itself and try to adjust the training algorithm's hyperparameters. As mention before, tuning these variables is not a trivial task, nonetheless, the following models had the PPO parameters suffering slight alterations. In figure 4.20 and at table 4.13 it can be seen the models that had their policy neural network altered to have two hidden layers, one with 64 neurons and the other with 128, contrary to the original two layers of 64 neurons each showed at table 4.2.

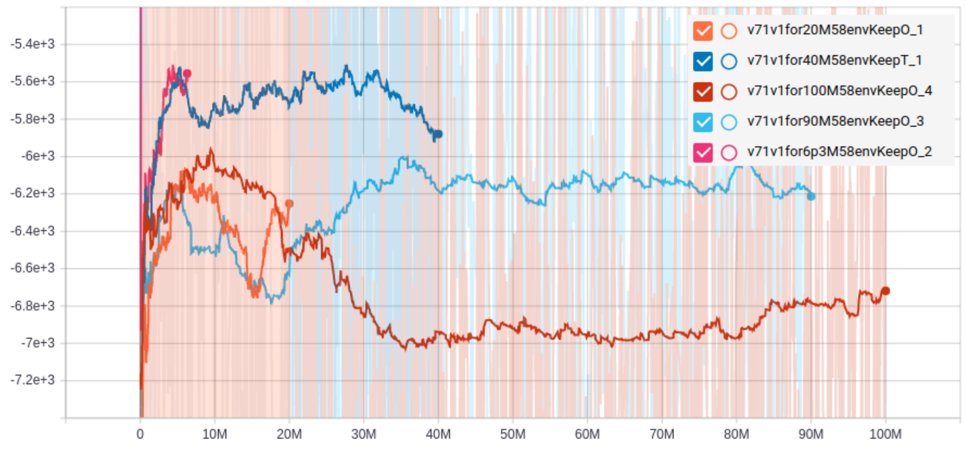


Figure 4.20: Plot of the cumulative rewards of several models trained in Environment 2 with a goalkeeper and a custom policy network. Smoothing = 0.995.

Table 4.13: Performance results from some of the models trained in Environment 2 with a goalkeeper and a custom policy network.

	Goals																
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**Mean Total**	**Mean of Inst**	**Var of Inst**	**R1 fall**	**Def**	**Achievement**
v71v2 for 6p3M 58env KeepO	13	11	12	9	12	9	10	9	15	11	13	124/11 = 11	41	2	12	24	38%
v71v2 for20M 58env KeepO	8	7	5	9	8	11	12	7	9	12	10	98/11 = 9	33	6	25	32	30%
v71v2 for90M 58env KeepO	9	6	8	11	5	7	6	4	10	8	5	79/11 = 7	26	17	20	54	24%

It was also tested changing only the *lam* parameter from 0.95 to 0.99. It is common to have this variable between 0.9 and 1 and the next set of models were meant to show the difference, if any, of altering *lam* to be practically equal to 1. As a way to see the effect of this alteration, the policy neural network for these models was the default version present at the PPO table instead of the custom presented above.

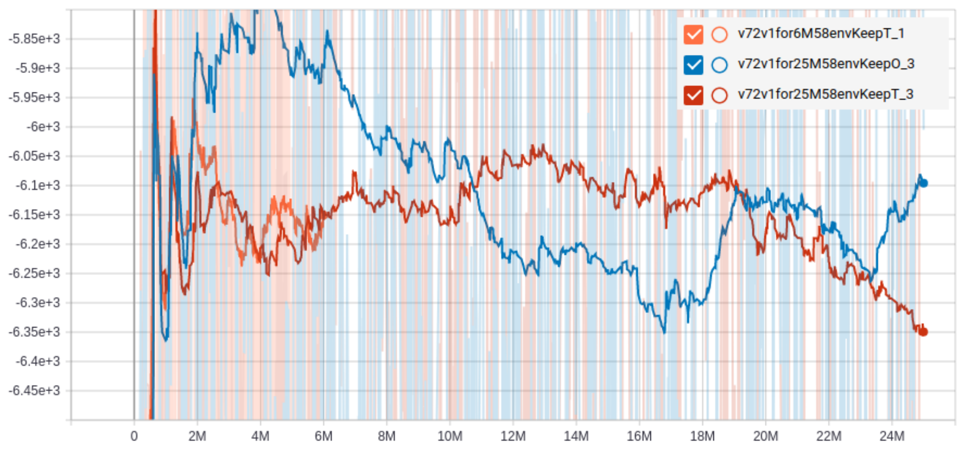


Figure 4.21: Plot of the cumulative rewards of several models trained in Environment 2 with a goalkeeper and *lam* = 0.99. Smoothing = 0.997.

Table 4.14: Performance results from some of the models trained in Environment 2 with a goalkeeper and *lam* = 0.99.

	Goals																
Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**Mean Total**	**Mean of Inst**	**Var of Inst**	**R1 fall**	**Def**	**Achie-vement**
v72v2 for6M 58env KeepT	6	8	6	6	14	12	8	14	9	20	8	111/11 = 10	37	7	13	35	34%
v72v2 for25M 58env KeepO	12	10	7	8	17	8	9	11	11	9	8	110/11 = 10	37	8	13	43	33%

4.5.4.2 Conclusions

The first conclusion from this environment is that the alterations to the target locking did not result in more goals, in addition, it did not resolve the problem with the failing kick as intended. Nonetheless, it makes for a more realistic application and gives results as good as the initial scenario. With this in mind, testing was followed by the manipulation of PPO's parameters. Once again the results did not show a significant improvement. However, by comparing the best model that has a different policy network with the best models from the other two tables, even though the achievement only improved slightly the variance between the SimSpark instances is significantly lower, therefore, it can be assumed that the *v71v2for6p3M58envKeepO* is a more trusting model, and so, its parameters' characteristics should be taken into consideration. The last set of models at table 4.14 do not show promising results, and so the *lam* should be kept with its original value.

4.5.5 Environment 3

Lastly, as mention before, it could be interesting to retrain a model that was originally trained with predefined initial positions for the ball now with a random set of initial coordinates instead. With this in mind the retrain of the *v71v2for6p3M58envKeepO* model is presented and, in the following table, the comparison of the results of this new model with the original used in a scenario where the initial position is random as well. A new model solely trained with arbitrary initial position is also detailed.

4.5.5.1 Baseline Implementation

Similar to what was done in the first Environment for the models without a keeper, an implementation was tested using tools from the team's code and without applying RL elements. This baseline matches most of the layout of the original *Version 1* with *R2*'s initial position set at random. However, this implementation has a goalkeeper present and the ball is also starting the episode in a random position. The usual 330 episodes will test this implementation and the results are displayed in the table below.

Table 4.15: Performance results from the baseline implementation with a goalkeeper and random initial position for the ball.

	Goals					
Baseline	**Total**	**R1 fall**	**R2 fall**	**Def**	**Ball Out**	**Achievement**
Version 1	55	19	200	40	16	17%

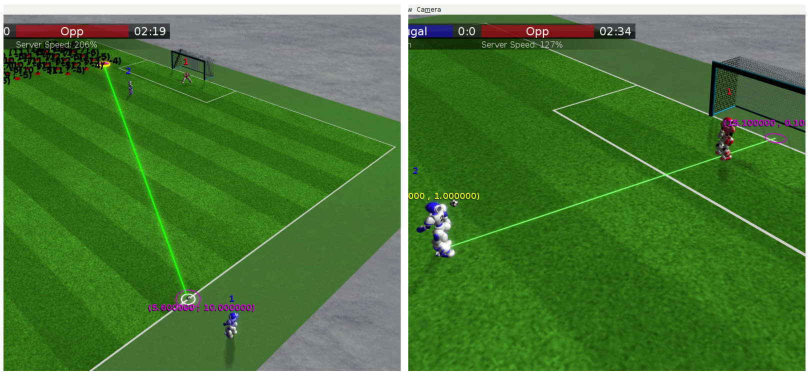


Figure 4.22: Baseline implementation example with a goalkeeper, the pink values above the pink targets are the ball's initial random coordinates and the*R2*'s target.

4.5.5.2 Results

The last table and graph contain information regarding models that were trained in different environments, as explained above, for a final comparison.

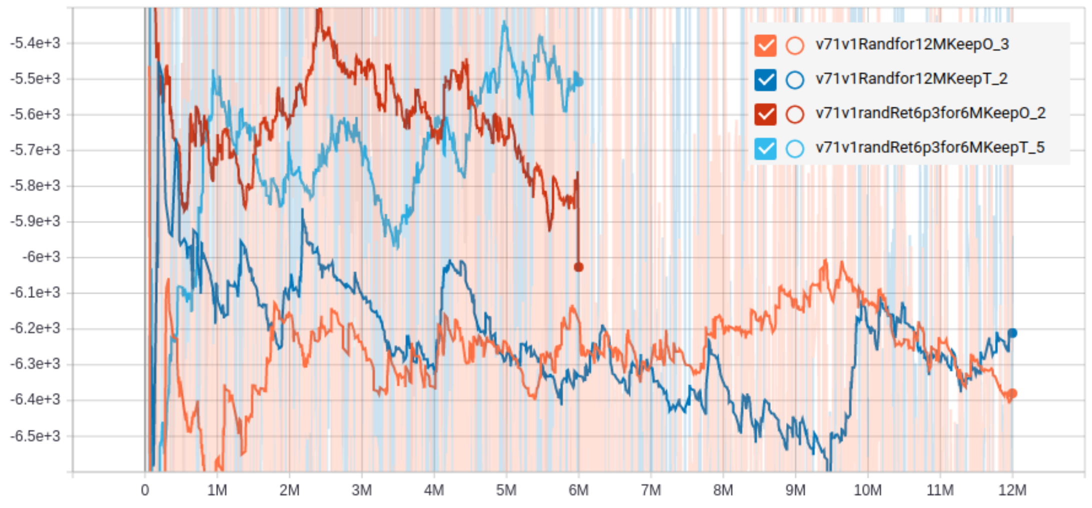


Figure 4.23: Plot of the cumulative rewards of several models trained in Environment 3. Smoothing = 0.990.

The table 4.16 includes extra information relative to the episodes' outcome including, how many times the ball was kicked outside, as well as, the number of times that the scoring agent fell and it was not a goal or ball outside. This can happen when this agent is attempting to kick the ball and falls, or does a bad kick that does not get the ball close enough to the keeper to count as a defense. As a final trigger, the *R*2's movements can make them lose balance and fall or it can lose it by being hit with the ball at high velocity.

Table 4.16: Performance results from some of the models trained in Environment 3 with a goalkeeper.

	Goals							
Model	**Total**	**Mean of Inst**	**Var of Inst**	**R1 fall**	**R2 fall**	**Def**	**Ball Out**	**Achievement**
v71v2Randfor12M KeepTWO	131	44	3	7	135	23	34	40%
v71v2randRet6p3 for6MKeepO	98	33	17	15	121	52	44	30%
*v71v2for6p3M 58envKeepO**	94	31	1	17	134	37	48	28%

4.5.5.3 Conclusions

Although the first model in the table above, *v71v2Randfor12MKeepTWO*, appears in figure 4.23 to not have had a successful training, it ends up being the model with the best results when training with a goalkeeper. This model was trained from scratch having the ball's initial position with random placement, from the table 4.16 is visible that the number of times that the agents scored goals is similar to the number of times that the second agent fell, this is supported with a visual interpretation that reveled various crosses from *R1* directly hitting its teammate making it fall and ending the episode and many passes happening though the back of *R2* making it also fall from trying to chase the ball and having to do a "brute" rotation when changing the angle of its trajectory. In a different view, it seems that the passing and positioning are better in this model resulting in more goals scored, however, some missed kicks still occur. Shifting to the previously trained model in Environment 2, *v71v2for6p3M58envKeepO**, once it is put on this third scenario, its quality decreases going from 37% to 28% in achievement, applying the retraining did improve the results seen in *71v2randRet6p3for6MKeepO*, nonetheless, it did not outshine the first model presented. It is important to notice that lesser tests were made with retraining compared to the rest, it could be possible that by adjusting the number of time-steps in training and retraining that the *v71v2Randfor12MKeepTWO* model is surpassed. Regarding the baseline implementation, all of the trained models present here exceeded the achievement percentage established by it. Nonetheless, with a more attentive visualization on the Roboviz of the tests done with the base, it was observed that even though the *R2*'s target was defined as soon as the ball was passed, meaning that this agent could prepare its kick since the ball started moving, yet poor executions still occurred. This reveals that some of the failing kicks are associated with the fact that none of these skills are perfect and guarantee a 100% of accuracy and efficacy. This also explains the occasional falls that the first agent shows in the results table which occurs less often given that this agent is in a less dynamic and "safer" zone than its teammate.

Chapter 5

Conclusion and Future Work

This project had the objective of training two cooperative agents to execute a setplay to be applied in situations of kick-ins and corners that always results in a goal. The base of construction should be the already existing skills developed previously in the team and then apply Deep Learning concepts, more specifically Deep Reinforcement Learning algorithms, to achieve the desired outcome. The what's and how's were decided by the author and discussed with the supervisor resulting in pursuing training with a continuous action space related to positioning, targets, and timing.

Most of the development of this project is reported on this document that began with a general overview of the world that this dissertation is set on as well as the motives and intentions for the project. Following, chapter 2, presented the theoretical background starting with trivial concepts in Reinforcement Learning going up to several existing divisions and differences in algorithms. In addition, the concept of a multi-agent environment and its comparison with real soccer is also present. Chapter 3 explores the FC Portugal team, the first half of this chapter discriminates several tools developed by the team as well as the existing skills that were created through the years, closing with recent publications from the team related to RL. The last part of the chapter includes the software utilized for training and a description of the process of doing it, plus, the alterations that need to happen to train two agents simultaneously. The following chapter 4, contains some of the approaches taken during the project to achieve the desired setplay. Starting with the training with no goalkeeper, these environments were meant to show the influence of the use of the neural network as well as the reward shaping. To improve the results obtained, environments should have included a better approach to the target locking problem instead of keeping the same output once the kicking stage is in action. Another alteration could come from simplifying the action space to just include the targets. Although, this is not a significant improvement to what is already being used in the team, the only difference would be that the targets won't be calculated using complex functions but instead are decided by a neural network. Nonetheless, it is highly possible that by only training targets the average achievement results would increase. Following, in the scenarios with a goalkeeper, it might have helped if the first environments had a static keeper starting in a random position close to the goal and within its limits instead of an active opponent right away. Nevertheless, the best model trained in these conditions is not a total deception, even though it only has 40% of achievement, it is important to notice that, when applying this model, the episode termination conditions were kept. In the 3D league games, even if the agent that is going to pass the ball falls it can get itself up and return to the ongoing setplay, similarly, the scoring agent also falling does not mean that is impossible for the team to score in this circumstance. In short, the conditions for testing the models were a bit more rigid than those that exist in games. In addition, when comparing the total of goals (in tables) from most models with the baseline implementations it is clear that the use of Reinforcement Learning for target selection improves scoring. As a final notice, even though the low-level skills are at a fair level to pursue higher implementations, these still show some weaknesses not being accurate 100% of the time. This project was able to reveal some of the points that need to be improved in future skills, since that, even in the baseline implementations, problems such as poor kicking or falling while walking to perform a kick were also present.

5.1 Future Work

Future implementations should include training the agents to chose the best kick for each situation (possibly with a discrete action space), also, add randomness to the setplay so that after being used in some games it doesn't become predictable, this can be done by having the passing agent trained to decide which teammate is the best to pass the ball too. For simpler developments, the definition in terms of reward attribution should also be better explored as well as the addition of each agents' targets as observations (using communication), deeper research related to using different algorithms or having a finer parameter tuning for the PPO should also be done. Finally, the applications of MARL algorithms would be highly interesting.

References

[1] Robocup 3d soccer simulation history. URL: https://ssim.robocup.org/3d-simulation/3d-history/ [last accessed 20-01-2021].

[2] Robocup objective. URL: https://www.robocup.org/objective [last accessed 20-01-2021].

[3] E. Alpaydin. *Introduction to Machine Learning*. Adaptive computation and machine learning. MIT Press, 2010. URL: https://books.google.pt/books?id=4j9GAQAAIAAJ.

[4] Rudolph. Russell. *Machine learning step-by-step guide to implement machine learning algorithms with Python*. CreateSpace Independent Publishing Platform, 2018.

[5] Philippe Thomas. Semi-Supervised Learning edited by O . Chapelle , B . Schölkopf and A . Zien (Review). *IEEE Transactions on Neural Networks*, 2009.

[6] Michael J. Wooldridge. *An introduction to Multiagent Systems*. John Wiley and Sons, 2009.

[7] L. Panait and S. Luke. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[8] Ben Kröse, Ben Krose, Patrick van der Smagt, and Patrick Smagt. An introduction to neural networks, 1993.

[9] Luis Bermudez. Overview of neural networks, Nov 2017. URL: https://medium.com/machinevision/overview-of-neural-networks-b86ce02ea3d1 [last accessed 23-01-2021].

[10] Alsaadi A. Mijwil MM. Overview of Neural Networks. *Computer Engineering Techniques Department*, 2019.

[11] Propagation function. URL: https://www.nnwj.de/propagation-function-term.html [last accessed 23-01-2021].

[12] Matiur Rahman Minar and Jibon Naher. Recent advances in deep learning: An overview. *arXiv*, 2006:1–31, 2018. arXiv:1807.08169, doi:10.13140/RG.2.2.24831.10403.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[14] Kung-Hsiang. Introduction to various reinforcement learning algorithms. part i (q-learning, sarsa, dqn, ddpg), 2018. URL: https://bit.ly/3mqKezK [last accessed 16-08-2021].

[15] Ram Sagar. On-policy vs off-policy reinforcement learning: The differences, Oct 2020. URL: https://analyticsindiamag.com/reinforcement-learning-policy/ [last accessed 13-02-2021].

[16] Hao Dong, Zihan Ding, and Shanghang Zhang. *Deep reinforcement learning: Fundamentals, research and applications*. Springer, 2020. doi:10.1007/978-981-15-4095-0.

[17] Sarthak Bhagat and Hritwick Banerjee. Deep Reinforcement Learning for Soft Robotic Applications : Brief Overview with Impending Challenges. *Unpublished*, pages 1–27, 2018. URL: http://rgdoi.net/10.13140/RG.2.2.24264.37125/1, doi:10.20944/preprints201811.0510.v2.

[18] Aristotelis Lazaridis, Anestis Fachantidis, and Ioannis Vlahavas. Deep Reinforcement Learning: A State-of-the-Art Walkthrough. *Journal of Artificial Intelligence Research (JAIR)*, Dec 2020. URL: https://jair.org/index.php/jair/article/view/12412.

[19] Lilian Weng. Policy gradient algorithms, Apr 2018. URL: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html [last accessed 20-08-2021].

[20] Actor-critic methods. URL: http://incompleteideas.net/book/first/ebook/node66.html [last accessed 23-08-2021].

[21] Volodymyr Mnih, Adria Puigdomenech Badia, Lehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016*, 4:2850–2869, 2016. arXiv:1602.01783.

[22] Jonathan Hui. Rl — trust region policy optimization (trpo) explained, Oct 2018. URL: https://bit.ly/3nIqoPN [last accessed 24-08-2021].

[23] Robert Moni. Reinforcement learning algorithms — an intuitive overview, Feb 2019. URL: https://bit.ly/3mpuSeO [last accessed 24-08-2021].

[24] John Schulman. Proximal policy optimization, Sep 2020. URL: https://openai.com/blog/openai-baselines-ppo.

[25] Piyush K. Sharma, Rolando Fernandez, Erin Zaroukian, Michael Dorothy, Anjon Basak, and Derrik E. Asher. Survey of recent multi-agent reinforcement learning algorithms utilizing centralized training. In Tien Pham and Latasha Solomon, editors, *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, volume 11746, pages 665 – 676. International Society for Optics and Photonics, SPIE, 2021. URL: https://doi.org/10.1117/12.2585808.

[26] Hassam Ullah Sheikh and Ladislau Boloni. Multi-agent reinforcement learning for problems with combined individual and team reward. *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020. doi:10.1109/ijcnn48605.2020.9206879.

[27] Justin Terry. Multi-agent deep reinforcement learning in 13 lines of code using pettingzoo, Jun 2021. URL: https://bit.ly/2ZDOZNU [last accessed 23-08-2021].

[28] Lilian Weng. Evolution strategies, Sep 2019. URL: https://lilianweng.github.io/lil-log/2019/09/05/evolution-strategies.html#what-are-evolution-strategies [last accessed 13-02-2021].

[29] Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial. *ArXiv*, 2016. URL: http://arxiv.org/abs/1604.00772, arXiv:1604.00772.

[30] Nuno Lau, Luís P. Reis, Abbas Abdolmaleki, and Gerhard Neumann. Deriving and improving CMA-ES with information geometric trust regions. *GECCO 2017 - Proceedings of the 2017 Genetic and Evolutionary Computation Conference*, pages 657–664, 2017.

[31] Ventura de Sousa Pereira. FCPortugal Multi-Robot Action Learning. Technical report, Faculdade de Engenharia da Universidade do Porto, 2020. URL: https://hdl.handle.net/10216/131458.

[32] Luís Paulo Reis, Nuno Lau, Luís Rei, Nima Shafii, and Bruno Pimentel. FC Portugal 3D Simulation Team : Team Description Paper 2019. *Science*, 2019.

[33] Nuno Lau and Luís P. Reis. FC Portugal - High-Level Coordination Methodologies in Soccer Robotics. *Robotic Soccer*, 2007. doi:10.5772/5130.

[34] Peter Stone and Manuela Veloso. Layered learning and flexible teamwork in RoboCup simulation agents. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1856:495–508, 2000. doi:10.1007/3-540-45327-x_42.

[35] Luís Paulo Reis, Fernando Almeida, Luís Mota, and Nuno Lau. Coordination in multi-robot systems: Applications in robotic soccer. In Joaquim Filipe and Ana Fred, editors, *Agents and Artificial Intelligence*, pages 3–21, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-36907-0_1.

[36] Hugo Picado, Marcos Gestal, Nuno Lau, Luís P. Reis, and Ana M. Tomé. Automatic generation of biped walk behavior using genetic algorithms. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5517 LNCS(PART 1):805–812, 2009. doi:10.1007/978-3-642-02478-8_101.

[37] Nima Shafii, Luís Paulo Reis, and Nuno Lau. Biped Walking Using Coronal and Sagittal Movements. *Lecture Notes in Computer Science*, 2011.

[38] Nima Shafii, Nuno Lau, and Luis Paulo Reis. Learning to Walk Fast: Optimized Hip Height Movement for Simulated and Real Humanoid Robots. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 80(3-4):555–571, 2015. doi:10.1007/s10846-015-0191-5.

[39] Nima Shafii, Abbas Abdolmaleki, Rui Ferreira, Nuno Lau, and Luís Paulo Reis. Omnidirectional Walking and Active Balance. *Lecture Notes in Computer Science*, 2013.

[40] Rui Ferreira, Nima Shafii, Nuno Lau, Luís Paulo Reis, and Abbas Abdolmaleki. Diagonal walk reference generator based on Fourier approximation of ZMP trajectory. *Proceedings of the 2013 13th International Conference on Autonomous Robot Systems, ROBOTICA 2013*, 2013. doi:10.1109/Robotica.2013.6623534.

[41] Nima Snafii, Abbas Abdolmaleki, Nuno Lau, and Luís Paulo Reis. Development of an Omnidirectional Walk Engine for Soccer Humanoid Robots. *International Journal of Advanced Robotic Systems*, 12(12):1–14, 2015. doi:10.5772/61314.

[42] Abbas Abdolmaleki, Nima Shafii, Luís Paulo Reis, Nuno Lau, Jan Peters, and Gerhard Neumann. Omnidirectional walking with a compliant inverted pendulum model. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8864:481–493, 2014. doi:10.1007/978-3-319-12027-0_39.

[43] Nima Shafii, Nuno Lau, and Luis Paulo Reis. Generalized learning to create an energy efficient ZMP-based walking. *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, 8992:583–595, 2015. doi:10.1007/978-3-319-18615-3_48.

[44] Nima Shafii, Nuno Lau, and Luis Paulo Reis. Learning a fast walk based on ZMP control and hip height movement. *2014 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2014*, pages 181–186, 2014. doi:10.1109/ICARSC.2014.6849783.

[45] S. Mohammadreza Kasaei, David Simões, Nuno Lau, and Artur Pereira. A Hybrid ZMP-CPG Based Walk Engine for Biped Robots. *Advances in Intelligent Systems and Computing*, 694(December 2017):743–755, 2018. doi:10.1007/978-3-319-70836-2_61.

[46] Miguel Abreu, Luís Paulo Reis, and Nuno Lau. Learning to Run Faster in a Humanoid Robot Soccer Environment Through Reinforcement Learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11531 LNAI:3–15, 2019. doi:10.1007/978-3-030-35699-6_1.

[47] Miguel Abreu, Nuno Lau, Armando Sousa, and Luís Paulo Reis. Learning low level skills from scratch for humanoid robot soccer using deep reinforcement learning. *19th IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2019*, 2019. doi:10.1109/ICARSC.2019.8733632.

[48] Henrique Teixeira, Tiago Silva, Miguel Abreu, and Luís Paulo Reis. Humanoid robot kick in motion ability for playing robotic soccer. *2020 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2020*, pages 34–39, 2020. doi:10.1109/ICARSC49921.2020.9096073.

[49] Luís P. Reis, Nuno Lau, and Luís Mota. FC Portugal 3D Simulation Team: Team Description Paper. *Iscte.Pt*, 2013. URL: https://archive.robocup.info/Soccer/Simulation/3D/TDPs/RoboCup/2013/FCPortugal_SS3D_RC2013_TDP.pdf.

[50] Luís Cruz, Luís Paulo Reis, Nuno Lau, and Armando Sousa. Advances in Artificial Intelligence – IBERAMIA 2012. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7637(November):491–500, 2012. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-84887902079&partnerID=tZOtx3y1, doi:10.1007/978-3-642-34654-5.

[51] Rui Ferreira, Luís Paulo Reis, and António Paulo Moreira. Omnidirectional kick for a humanoid robot. *Iberian Conference on Information Systems and Technologies, CISTI*, 2012.

[52] Abbas Abdolmaleki, David Simões, Nuno Lau, Luís Paulo Reis, and Gerhard Neumann. Learning a humanoid kick with controlled distance. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9776 LNAI(July):45–57, 2017. doi:10.1007/978-3-319-68792-6_4.

[53] David Simões, Pedro Amaro, Tiago Silva, Nuno Lau, and Luís Paulo Reis. Learning Low-Level Behaviors and High-Level Strategies in Humanoid Soccer. *Advances in Intelligent Systems and Computing*, 1093 AISC:537–548, 2020. doi:10.1007/978-3-030-36150-1_44.

[54] Nuno Lau, Luís Paulo Reis, Hugo Marques, and Hugo Penedones. FC Portugal 2004 Team : Strategic Positioning for 3D Soccer. *World*, pages 3–4, 2004.

[55] Luís Paulo Reis and Nuno Lau. COACH UNILANG - A standard language for coaching a (Robo)Soccer team. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2377 LNAI:183–192, 2002. doi:10.1007/3-540-45603-1_19.

[56] Luís Paulo Reis and Nuno Lau. FC Portugal team description: RoboCup 2000 Simulation League Champion. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019 LNAI(May 2014):29–40, 2001. doi:10.1007/3-540-45324-5_2.

[57] João Cravo, Fernando Almeida, Pedro Henriques Abreu, Luís Paulo Reis, Nuno Lau, and Luís Mota. Strategy planner: Graphical definition of soccer set-plays. *Data and Knowledge Engineering*, 94(PA):110–131, 2014. URL: http://dx.doi.org/10.1016/j.datak.2014.10.001, doi:10.1016/j.datak.2014.10.001.

[58] Luís Mota, Nuno Lau, and Luís Paulo Reis. Co-ordination in RoboCup's 2D simulation league: Setplays as flexible, multi-robot plans. *2010 IEEE Conference on Robotics, Automation and Mechatronics, RAM 2010*, pages 362–367, 2010. doi:10.1109/RAMECH.2010.5513166.

[59] Miguel Abreu, Luís Paulo Reis, and Henrique Lopes Cardoso. Learning high-level robotic soccer strategies from scratch through reinforcement learning. *19th IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2019*, pages 0–6, 2019. doi:10.1109/ICARSC.2019.8733606.

[60] Patrick MacAlpine and Peter Stone. Overlapping layered learning. *Artificial Intelligence*, 254(January):21–43, 2018. doi:10.1016/j.artint.2017.09.001.

[61] About simspark. URL: http://simspark.sourceforge.net/wiki/index.php/About_SimSpark [last accessed 10-02-2021].

[62] Joschka Boedecker and Minoru Asada. Simspark– Concepts and Application in the Robocup 3d Soccer Simulation League. *Autonomous Robots*, 174:181, 2008.

[63] Soccer simulation · wiki · robocup simulation / simspark. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Soccer-Simulation [last accessed 10-02-2021].

[64] Maziar Palhang, Nuno Lau, and David Simões. RoboCup Soccer Simulation 3D League Rules. 2019. URL: https://ssim.robocup.org/3d-simulation/3d-rules/.

[65] URL: https://ssim.robocup.org/3d-simulation/3d-rules/ [last accessed 16-08-2021].

[66] Agent proxy · wiki · robocup simulation / simspark. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Agent-Proxy [last accessed 13-08-2021].

[67] Network protocol · wiki · robocup simulation / simspark. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Network-Protocol [last accessed 13-08-2021].

[68] Agent sync mode · wiki · robocup simulation / simspark. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Network-Protocol [last accessed 13-08-2021].

[69] magmaOffenburg. magmaoffenburg/roboviz, Sep 2020. URL: https://github.com/magmaOffenburg/RoboViz/blob/master/README.md [last accessed 10-02-2021].

[70] Nao the humanoid and programmable robot: Softbank robotics. URL: https://www.softbankrobotics.com/emea/en/nao.

[71] Models · wiki · robocup simulation / simspark. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Models [last accessed 10-02-2021].

[72] Dec 2016. URL: https://team.inria.fr/perception/nao/ [last accessed 16-08-2021].

[73] Martín Abadi, Paul Barham, Jianmin Chen, and et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 265–283, USA, 2016. USENIX Association.

[74] Tensorboard integration. URL: https://stable-baselines.readthedocs.io/en/master/guide/tensorboard.html.

[75] Acer. URL: https://stable-baselines.readthedocs.io/en/master/modules/acer.html [last accessed 15-08-2021].

[76] Acktr. URL: https://stable-baselines.readthedocs.io/en/master/modules/acktr.html [last accessed 15-08-2021].

[77] Her. URL: https://stable-baselines.readthedocs.io/en/master/modules/her.html [last accessed 15-08-2021].

[78] Gail. URL: https://stable-baselines.readthedocs.io/en/master/modules/gail.html [last accessed 15-08-2021].

[79] Vectorized environments. URL: https://stable-baselines.readthedocs.io/en/master/guide/vec_envs.html [last accessed 16-08-2021].

[80] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL: http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17.

[81] Ppo2. URL: https://stable-baselines.readthedocs.io/en/master/modules/ppo2.html.