EVALUATING REINFORCEMENT LEARNING ON ROBOCUP SOCCER SIMULATION 2D

By

MATEUS GONÇALVES MACHADO

B.Sc. Dissertation

Federal University of Pernambuco
secgrad@cin.ufpe.br
www.cin.ufpe.br/~secgrad

RECIFE/2019

Agradecimentos

Gostaria de agradecer primeiramente a minha família por ter me dado sempre uma educação incrível. Dona Selma, seu Alexandre, parabéns por estarem formando o último filho. Agradecer a meu irmão e irmã que sem eles eu tenho certeza que não teria tanta inspiração para a maioria das coisas da minha vida inteira. Gostaria também de agradecer à Samara, minha namorada, que por querer ter tanta força e garra como ela, venho me tornando melhor a cada dia. Agradecer aos meus amigos, principalmente (só pela ordem alfabética): Ana Luiza, Anna Letícia, Cristiano Oliveira e Jailson Gomes. Obrigado por estarem sempre e sempre comigo, tenham certeza que tenho em vocês irmãos que escolhi ter. Um obrigado especialsíssimo a Walber Macêdo por ter limite emocional para me aguentar. E claro, obrigado a equipe que se tornou minha segunda família, a principal razão de eu ter seguido essa linha de pesquisa. Obrigado RobôCIn! A cada um que agradeço neste documento, amo vocês demais.

MAMÃE ACABOU!

Firmeza total, mais um ano se passando
Graças a Deus a gente tá com saúde aí, 'morô?
Muita coletividade na quebrada, dinheiro no bolso
Sem miséria, e é nóis
Vamos brindar o dia de hoje
Que o amanhã só pertence a Deus, a vida é loka.

—RACIONAIS MC'S

Resumo

A liga de simulação 2D de futebol do RoboCup é uma das mais maduras da competição, iniciada em 1996. Uma das tarefas a ser realizada pelos agentes inteligentes é a linha de defesa. Ela consiste em saber a hora certa entre interceptar a bola ou fazer uma marcação inteligente (bloqueando prováveis chutes ou passes). Uma investida errada do agente pode levar a um drible ou passe do atacante e um provável gol. Aprendizagem supervisionada e algoritmos determinísticos são as técnicas mais usadas pelas 5 melhores equipes para resolver o problema. Pesquisas recentes usando Aprendizagem Profunda por Reforço (APR) para treinar agentes autônomos em sistemas multiagentes superaram os agentes com base em algoritmos supervisionados ou determinísticos. Um exemplo é a equipe CYRUS2019 que produziu jogadores defensivos treinados com o APR, alcançando em terceiro lugar na RoboCup 2019. Este trabalho compara três técnicas de APR em agentes defensivos com base no CYRUS2019, adaptando o *Half Field Offensive* para ser um ambiente semelhante aos da OpenAI GYM e aplicar a melhor técnica aos agentes do time RoboCIn2d.

Abstract

The Simulation Soccer 2D League of RoboCup is one of the most matures leagues of the competition having started in 1996. One of the tasks of the intelligent agents is the defense line. It consists of knowing the timing to intercept the ball or do a clever mark (blocking future shoots or passes). A bad timing will lead to an attacker's dribble or pass and probably a goal. Supervised Learning and deterministic algorithms are the usual techniques used by the TOP 5 teams to solve that problem. Recent researches using Deep Reinforcement Learning to train autonomous agents in multi-agent systems have shown that it outperforms agents based on supervised or deterministic algorithms. The team CYRUS2019 has released defensive players trained with Deep Reinforcement Learning achieving in third place on RoboCup2019. This work compares three DRL techniques for defensive agents based on CYRUS2019 adapting the Half Field Offensive environment to an OpenAI GYM like environment and apply the best technique on RoboCIn2d's agents.

List of Figures

1.1 Example of real game of Simulation Soccer 2D League. 12

2.1 Default field parameters given by the server. Image from CHEN et al. (2003). . 15
2.2 Actual Footage of Fedit2. Image from AKIYAMA; NAKASHIMA (2014). . . 15
2.3 Example of game analysis by Cyrus2014's coach. 16
2.4 Ranges of view of the players. Image from CHEN et al. (2003). 17
2.5 Description of Catchable Area. Image from CHEN et al. (2003). 18
2.6 Half Field Offensive diagram. Image from HAUSKNECHT (2015). 18

3.1 Example of model of Reinforcement Learning. Image from SUTTON; BARTO (2011). 20
3.2 Formal description of *Q-Value*. Image from OLIVEIRA LIMA JúNIOR; BASANI (2019). 20
3.3 Example of Q-Table. By LearnDataSci - Own work - Article, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=69947708, accessed on 15/09/2019. 22
3.4 $L_i(\theta_i)$ as loss function of DQN's Neural Networks. 22
3.5 Deep Q Networks algorithm with Experience Replay. 23
3.6 Double DQN algorithm. Image from https://towardsdatascience.com/double-deep-q-networks-905dd8325412, accessed on 15/11/2019. 24
3.7 Q-Value in fuction of time for some Atari games: estimation and truth values for DQN and Double DQN. Image from VAN HASSELT; GUEZ; SILVER (2015). 24
3.8 Deep Q Networks architecture against Dueling Deep Q Networks architecture. Image from WANG; FREITAS; LANCTOT (2015) 25
3.9 Illustration of an Actor Critic system. Image from SUTTON; BARTO (2011). . 25
3.10 Deep Deterministic Policy Gradient algorithm. Image from LILLICRAP et al. (2015). 26

4.1 Cross Mark behaviour. Observe the positioning of numbers 3, 4, 7 and 8 how the get in "front" of the attackers. Image from TAVAFI et al. (2011). 28
4.2 Play_on Mark behaviour. Image from TAVAFI et al. (2011). 28
4.3 Play_on Mark behaviour. Observe the positioning of numbers 5 and 3 ready to block a through pass. 29
4.4 Block behaviour. Observe the yellow team positioning of numbers 6, 7 and 10 ready to block the pass. 29
4.5 Block behaviour. Observe the positioning of number 5 going to intercept the ball. 30

4.6 WrightEagle's defensive net. The blue area represents the area to be defended and the blacks, all calculated areas. Image from ZHANG et al. (2014). 31
4.7 Gliders' piece of code showing the fine tuning of pressing variable. Image from PROKOPENKO; WANG (2018). 31
4.8 Neurohassle's *training situations*. Image from GABEL; RIEDMILLER; TROST (2009). 33
4.9 Neurohassle's *training regions*. Image from GABEL; RIEDMILLER; TROST (2009). 33
4.10 Example of Learning Curve for Learning to Hassle. Image from GABEL; RIEDMILLER; TROST (2009). 34
4.11 *Training regions* analysis for some teams from RoboCup2006. Image from GABEL; RIEDMILLER; TROST (2009). 34
4.12 Cyrus' Block movement. Image from ZARE et al. (2019). 35
4.13 Intercept movement. Image from ZARE et al. (2019). 35
4.14 Cyrus' Goalie Assistant movement. Image from ZARE et al. (2019). 35
4.15 Cyrus' reward model. Image from ZARE et al. (2019). 36
4.16 Cyrus' DDPG results. The best result is choosing a action and then CYRUS2018 (ZARE et al. (2018)) doing that action. Image from ZARE et al. (2019). 36

6.1 Neural Networks' and Memory's parameters. 40
6.2 Deep Q Network results. Loss convergence and Rewards in function of number of episodes. 41
6.3 Deep Deterministic Policy Gradient results. Loss convergence of Actor and Critic networks and Rewards in function of number of episodes. 42
6.4 Duelling Double Deep Q Network results. Loss convergence and Rewards in function of number of episodes. 42

List of Tables

6.1 Result baseline. Rows are the defenders and Columns are the attackers of each original team. Percentage of successful defenses for 3000 episodes. 43
6.2 Tests for 3000 episodes. Rows are the DRL defenders with Helios2013's goalie and Columns are the attackers. Percentage of successful defenses for 3000 episodes. 43
6.3 Tests for 3000 episodes. Rows are the DRL defenders with RoboCIn2019's goalie and Columns are the attackers. Percentage of successful defenses for 3000 episodes. 43

Contents

1 Introduction **11**
1.1 Objectives . . . 12

2 Soccer Simulation 2D **14**
2.1 Server . . . 14
2.2 Formation . . . 15
2.3 Agents . . . 16
2.3.1 Coach . . . 16
2.3.2 Players . . . 16
2.4 Half Field Offensive . . . 18

3 Reinforcement Learning **19**
3.1 Concepts and Definitions . . . 19
3.2 Q-Learning . . . 21
3.3 Deep Q Network . . . 22
3.4 Double Deep Q Networks . . . 23
3.5 Dueling Double Deep Q Networks . . . 24
3.6 Deep Deterministic Policy Gradient . . . 25

4 Background **27**
4.1 Deterministic Agents . . . 27
4.1.1 Marlik2011 Defense . . . 27
4.1.1.1 Cross Mark . . . 27
4.1.1.2 Play_on Mark . . . 28
4.1.1.3 Block . . . 29
4.1.2 WrightEagle2014 Defense . . . 30
4.1.3 Gliders2d Defense . . . 30
4.2 Reinforcement Learning Agents . . . 32
4.2.1 Brainstormers2d . . . 32
4.2.1.1 Intercept Ball Task . . . 32
4.2.1.2 The NeuroHassle Approach . . . 32
4.2.2 Cyrus2019 . . . 35

5 Environment **37**
5.1 HFO Server's Referee . . . 37
5.2 Feature Set . . . 38
5.3 Actions . . . 38

5.4 Reward System 39

6 Experiments and Results **40**

6.1 Experiments 40

6.2 Results 41

6.2.1 Training 41

6.2.2 Tests 42

7 Conclusion **44**

References **45**

1
Introduction

The Simulation Soccer 2D League of ROBOCUP (2019) (Sim2d) is one of the most matures leagues of the competition having started in 1996. One of the tasks of the intelligent agents is the defense line. It consists of knowing the timing to intercept the ball or do a clever mark (blocking future shoots or passes). A bad timing will lead to an attacker's dribble or pass and probably a goal. Supervised Learning and deterministic algorithms are the usual techniques used by the TOP 5 teams to solve that problem. Recent researches using Deep Reinforcement Learning to train autonomous agents in multi-agent systems have shown that it outperforms agents based on supervised or deterministic algorithms. The team CYRUS2019, ZARE et al. (2019), has released defensive players trained with Deep Reinforcement Learning achieving in third place on RoboCup2019. This work compares three DRL techniques for defensive agents based on CYRUS2019 adapting the Half Field Offensive environment to an OpenAI GYM, Brockman et al. (2016), like environment and apply the best technique on ROBOCIN (2019) agents.

With the goal of beating the human soccer World Champion team in 2050, Robocup has been a great competition to develop and share robotics and autonomous agents techniques. Among the leagues, Soccer Simulation 2D league is a great domain to invest machine learning (ML) algorithms and it is been a challenge as KITANO et al. (1997) say. At the very beginning of the league at 1996, the competitors used to implement by their own hands the behaviours of the players. Once it's a simulated league, Machine Learning techniques became natural since there is no mechanic or electronic interference. The German team, Brainstormers was one of the pilots with an approach using Reinforcement Learning (RIEDMILLER et al. (2002)). Since then teams frequently uses Artificial Intelligence algorithms to have unnatural and unpredictable behaviours.

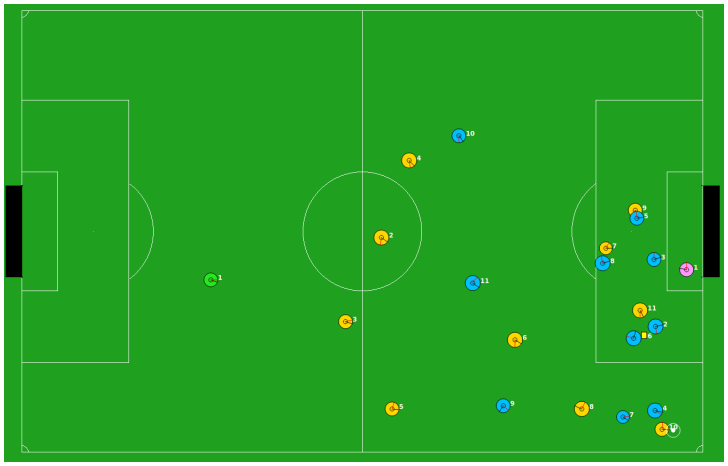

Figure 1.1: Example of real game of Simulation Soccer 2D League.

This work focus on the defense line, specifically in a single intelligent defender cooperating with a hand-coded goalie against a single attacker. As there is more works about the attackers like KYRYLOV; BERGMAN; GREBER (2005), STONE; SUTTON; KUHLMANN (2005) and more recent HAUSKNECHT; STONE (2015), the defense line became attractive to the RoboCIn team.

DeepMind (2014) researchers MNIH et al. (2013) have shown that Deep Q Learning was so effective on Atari games that it outplays human controlled agents. In 2015 Deepmind's AlphaGo agent, SILVER et al. (2016), won from the European Go Champion Fan Hui and later on 2016 from Lee Sedol, one of the best worldwide. Since then OpenAI and DeepMind has invested a lot of effort on Reinforcement Learning (RL) and Autonomous Agents (AA) systems. Recently OpenAI (2018) has become the best agent on Dota2 (2011) on 1v1 games and it has a great teamwork performance on 5v5 games winning from most of amateur teams and a few professionals.

Inspired by those recent researches ZARE et al. (2019) modeled a defensive autonomous agent and it has almost the same efficiency of the champions' Fractals (PROKOPENKO; WANG (2019), and Helios, AKIYAMA1 et al. (2016), defenses.

1.1 Objectives

Based on ZARE et al. (2019) work we compare the Deep Reinforcement Learning models: MNIH et al. (2013), WANG; FREITAS; LANCTOT (2015), LILLICRAP et al. (2015); applied to an environment very likely to the real Soccer Simulation 2D for a defensive agent cooperating with a goalie from a deterministic team to be introduced in RoboCIn2d's team.

This work is divided in 7 chapters. Chapter 2 is about the simulator itself, the description of each piece that makes a playable agent. Chapter 4 is about the most influential works about deterministic and RL agents applied to defense line in the Soccer Simulation 2D league. Chapter 5 is the environment that we used in this work, specifying state's feature set, actions and reward

system. Chapter 3 details the DRL architectures that we used and theirs training algorithms. Chapter 6 shows the results of the experiments that we made and the comparisons with some teams. Chapter 7 is the discussion of each result and comparison.

2

Soccer Simulation 2D

In this chapter we explain the architecture of the Simulator itself. It is divided in 5 sections: about RoboCup Soccer Simulation Server (rcssserver), ROBOCUP (2018), explanation of the formation tactics and the formation editor, a section about the two major agents: the coach and the players; and a Section to show the environment used to train our agents.

2.1 Server

Robocup Soccer Simulation Server (RCSSS) is the core of SS2D. It processes the whole environment receiving the messages from every agent and returning the actual state of the game in discretized time. An old challenge to teams in the league is that the returned state is noisy, therefore, not fully trustful. Also, it establishes the communication between the communication with monitors, from which one can watch the game.

RCSSS follows an UDP/IP client-server style, which allows teams to develop on any programming language since they do that communication. There's an specific port for the 11 infield players and another to the coach. Only messages with the allowed protocols is processed by the server. For example, only the goalie can do catch, if any other player do catch, the action is invalidated and the agent looses a cycle. The two agents with special commands are the goalie and the coach, which will be explained in Section 2.3.1.

After the initial connection with the clients, the server commits the it's parameters. Those parameters are divided in server-params, the environment itself (position of goal, size of ball, how many cycles it will wait for an action, etc), and the 18 player types. Each player type has random params for speed, acceleration, size (which influences in the tackle action), maximum stamina and kickable area. In game, clients can exchange N messages with other clients, N being given by the server-params.

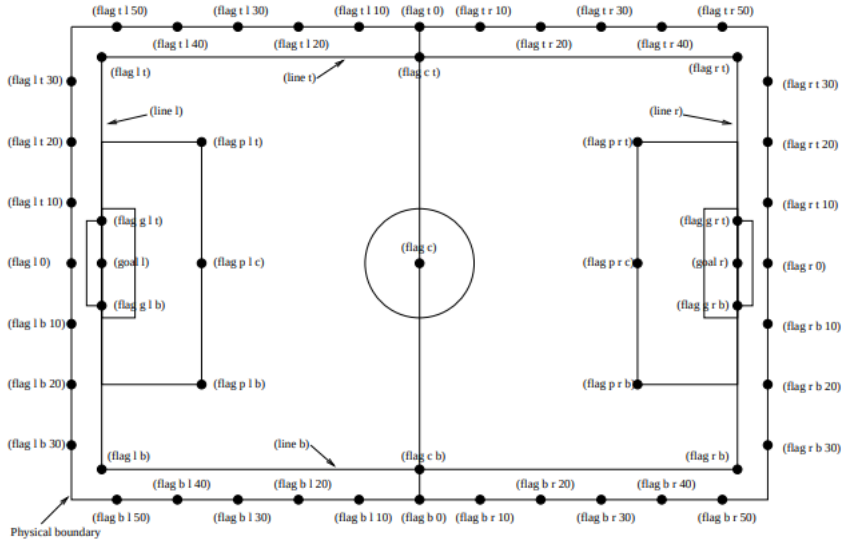

Figure 2.1: Default field parameters given by the server. Image from CHEN et al. (2003).

2.2 Formation

The formation file is one of the most tactical information that the players have. It describes positions depending on ball's position and, for an agent based on Helios Base, also called Agent2D, (AKIYAMA; NAKASHIMA (2014)), like great part of the teams in the league, it is indispensable this information. With the Agent2D, it was also released Fedit2. It is an user interface(UI) that you can create formations files with up to 128 static situations. See Figure 2.2.

Teams usually have more than one formation file to be used in game. The formations can be changed given a certain situation. For example, if our team is winning and has great chance of winning, we can choose an aggressive formation instead of a defensive one.

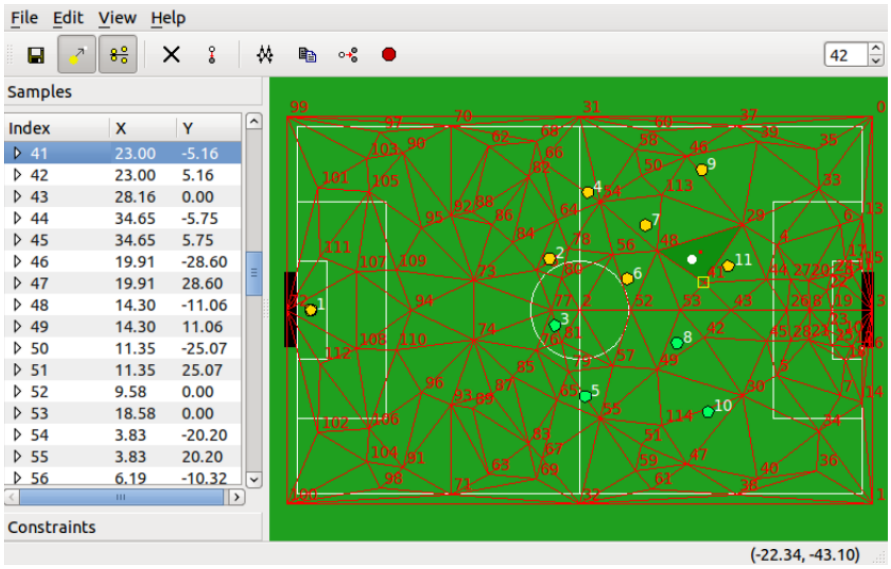

Figure 2.2: Actual Footage of Fedit2. Image from AKIYAMA; NAKASHIMA (2014).

2.3 Agents

In this section we shall talk about the main High-level agents: Coach and Player. The idea of each agent is based on a real soccer game where there is a coach and 11 players, where the coach can note things that the players cannot but they can send a message to a specific agent and only the player can execute the actions that the coach instructed them.

2.3.1 Coach

The coach is an special agent which can see all the field noiseless and has the greater message area of any agent. As it has global vision of the game, it can analyze globally the game (e.g: where our defense line is breaking, how we do goals more often, how many passes we got right). Cyrus2014, KHAYAMI et al. (2014), realized their code which shows some statistical information of the game. See Figure 2.3.

For the next RoboCup, the teams will not be able to see each others names, so strategies developed for a given team will be more difficult to be applied. For that, the coach is the most quoted agent to analyse the game in real time and tell what probable team is the opponent and then apply the specific strategy.

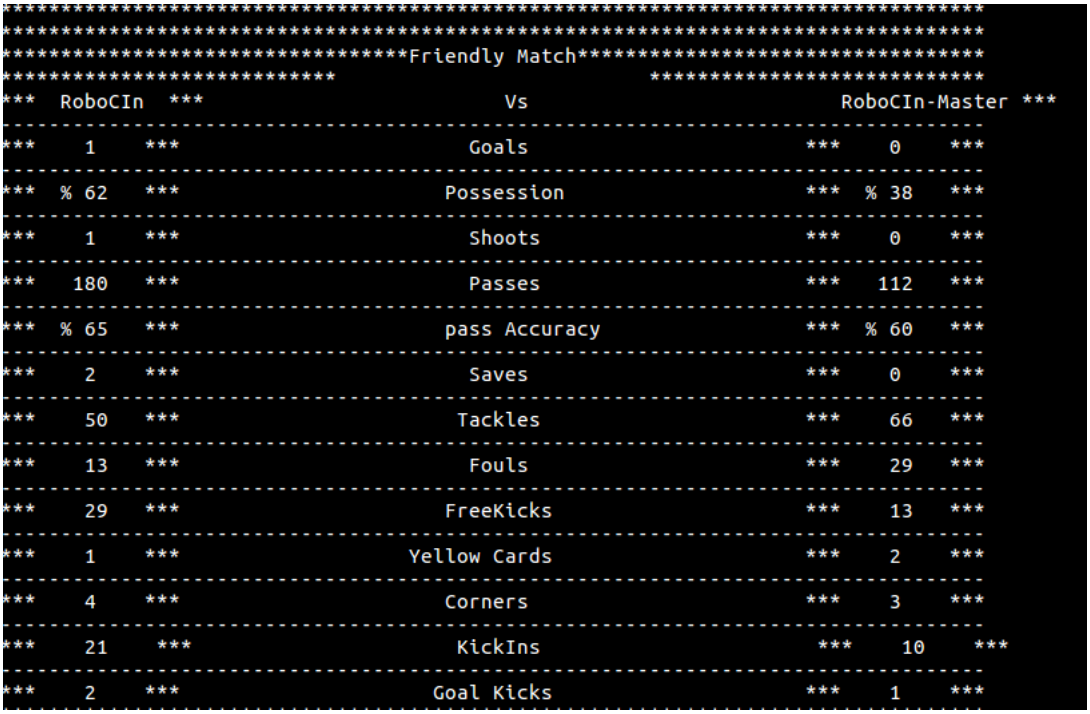

Figure 2.3: Example of game analysis by Cyrus2014's coach.

2.3.2 Players

The players are the usual agents in field. The players can take 8 actions:

- Dash: given a power of dash $\alpha \in [-100, 100]$, the agent "runs" with α in its body's direction.
- Turn: given an angle $\gamma \in [-180, 180]$, the agent turns its body in γ degrees.
- Kick: given a power $\alpha \in [-100, 100]$ and an angle $\gamma \in [-180, 180]$, the agent kicks in γ direction with power α.
- Tackle: given a power $\alpha \in [-100, 100]$, the agent tackles the ball.
- Say: given a message M and a target N, the agent sends to server M to delivered to N.
- Turn_neck: given an angle $\gamma \in [-180, 180]$, the agent turns its neck in γ degrees.
- Move: given a point (x,y) the agent is teletransported to (x,y). It is a special action done only while the game is paused.
- Catch: only the goalie can realize that action.

The agent has some restrictions of view as well. As far as the agent is of a target P, it becomes more difficult to see P. Figure 2.4 describes the view model, where point a, c and d the agent can see all parameters of P, at point e it cannot see the uniform number of P, at point f it cannot see which team P belongs to, at point b and g the agent cannot see. To cure some tactics that depends of all agents in field, AKIYAMA; NAKASHIMA (2014) created a memory for the agent that allows check an old position of the target. The catch and kick action also has some restrictions, it can only be performed if the ball is the kickable or catchable area. Figure 2.5 shows an example of catchable area.

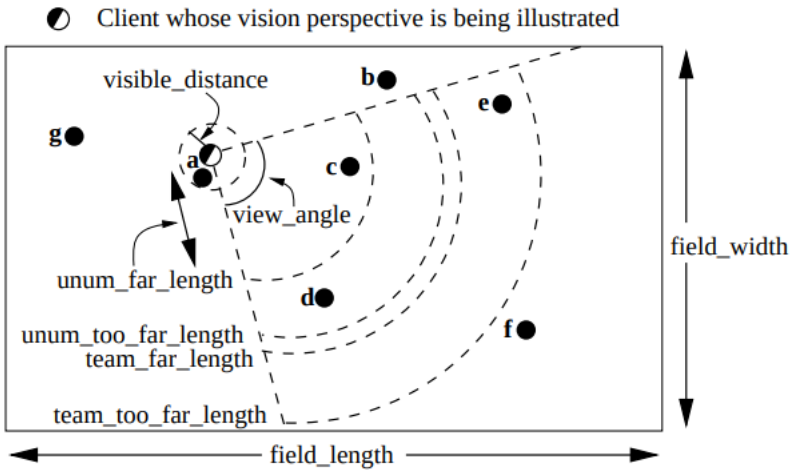

Figure 2.4: Ranges of view of the players. Image from CHEN et al. (2003).

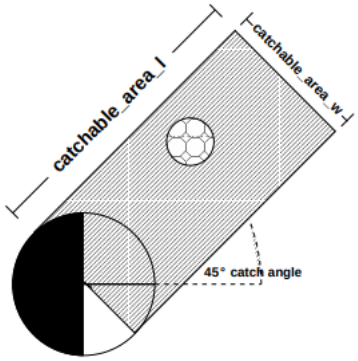

Figure 2.5: Description of Catchable Area. Image from CHEN et al. (2003).

2.4 Half Field Offensive

The Half Field Offensive (HFO), HAUSKNECHT (2015), is an interface environment designed specifically to train SARSA (state, action, reward, next state, next action) agents based on the usual OpenAI environments. HFO supports: a delayed message from the agent due to an algorithm training or some heavy process, writing the agent's code in Python Programming Language (VAN ROSSUM; DRAKE JR (1995)) using the original actions of the C++ Programming Language (ISO (1998)) agent via an interface, see figure 2.6. It provides 2 spaces of states and actions:

- Low-Level Features and Actions- Uses raw features from SS2D server (angle, positioning) and provides raw actions (kick, dash, turn).
- High-Level Features and Actions - Uses processed features (distance to opponent, interceptable ball) and only complex or chained actions (dribble, pass to teammate).

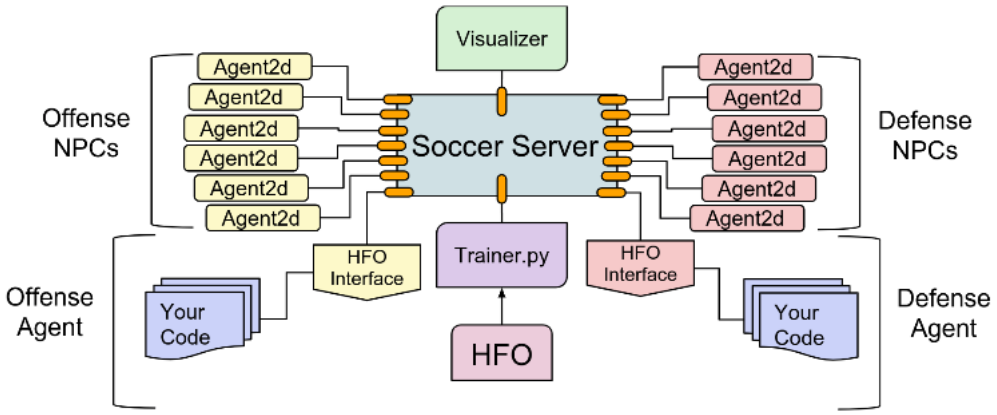

Figure 2.6: Half Field Offensive diagram. Image from HAUSKNECHT (2015).

3

Reinforcement Learning

In this chapter we explain the some concepts of Reinforcement Learning, a starting technique of RL and the three techniques that we used in this work. It is dived in 5 sections: explanation of the concepts and definitions of Reinforcement Learning, Q-Learning, Deep Q Network, Dueling Double Deep Q Networks and Deep Deterministic Policy Gradient.

3.1 Concepts and Definitions

Reinforcement Learning is a field of machine learning that studies a computational way to learn how to reach an objective taking actions over the time to maximize a reward function depending on environment's reaction of each action taken. To understand the RL techniques that we explain, we must learn some concepts and definitions.

An **agent** is the actor that will interact with the environment and understand if the action that it took was good or bad for the task. The **environment** is everything that the agent will interact, eg.: if we are drinking water the environment is the kitchen, the bottle of water, how much water last in the bottle. The environment is formally announced as a a Markov Decision Process (MDP), BELLMAN (1957). A MDP is a 4-tuple (S, A, P_a, R_a) where:

- S is a finite set of state that describe the features of the environment. A state s describes the environment in a certain time t.
- A is a finite set of actions that the agent can take at any time. An action a describes an action taken in a state s.
- $P_a(S, S') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability of agent taking the action a on state s at time t will lead to state s'.
- $R_a(S, S')$ is the reward for the agent taking an action a on a state s at time t leading on a state s'.

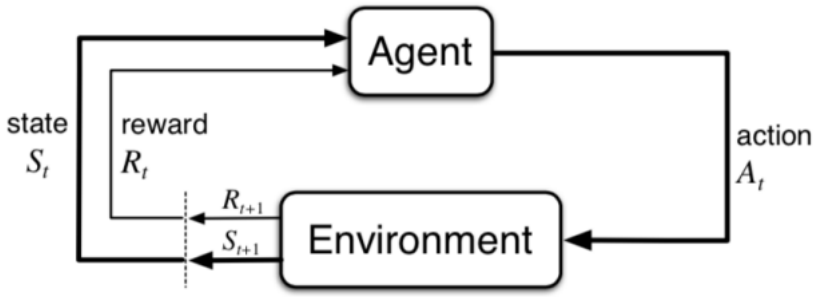

Figure 3.1: Example of model of Reinforcement Learning. Image from SUTTON; BARTO (2011).

Some definitions of RL:

Definition 3.1. A **Policy** π is a rule to select actions. It can be deterministic, if the state maps the actions, or stochastic, when the action choice depends on a probability distribution. The optimal policy is called π^*.

Definition 3.2. An **Episode** is analogous to an epoch in RL. An Episode τ is a sequence of 3-tuple (s,a,r).

Definition 3.3. A **Step** is the acting of the agent. Each episode have certain number of steps. In our domain, the agent can take approximately 300 steps until the current episode finishes.

Definition 3.4. The **Experience Replay** buffer (or **Memory**) of an agent is analogous to a dataset in RL. It is the previous N episodes passed, with N being defined by the programmer.

Definition 3.5. The **Value** of a state $V(s)$ describes the expected return after state s. It can be interpreted as how good s can be, based on previous memories.

Definition 3.6. The **Q-Value** $Q(s,a)$ describes how good is to take an action a in a given state s. The optimal *Q-Value* is called Q^*. The *Q-Value* is formally described as:

$$Q^{\pi}(s,a) = \underset{\tau\sim\pi}{E}[R(\tau)|s_0 = s, a_0 = a].$$

Figure 3.2: Formal description of *Q-Value*. Image from OLIVEIRA LIMA JúNIOR; BASANI (2019).

Definition 3.7. The **Bellman Equation** is the base of Reinforcement Learning. It can describe $V(s)$ and $Q(s,a)$ recursively based on the actual 4-tuple (s,a,r,s') and t-tuple (s,a,r,s',a') respectively. See Equations 3.1, 3.2 and 3.3.γ is the discount rate which represents how much $Q(s',a')$ can impacts on $Q(s,a)$.

$$V(s) = \max_a(R + \gamma V(s')) \tag{3.1}$$

$$Q(s,a) = \max_a (R + \gamma Q(s',a')) \tag{3.2}$$

$$R = R(s,a,s') \tag{3.3}$$

3.2 Q-Learning

Q-Learning (WATKINS; DAYAN (1992)) is one of the basics algorithms of Reinforcement Learning. It works based on a matrix called *Q-Table* where the columns represents the actions that the agent can take and the rows represents the set S of states. The *Q-Table* is updated for each step based on the Equation 3.4 which is called *Q-Function*, an adaption of the Bellman's Equation. See Figure 3.3. The algorithm does the work when the environment has a small discretized state space, otherwise it would take a very long time to the matrix converge. Other problem of the technique is the keeping of a local maximum due to the lack of generalisation of states.

$$\underbrace{Q(s,a)}_{\text{Q-Value}} = Q(s,a) + \underbrace{\alpha}_{\text{Learning rate}}[\underbrace{R(s,a)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount rate}} \overbrace{\max Q'(s',a')}^{\text{Maximum predicted reward, given new state and all possible actions}} - Q(s,a)] \tag{3.4}$$

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Figure 3.3: Example of Q-Table. By LearnDataSci - Own work - Article, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=69947708, accessed on 15/09/2019.

3.3 Deep Q Network

Deep Q Network (DQN) is the Deep Learning approach for Q-Learning created by MNIH et al. (2013). Although basic (comparing with today's techniques), it opened the eyes for Reinforcement Learning achieving impressive results with Atari's games. DQN solves the problems of Q-Learning using a non-linear function (Neural Networks) as a $Q*$ approximator. The loss function of the DQN changes every iteration i and is derived from the Bellman's Equation. See Figure 3.4. θ represents the network's weights and $\rho(s,a)$ is the probability distribution over states and actions, also called behaviour distribution. DQN's algorithm is in the Figure 3.5.

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

Figure 3.4: $L_i(\theta_i)$ as loss function of DQN's Neural Networks.

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise state s_t
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Set $s_{t+1} = s_t$
    Sample random minibatch of transitions (s_t, a_t, r_t, s_{t+1}) from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } s_{t+1} \\ r_j + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) & \text{for non-terminal } s_{t+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s_t, a_j; \theta))^2$
  end for
end for

Figure 3.5: Deep Q Networks algorithm with Experience Replay.

3.4 Double Deep Q Networks

Deep Q Networks brings a maximization bias learning due to the $max'_a Q^*(s', a')$ part of the Q-Learning equation. In other words, it says that some Q-Values are more valuables than it should. And such overestimation can be problematic. VAN HASSELT; GUEZ; SILVER (2015) brought the idea of Double Deep Q Networks to cure the overestimation problem based on the cure for the same problem in Q-Learning with Double Q-Learning, HASSELT (2010).

VAN HASSELT; GUEZ; SILVER (2015) proposes a solution involving two $Q*$ estimators, one to update the other with certain rate of averaging τ. Using it, the *Q-Value* will be unbiased by the opposite estimator of the action selector. The algorithm proposed is in Figure 3.6. See Figure 3.7 for the results.

Initialize primary network Q_θ, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
for each iteration **do**
  for each environment step **do**
    Observe state s_t and select $a_t \sim \pi(a_t, s_t)$
    Execute a_t and observe next state s_{t+1} and reward $r_t = R(s_t, a_t)$
    Store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
  for each update step **do**
    sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
    Compute target Q value:
      $Q^*(s_t, a_t) \approx r_t + \gamma\ Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
    Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
    Update target network parameters:
      $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$

Figure 3.6: Double DQN algorithm. Image from https://towardsdatascience.com/double-deep-q-networks-905dd8325412, accessed on 15/11/2019.

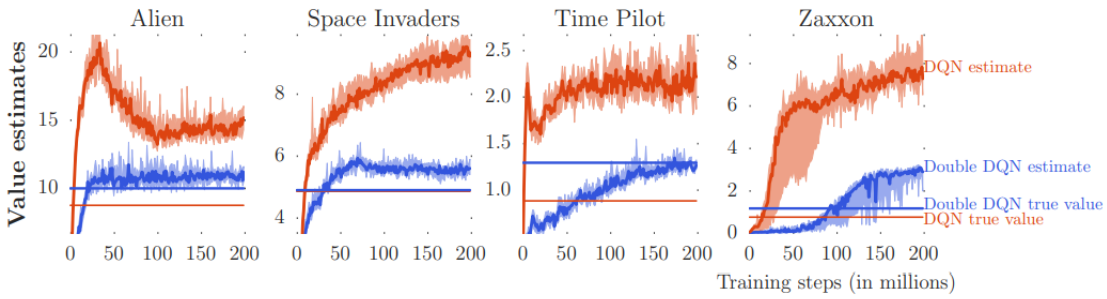

Figure 3.7: Q-Value in fuction of time for some Atari games: estimation and truth values for DQN and Double DQN. Image from VAN HASSELT; GUEZ; SILVER (2015).

3.5 Dueling Double Deep Q Networks

Dueling Double Deep Q Networks (DDQN) brings the idea of Double DQN and a plus. WANG; FREITAS; LANCTOT (2015) propose a dueling architecture for the final layers of the Neural Networks. It separates explicitly the state values and state-dependent action advantage via two different Neural Networks.

The motivation of the technique is that some environments do not depend fully of an action, sometimes if the agent does not do some action or does a single action, it takes more reward than taking multiple actions. It applies very well on our multi-agent context, for example if the right winger is intercepting the ball, it's more valuable to the left winger stays on it's position than intercepting.

The new *Q-Function* will be naively described as Equation 3.5. Due to identifiability problem of sum, the back propagation can not tell what is $V(s) or A(s,a)$, so WANG; FREITAS; LANCTOT (2015) propose the Equation 3.6 to train the agent.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) \tag{3.5}$$

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{A}\sum_{a'} A(s,a';\theta,\alpha)) \tag{3.6}$$

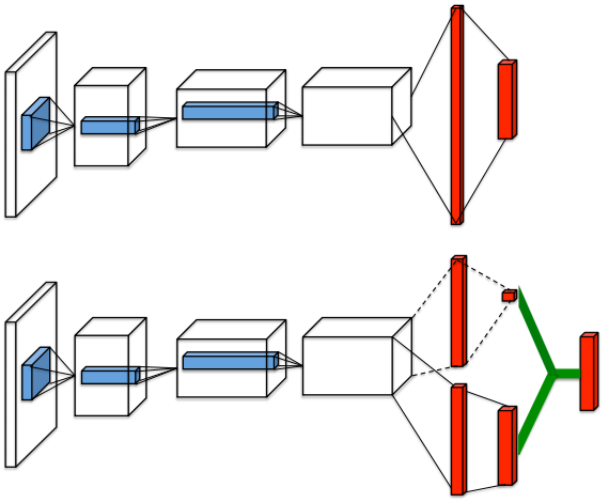

Figure 3.8: Deep Q Networks architecture against Dueling Deep Q Networks architecture. Image from WANG; FREITAS; LANCTOT (2015)

3.6 Deep Deterministic Policy Gradient

There is another branch of RL that studies the Policies and Values themselves. It is like the idea of a DDQN, where there is a Critic and an Actor algorithm, see Figure 3.9. Based on that architecture, LILLICRAP et al. (2015) develop the Deep Deterministic Policy Gradient (DDPG). The DDPG tries to learn an optimal deterministic policy μ^*.

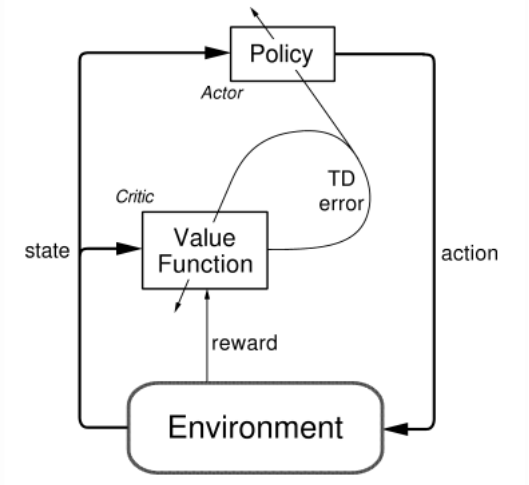

Figure 3.9: Illustration of an Actor Critic system. Image from SUTTON; BARTO (2011).

The input of the Actor is the current state and the output is a single real value representing an action chosen from a continuous action space. The output of the Critic is $V(s)$ where s is the current state. See Figure 3.10 to check DDPG's algorithm.

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2)$
    Update the actor policy using the sampled gradient:

$$\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 3.10: Deep Deterministic Policy Gradient algorithm. Image from LILLICRAP et al. (2015).

4

Background

In this chapter we introduce techniques described in Team Description Papers of the ROBOCUP (2019) and papers about Soccer Simulation 2D. In Section 4.1 we present deterministic techniques for defensive agents. Also in this chapter, in Section 4.2 we show Reinforcement Learning techniques applied to defensive.

4.1 Deterministic Agents

In this section we shall explain some successful deterministic techniques applied to Soccer Simulation 2D league. Most of them are very useful as baseline or the teams actual defense line.

4.1.1 Marlik2011 Defense

TAVAFI et al. (2011) created one of the best Defensive behaviour presented in the Soccer Simulation 2D League used since then in great teams like Fractals2019, PROKOPENKO; WANG (2019). It was combined by three high-level actions: Cross Mark, Play_on Mark and Block. Since they released the code, the RoboCIn team included those actions to their agent and it raised up to 60% of the successful defenses.

4.1.1.1 Cross Mark

This behaviour occurs when an opponent reaches near one of the defensive corner flags, usually when -36 > Ball-X > -53 and 20 < |Ball-Y| < 34. The main idea of the algorithm is to position the marker between the nearest attacker and the ball to reach the ball faster. When the number of defenders is greater than the attackers, man-to-man marking will be done and the rest of the markers will guard the goal for probable shots. When attackers have an advantage, the three midfielders (usually numbers 6, 7 and 8) will try to mark opponents depending on their stamina. In Figure 4.1 we can see an example of the behaviour.

Figure 4.1: Cross Mark behaviour. Observe the positioning of numbers 3, 4, 7 and 8 how the get in "front" of the attackers. Image from TAVAFI et al. (2011).

4.1.1.2 Play_on Mark

This type of marking is to avoid through passes and it is only applied on defenders, usually numbers 2, 3, 4 and 5. The idea is to disarm the attack when they are trying to break the defense line. Depending on attackers position and the defenders formation, the marker will choose one of the attackers to stay "behind" and then block a possible through pass. In Figure 4.2 and 4.3 we can see examples of the behaviour.

Figure 4.2: Play_on Mark behaviour. Image from TAVAFI et al. (2011).

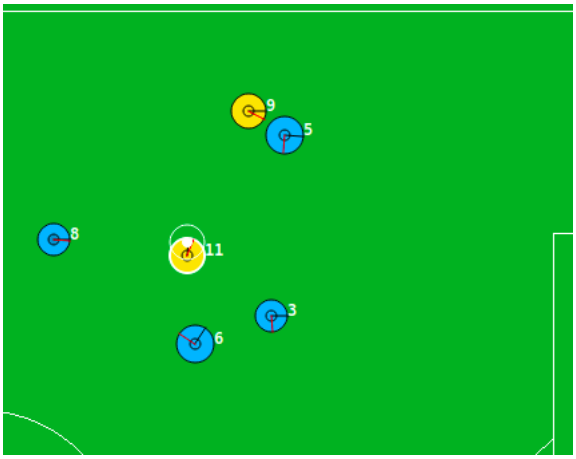

Figure 4.3: Play_on Mark behaviour. Observe the positioning of numbers 5 and 3 ready to block a through pass.

4.1.1.3 Block

Block is the most efficient technique of Marlik2011. Including only it in RoboCIn's code, the behaviour reduced from an average of 7 goals taken from HELIOS2018 to an average of 2 goals taken.

It has two important parts: moving to the best block point and decide how to act then. When the defender is the near from the ball, the best block point is calculated by a prediction of opponent's future dribble target. When the agent is too far from the attacker in possession, it marks the nearest opponent, see Figure 4.4. Once the agent have reached the best position to block the attack, the defender will decide if it will continue doing the block, intercept the ball or push the attackers back, see Figure 4.5.

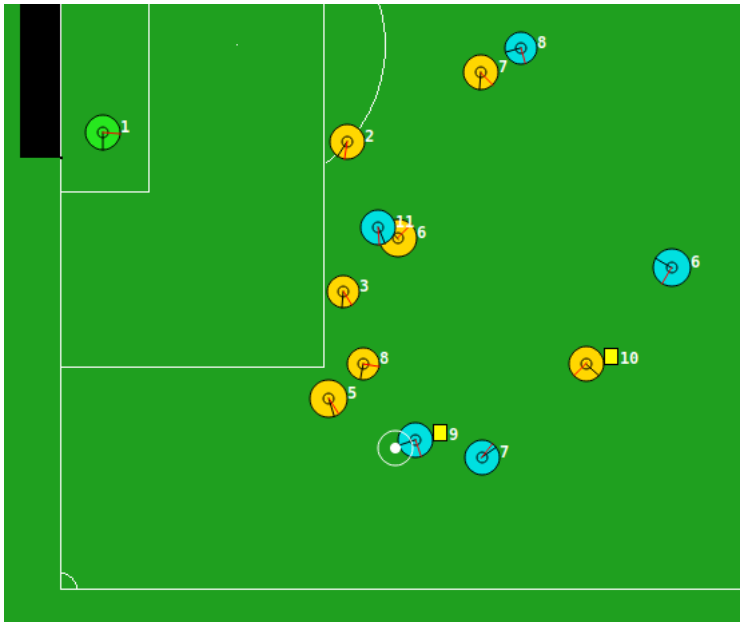

Figure 4.4: Block behaviour. Observe the yellow team positioning of numbers 6, 7 and 10 ready to block the pass.

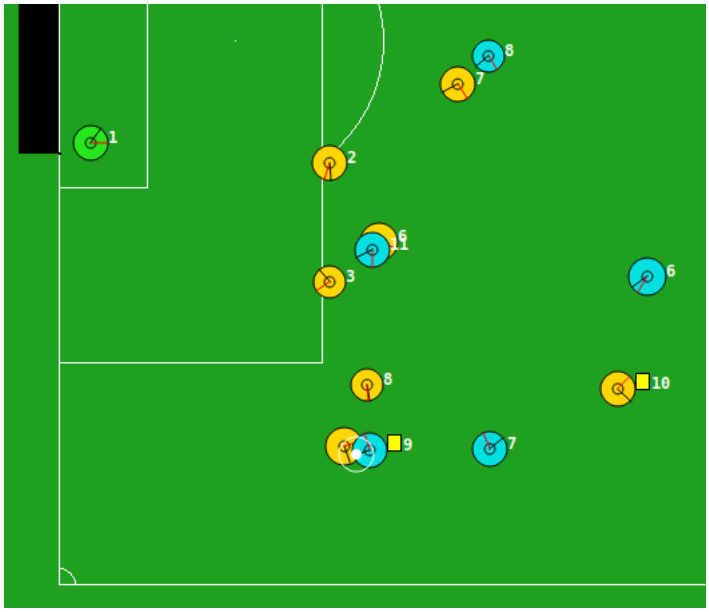

Figure 4.5: Block behaviour. Observe the positioning of number 5 going to intercept the ball.

4.1.2 WrightEagle2014 Defense

The WrightEagle team, ZHANG et al. (2014), did an interesting research on defenses areas and ranges. Their algorithm draws defensible areas based on defender formation's position and a number K opponents attackers. K represents how many opponents can represent better the attack or the most valuated weights calculated by the algorithm. The K weights are summed up and normalized and then the area with greater weight is chosen to be defended. In figure 4.6 we can see an example of defense nets.

4.1.3 Gliders2d Defense

PROKOPENKO; WANG (2018) showed a simple defense improving with Gliders2D that caused their winning of RoboCup 2016. They fine tuned the pressing variable considering the role of the agent, the position of the ball and the opponent team, see Figure 4.7.

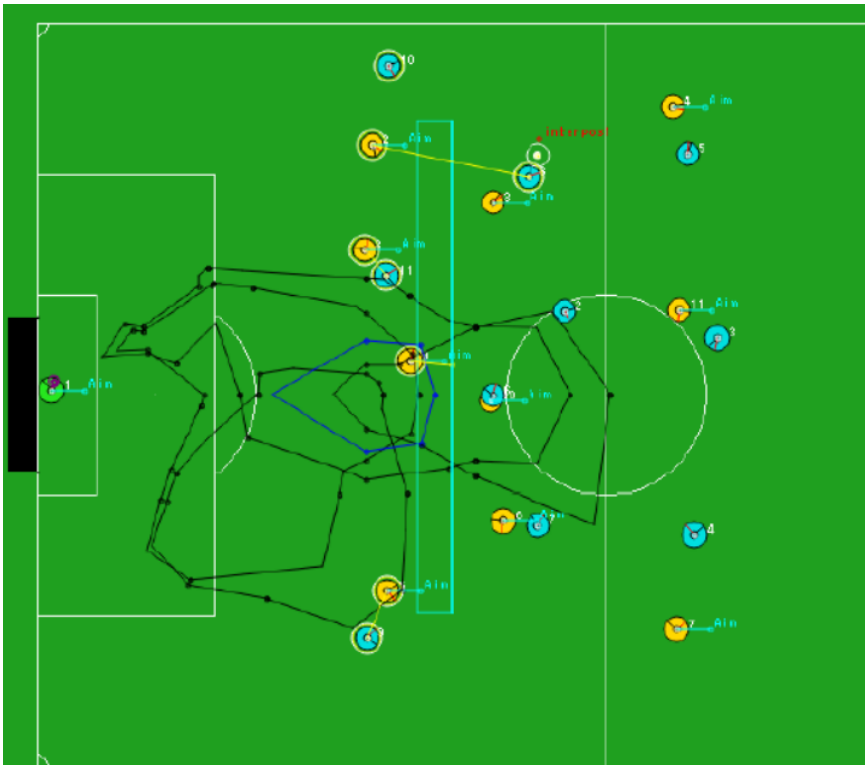

Figure 4.6: WrightEagle's defensive net. The blue area represents the area to be defended and the blacks, all calculated areas. Image from ZHANG et al. (2014).

```
// G2d: pressing
     int pressing = 13;

     if ( role >= 6 && role <= 8 && wm.ball().pos().x > -30.0
            && wm.self().pos().x < 10.0 )
            pressing = 7;

     if ( fabs(wm.ball().pos().y) > 22.0 && wm.ball().pos().x < 0.0
            && wm.ball().pos().x > -36.5 && (role == 4 || role == 5) )
            pressing = 23;

     if (helios2018)
            pressing = 4;

     if ( ! wm.existKickableTeammate()
            && ( self_min <= 3
                   || ( self_min <= mate_min
                          && self_min < opp_min + pressing )
                )
          )
     {
          Body_Intercept().execute( agent );
          ...
     }
```

Figure 4.7: Gliders' piece of code showing the fine tuning of pressing variable. Image from PROKOPENKO; WANG (2018).

4.2 Reinforcement Learning Agents

In this section we show some algorithms that introduced Reinforcement Learning into the SS2D. All teams here mentioned had a great performance in their years of competition.

4.2.1 Brainstormers2d

The pilot of Reinforcement and Deep Reinforcement Learning was Brainstormers2d. Since 2002 they invest in RL algorithms (RIEDMILLER et al. (2002)) but here we mention only the techniques that gave them the first place in the competition: Intercept ball with RL and the NeuroHassle approach.

4.2.1.1 Intercept Ball Task

GABEL; RIEDMILLER (2006) formalized their technique as a Markov Decision Process, BERTSEKAS; TSITSIKLIS (1996), where the State space was represented by the ball's velocity X and Y directions, agent's velocity X and Y directions, the distance between the agent and the ball and the relative angle between ball and agent. The actions available to the agent were turn and dash commands. They discretized the Environment Space into a grid of 6×10^5 cells. They also considered a noise-free environment to do the experiments.

As GABEL; RIEDMILLER (2006) say, the technique outperformed their previous NI'02 Intercept Model and takes only one more cycle to intercept the ball then the Model Based algorithm described in the paper.

4.2.1.2 The NeuroHassle Approach

Brainstormers did a disruptive job introducing Neural Reinforcement Learning in the Soccer Simulation 2D league in 2006. GABEL; RIEDMILLER; TROST (2009) technique (it was develop in 2006 but the paper was only published in 2009) shows that a well modeled reinforcement learning system can outperform a deterministic one. They restricted their state space into a 9 dimensions one considering:

- Distance d between defender and the attacker
- Velocity (v_x and v_y component) defender
- Absolute value of the attacker's velocity
- Position (b_x and b_y component) of the ball
- Defender's body angle relative to the attacker's position
- Attacker's body angle relative to his direction towards the goal

- Value of the angle $\gamma = \angle GAD$ with G as position of the goal, A as position of the attacker, and D as the position of the defender

They put as actions a discretized space of 76 dimensions where the first 40 were the dash command varying the power from -100 to 100 with a 5 step and the last 36 were turn command varying the angle from -180 to 180 with a 10 degree step.

To specialize the agent, they created a set of *training situations* S ($|S| = 5000$ described in Figure 4.8 and defined four *training regions* shown in Figure 4.9.

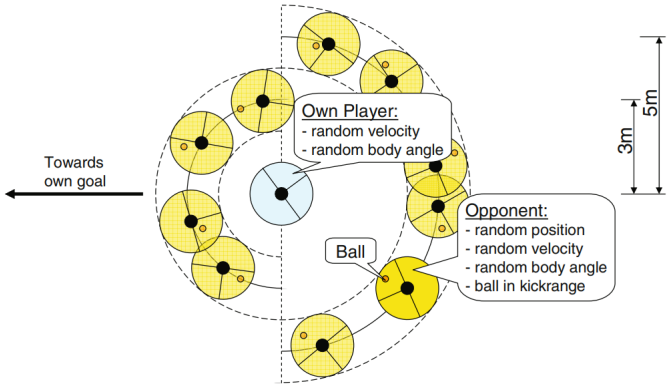

Figure 4.8: Neurohassle's *training situations*. Image from GABEL; RIEDMILLER; TROST (2009).

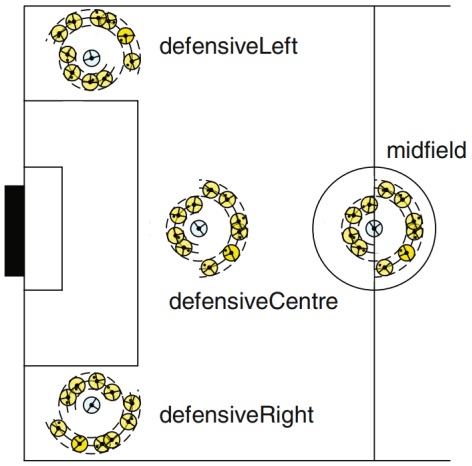

Figure 4.9: Neurohassle's *training regions*. Image from GABEL; RIEDMILLER; TROST (2009).

They did a simple reward function:

- For each step without success: small negative reward
- Failure: Large negative reward
- Success: Large positive reward

The Temporal Difference (TD) loss was calculated as follows:

$$V(s_k) = (1-\alpha) * V(s_k) + \alpha * ret(s_k)$$
$$ret(s_k) = \sum_{j=k}^{N} r(s_k, \pi(s_k))$$

Where $V(s_k)$ means the state value function, $ret(s_k)$ the summed rewards following state s_k and α the learning rate. They employed a multi-layer perceptron (MLP) neural networks HAYKIN (1998) with a 9:18:1-topology to train the agent. See the results on Figure 4.10 and Figure 4.11.

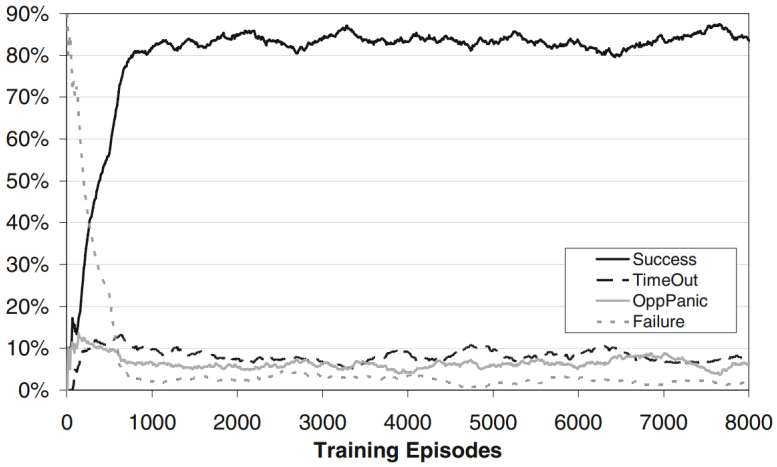

Figure 4.10: Example of Learning Curve for Learning to Hassle. Image from GABEL; RIEDMILLER; TROST (2009).

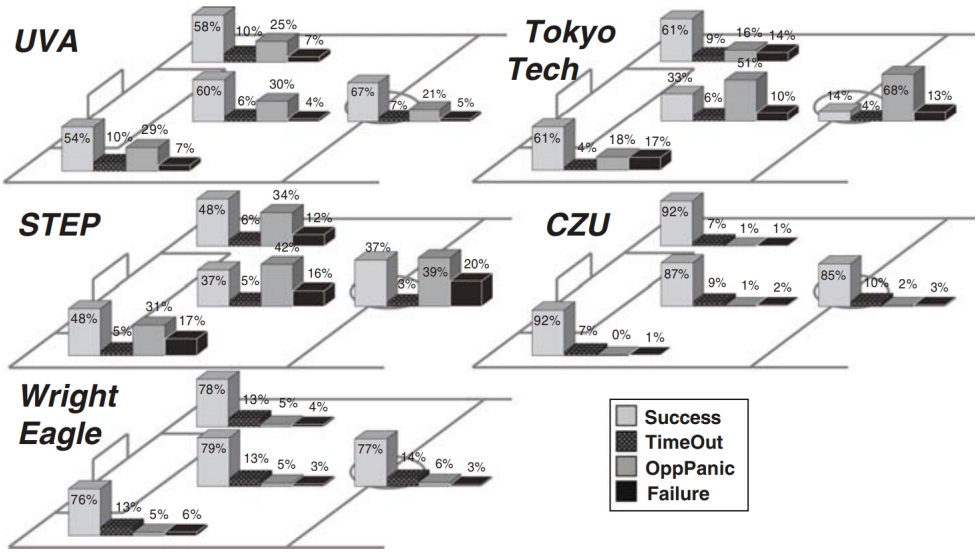

Figure 4.11: *Training regions* analysis for some teams from RoboCup2006. Image from GABEL; RIEDMILLER; TROST (2009).

4.2.2 Cyrus2019

ZARE et al. (2019) did an interesting job with a single agent cooperating with the goalie against an attacker. They did not specified the space features that they used, but they did tell us about HAUSKNECHT (2015) environment, so we assumed that they used the high level state features from HFO. The actions they chose were block (KHAYAMI et al. (2014), a Cyrus' implementation of TAVAFI et al. (2011)'s Marlik block), intercept ball, cooperation with goalie (the defender) and move (execute the movement according to the formation file). See Figure 4.12, 4.13, 4.14 for the high level actions.

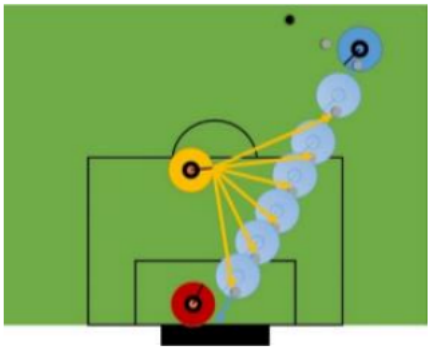

Figure 4.12: Cyrus' Block movement. Image from ZARE et al. (2019).

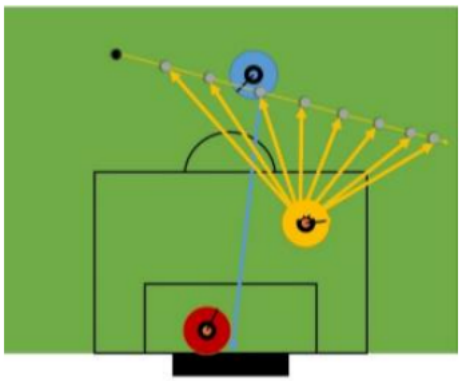

Figure 4.13: Intercept movement. Image from ZARE et al. (2019).

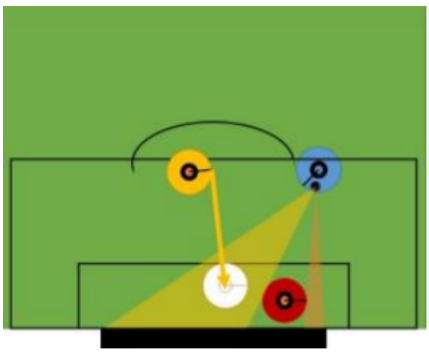

Figure 4.14: Cyrus' Goalie Assistant movement. Image from ZARE et al. (2019).

The reward modeling is described in Figure 4.15. They used a Deep Deterministic Policy Gradient (LILLICRAP et al. (2015)) to train the agent with the TD function explained in Section 3.6. The results are shown in Figure 4.16

Passing of 150 cycles	10
Ball leaving the field	10
Gaining possession by the goalie or defenders	10
A goal being conceded	-10

Figure 4.15: Cyrus' reward model. Image from ZARE et al. (2019).

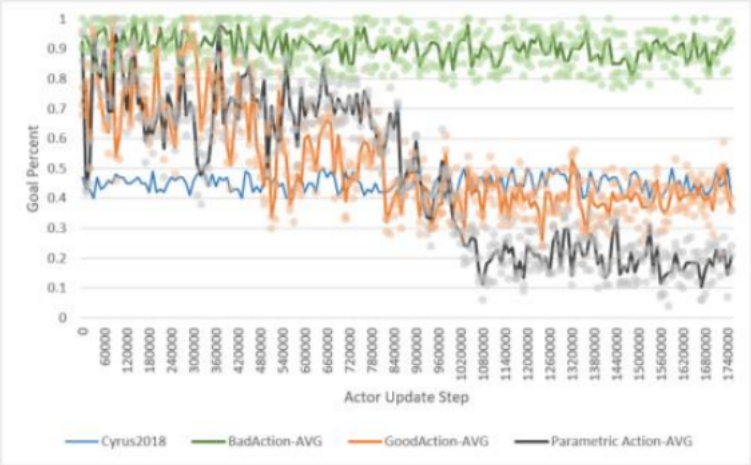

Figure 4.16: Cyrus' DDPG results. The best result is choosing a action and then CYRUS2018 (ZARE et al. (2018)) doing that action. Image from ZARE et al. (2019).

5

Environment

In this chapter we discuss about the simulating environment that we used or developed in our experiments. It is divided in 3 sections: Server's Referee: the changes we made on HFO's referee; Feature set: the features we chose to receive from HFO's environment; Actions: the output of our Neural Networks; Reward System: the learning conditions to our Neural Networks.

5.1 HFO Server's Referee

The Half Field Offensive environment has a principle: start an episode with the ball positioned in the middle field. That is not applicable to our real domain. We did some changes in HFO's Server due to this incompatibility of domains:

- Half Field Offensive's referee restricts all agents on the right side of the field. Our problem is more complex due to defending on opponent's side as well. Our referee covers the whole field.
- Given two areas O and T, opponents spawns randomly on O and the same for teammates on T. We need a spawn similar to a real attack situation in Soccer Simulation 2D. For the defender team We fixed specifics X axis for types of agent and randomizes the Y axis. The midfielders starts in front of the defenders. For the attacking team, is the same logic for attackers and midfielders. We decided to do not randomize the Y axis and fix it on 0 for all attackers.
- The ball always starts near the midfielders of the attacking team in our environment. Doing so, We simulate a counterattack of the opponent.
- We use a noise-free server due to the agents initial learning. Once well trained, it can train in a noisy system.

5.2 Feature Set

HFO's high-level features set returns many features that is not relevant for our problem, such as opening angle to opponent's goal or pass opening angle to a teammate. We decided to remove those variables for the models understand more easier what to do. Another change was in the normalization of the features. The original environment returned normalized features in relation due to the half field problem. Once our problem is more comprehensive and the agents are in the same space and the features are strict, We maintained the without normalization.

Let T denote the number of teammates in game and O the number of opponents. There are a total of $10+3T+2O+1$ high-level features in our environment.

0. **X position** - The agent's x-position on the field.
1. **Y position** - The agent's y-position on the field.
2. **Orientation** - The global direction that the agent is facing.
3. **Ball X** - The ball's x-position on the field.
4. **Ball Y** - The ball's y-position on the field.
5. **Able to Kick** - Boolean indicating if the agent can kick the ball.
6. **Goal Center Proximity** - Agent's proximity to the center of the goal.
7. **Goal Center Angle** - Angle from the agent to the center of the goal.
8. **Proximity to Opponent** - If an opponent is present, proximity to the closest opponent. Invalid if there are no opponents.

T **Proximity from Teammate i to Opponent** - For each teammate i: the proximity from the teammate to the closest opponent. This feature is invalid if there are no opponents or if teammates are present but not detected.

$2T$ **X, Y of Teammates** - For each teammate: the x-position, y-position.

$2O$ **X, Y of Opponents** - For each opponent: the x-position, y-position.

$+1$ **Interceptable** - Whether the agent can intercept the ball or not.

5.3 Actions

Based on ZARE et al. (2019), We chose four actions:

1. Move: Performs the basic move, going to the position according to the formation file.
2. Go to ball: Performs an interception move, tackling the opponent when it can.
3. Defend Goal: Goes to the circumcenter position of the triangle goalie position, right or left goal post position and attacker position.
4. Block: Performs TAVAFI et al. (2011)'s Marlik Block.

We decided to choose Marlik's block over Cyrus' due to Cyrus' harassment and effectiveness. Cyrus' block is too advanced and that induces great chances of a dribble and then a goal.

5.4 Reward System

The reward system is also very similar to ZARE et al. (2019)'s. We just removed the cycles condition. We tried to adapt OLIVEIRA LIMA JúNIOR; BASANI (2019)'s ball potential equation to Soccer Simulation 2d domain to be a reward as well but it did not fit in our experiments.

6

Experiments and Results

6.1 Experiments

For the experiments, we used, for the first 100K training episodes of the three techniques, Helios2013's goalie and Helios2013's offensive and for 10K we changed the goalie to RoboCIn2019's. We used MNIH et al. (2013)'s technique to stack 32 states, therefore a state of shape [32x1x16]. The Networks' and memory's parameters is in Figure 6.1. For DQN and DDQN we used the ε-greedy technique with ε's maximum at 1.0 and minimum at 0.01. For DDPG we added a noise of 0.1 in the action. The full code is available in `https://github.com/goncamateus/graduationMgm`.

The Neural Network's topology were (all using Rectified Linear Unit activation function AGARAP (2018)):

- Deep Q Networks: 512:128:256:4.
- Duelling Double Deep Q Networks:

  Advantage - [512:128:128:4]; Value - [512:128:128:1].
- Deep Deterministic Policy Gradient:

  Actor - [512:256:128] with output of shape [4x1]; Critic - [512:256:128:1].

```
# misc agent variables
self.GAMMA = 0.99
self.tau = 1e-2
self.LR = 1e-4

# memory
self.TARGET_NET_UPDATE_FREQ = 1000
self.EXP_REPLAY_SIZE = 100000
self.BATCH_SIZE = 32
```

Figure 6.1: Neural Networks' and Memory's parameters.

6.2 Results

In this section we show the graphs and tables of training and tests. As said in Section 6.1, the training was 100K episodes with Helios2013's goalie and 10K with RoboCIn2019's goalie. The whole training against Helios2013's attacker.

6.2.1 Training

During the training we could see some interesting behaviours of each learning algorithm. The DQN agent turned out to be the most aggressive agent. Before episode 10K, it started to chase the attacker due to Intercept action. As there is no fault in our domain, the Network found this strategy but it is not the best thing to be done during a real game. Figure 6.2 shows it's performance with rewards.

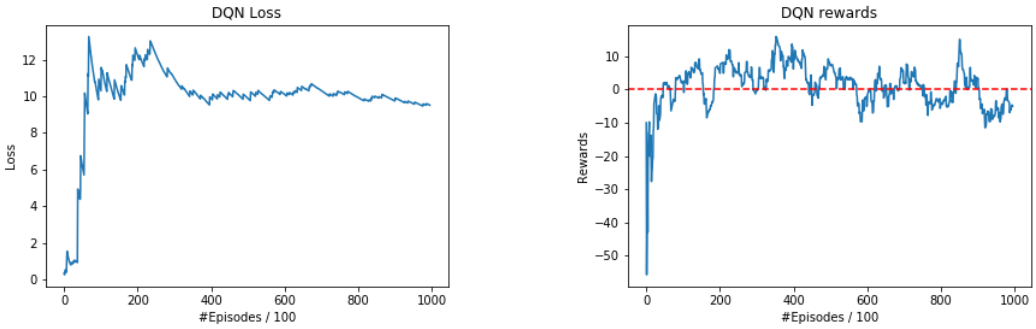

Figure 6.2: Deep Q Network results. Loss convergence and Rewards in function of number of episodes.

The DDPG agent found a policy very similar to RoboCIn2019's defender but a bit more aggressive. The agent blocks the shoot of the attacker most of time but intercept in the right gap of the attacker. Sometimes the episode takes too long due to the attacker's incapability to decide what to do, that would lead to a fault in real game and the defender would win the ball. Figure 6.3 shows the performance of the agent. As we can see in the Actor Loss, sometime before the episode 20K it found that policy and converged to it.

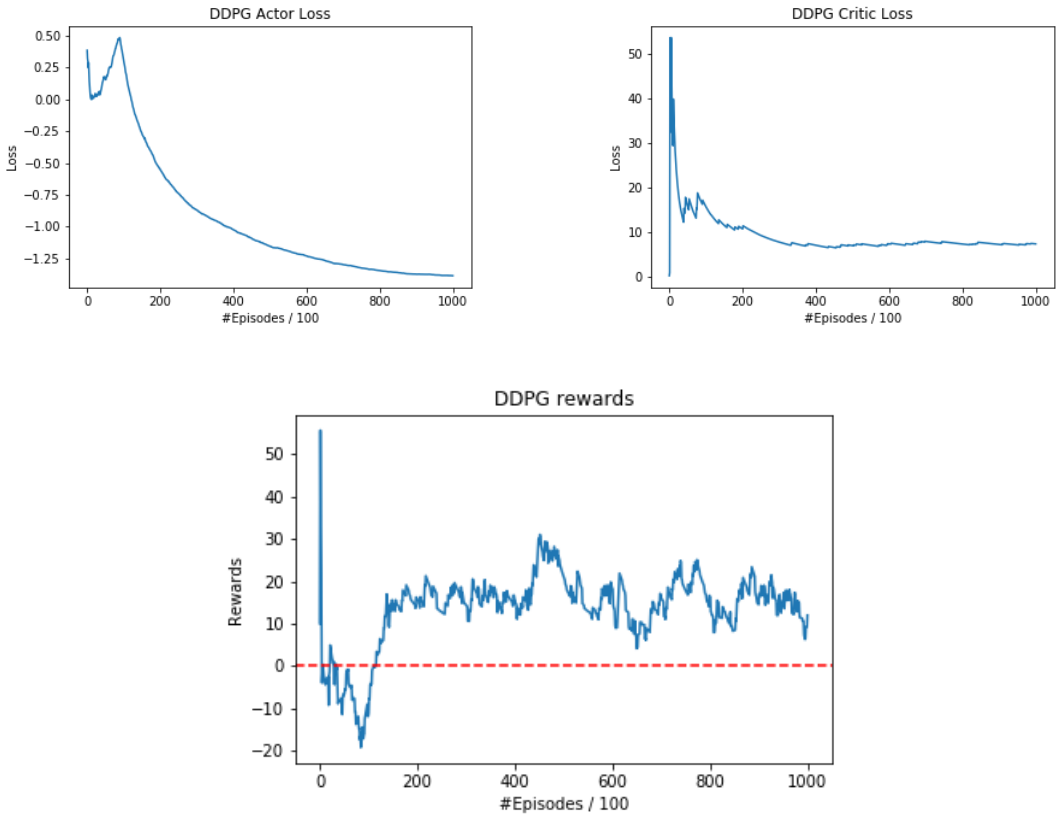

Figure 6.3: Deep Deterministic Policy Gradient results. Loss convergence of Actor and Critic networks and Rewards in function of number of episodes.

The DDQN agent took an intermediate behaviour. Sometimes it acted as DQN's agent but most of time it acted just like DDPG's. The episodes that it acted too aggressive, the attacker shoots to the goal and scores, most of time. Figure 6.4 shows the performance of the agent.

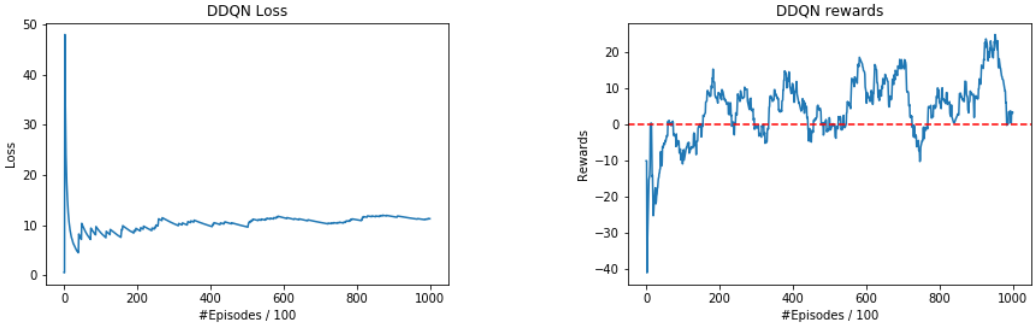

Figure 6.4: Duelling Double Deep Q Network results. Loss convergence and Rewards in function of number of episodes.

6.2.2 Tests

To test, we wanted to compare with the Helios2013 and RoboCIn2019. Helios2013 was chosen because HFO already had support for the team. RoboCIn2019 was chosen because we

want to put the best agent in the team. We tested with both attackers and both goalies for 3000 episodes each combination. Table 6.1 shows the baseline with Helios2013's and RoboCIn2019's agents. Tables 6.2 and 6.3 show the results of the DRL agents with the cooperating with the deterministic goalies against each attacker.

	Helios2013	RoboCIn2019
Helios2013	77,5%	77.4%
RoboCIn2019	71%	53.3%

Table 6.1: Result baseline. Rows are the defenders and Columns are the attackers of each original team. Percentage of successful defenses for 3000 episodes.

	Helios2013	RoboCIn2019
DQN 110k training	63%	76.2%
DDQN 110k training	75.1%	72.4%
DDPG 110k training	82.1%	95%

Table 6.2: Tests for 3000 episodes. Rows are the DRL defenders with Helios2013's goalie and Columns are the attackers. Percentage of successful defenses for 3000 episodes.

	Helios2013	RoboCIn2019
DQN 110k training	51.3%	64.2%
DDQN 110k training	62.9%	83.6%
DDPG 110k training	70%	86.7%

Table 6.3: Tests for 3000 episodes. Rows are the DRL defenders with RoboCIn2019's goalie and Columns are the attackers. Percentage of successful defenses for 3000 episodes.

7
Conclusion

The Defense Line is one of the most important task in the Soccer Simulation 2D league. There are many techniques implemented by other teams using deterministic policies that has a great performance. These algorithms came through deep studies in human behaviours in real soccer based on timing of intercept, positioning and blocking passes.

The main problem of those policies is that it is predictable therefore there is a way to break that line. Our work's main intent is to create unpredictable agent to help the goalie and do not receive goals. This work compares 3 Deep Reinforcement Learning algorithms with 2 deterministic teams with great rate of defenses in the SS2D.

We can conclude that DRL has a great potential of applicability in Soccer Simulation 2D due to it's elasticity to learn about the opponent and teammates. We can see in Tables 6.2 and 6.3 that the agent has to learn more about RoboCIn2019's goalie than the Helios2013's. DDPG had a great performance in our experiments performing a great percentage of defenses, being better to use rather than the deterministic teams. Also DQN and DDQN had a good performance but stays behind the deterministic algorithms. Based on the results, probably a more specific training for each team will lead to a better performance of each algorithm.

References

AGARAP, A. F. **Deep Learning using Rectified Linear Units (ReLU)**. 2018.

AKIYAMA, H.; NAKASHIMA, T. HELIOS base: an open source package for the robocup soccer 2d simulation. In: **Anais** 2014. v.8371, p.528–535.

AKIYAMA1, H. et al. HELIOS2016: team description paper. , 2016.

BELLMAN, R. A Markovian decision process. **Journal of mathematics and mechanics**, p.679–684, 1957.

BERTSEKAS, D. P.; TSITSIKLIS, J. N. **Neuro-dynamic programming**. Athena Scientific Belmont, MA, 1996. v.5.

Brockman, G. et al. OpenAI Gym. **arXiv e-prints**, p.arXiv:1606.01540, Jun 2016.

CHEN, M. et al. **RoboCup Soccer Simulator Documentation**. 2003. https://rcsoccersim.github.io/manual/soccerserver.html.

DeepMind. **DeepMind**. 2014. https://deepmind.com/.

Dota2. **Dota2**. 2011. http://br.dota2.com/?l=brazilian.

GABEL, T.; RIEDMILLER, M. Learning a Partial Behavior for a Competitive Robotic Soccer Agent. **KI**, v.20, p.18–23, 01 2006.

GABEL, T.; RIEDMILLER, M.; TROST, F. A Case Study on Improving Defense Behavior in Soccer Simulation 2D: the neurohassle approach. In: ROBOCUP 2008: ROBOT SOCCER WORLD CUP XII, Berlin, Heidelberg. **Anais** Springer Berlin Heidelberg, 2009. p.61–72.

HASSELT, H. V. Double Q-learning. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS. **Anais** 2010. p.2613–2621.

HAUSKNECHT, M. **RoboCup 2D Half Field Offense**. 2015. https://github.com/LARG/HFO.

HAUSKNECHT, M.; STONE, P. **Deep Reinforcement Learning in Parameterized Action Space**. 2015.

Upper Saddle River, NJ, USA Prentice Hall PTR, 1998.

ISO. **ISO/IEC 14882:1998**: Programming languages — C++. 1998. 732p. Available in electronic form for online purchase at `http://webstore.ansi.org/` and `http://www.cssinfo.com/`.

KHAYAMI, R. et al. Cyrus 2D Simulation 2014 Team Description Paper. , 2014.

KITANO, H. et al. RoboCup: a challenge problem for ai. **AI Magazine**, v.18, p.73–85, 01 1997.

KYRYLOV, V.; BERGMAN, D. S.; GREBER, M. Multi-criteria optimization of ball passing in simulated soccer. **Journal of Multi-Criteria Decision Analysis**, v.13, n.2-3, p.103–113, 2005.

LILLICRAP, T. P. et al. **Continuous control with deep reinforcement learning**. 2015.

MNIH, V. et al. Playing Atari with Deep Reinforcement Learning. **CoRR**, v.abs/1312.5602, 2013.

OLIVEIRA LIMA JúNIOR, J. N. de; BASANI, H. Sim-to-Real: adapting the control for robots playing soccer in the real world. , 2019.

OpenAI. **OpenAI Five**. 2018.

PROKOPENKO, M.; WANG, P. **Gliders2d**: source code base for robocup 2d soccer simulation league. 2018.

PROKOPENKO, M.; WANG, P. **Fractals2019**: combinatorial optimisation with dynamic constraint annealing. 2019.

RIEDMILLER, M. et al. Brainstormers 2002 - Team Description Paper. , 2002.

ROBOCIN. **RoboCIn Team**. 2019. https://robocin.com.br/.

ROBOCUP. **RoboCup Soccer Simulation Server**. 2018. https://github.com/rcsoccersim/rcssserver.

ROBOCUP. **RoboCup Federation**. 2019. https://www.robocup.org/.

SILVER, D. et al. Mastering the game of Go with deep neural networks and tree search. **Nature**, v.529, 2016.

STONE, P.; SUTTON, R. S.; KUHLMANN, G. Reinforcement learning for robocup soccer keepaway. **Adaptive Behavior**, v.13, n.3, p.165–188, 2005.

SUTTON, R. S.; BARTO, A. G. Reinforcement learning: an introduction. , 2011.

TAVAFI, A. et al. MarliK 2011 Soccer 2D Simulation Team Description Paper. , 2011.

VAN HASSELT, H.; GUEZ, A.; SILVER, D. Deep reinforcement learning with double q-learning. In: THIRTIETH AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE. **Anais** 2015.

VAN ROSSUM, G.; DRAKE JR, F. L. **Python tutorial**. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

WANG, Z.; FREITAS, N. de; LANCTOT, M. Dueling Network Architectures for Deep Reinforcement Learning. **CoRR**, v.abs/1511.06581, 2015.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. **Machine Learning**, v.8, n.3, p.279–292, May 1992.

ZARE, N. et al. Cyrus 2D Simulation 2018 Team Description Paper. , 2018.

ZARE, N. et al. Cyrus 2D Simulation 2019 Team Description Paper. , 2019.

ZHANG, H. et al. WrightEagle 2D Soccer Simulation 2014. , 2014.