

Machine Learning for Smartphone Games

Dissertation presented by
Antoine VAN MALLEGHEM , Nicolas VAN WALLENDael

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Pierre DUPONT

Reader(s)
Pierre DUPONT, Vincent BRANDERS , Jérôme PAUL

Academic year 2015-2016

Acknowledgements

We'd like to thank our assistant, V. Branders, for his precious advice during this school year as well as his long and complete mail responses. Also, we have to thank our thesis director, P. Dupont, for letting us work on this thesis and for its comments during our work. We can finally finish by thank all the people of the Intel Room for some of your insights and for putting up with our many debates.

Abstract

This thesis concerns the creation of an artificial player for a smartphone game able to learn the rules and how to win by itself thanks to Machine Learning techniques. The chosen game is an adaptation of the well known "rock-paper-scissors" game.

The first step of this work is to clearly define the game with frameworks game theory and Machine Learning and to analyze the imposed constraint which is the smartphone hardware.

The second step is to create the agent according to these frameworks by improving and assessing its performances. This part will also include the final results obtained and show that it can compete against a human.

Contents

I	Machine learning, smartphones and games	2
1	Artificial Intelligence and Games	4
1.1	AI vs human	4
1.2	Early stage of AI	5
1.3	When the human cannot beat the machine anymore	7
1.4	Let the machine think	8
1.5	Machine Learning	8
1.6	Current milestones of AI	9
2	Smartphones era and machine learning	10
2.1	On device machine learning	11
2.2	Smartphone's hardware	12
2.2.1	Computational performances	12
2.2.2	Storage performances	14
2.3	Smartphones in action	16
2.3.1	Berkeley's experiment	16
2.3.2	GATECH's experiment	18
2.4	Application on real world problem	20
2.5	Conclusion	21
3	Game theory	22
3.1	Game theory notion	22
3.2	Basic definition	23
3.3	Game representation	24
3.3.1	Normal form	24
3.3.2	Extensive form	25
3.4	Type of games	27
3.4.1	Cooperative and Non Co-operative Games	27
3.4.2	Zero- and non-zero-sum games	27
3.4.3	Stochastic Games	28
3.5	Equilibrium and optimality concepts	29
3.5.1	Nash Equilibrium (NE)	29
3.5.2	Pareto optimal	30
3.6	Game theoretic models and Human nature	31
4	The game	32
4.1	A game ?	32
4.2	"Cowboy game"	33
4.3	Cowboy game as a ML problem	34

4.4	Game Theory applied to Cowboy	35
4.4.1	State 1	35
4.4.2	State 2	36
4.4.3	State 3	37
4.4.4	State 4	38
4.4.5	Interpretation	39
4.5	The mobile application	39
4.5.1	The application	39
4.5.2	The constraints	40
5	Conclusion	44
II	Algorithms and experiments	46
6	Markov decision process	48
6.1	From Markov chain to Markov decision process	48
6.1.1	Markov chain	48
6.1.2	Markov Decision Process	50
6.2	Drawback	53
7	Q-learning	54
7.1	Q-learning algorithm and Reinforcement learning	54
7.1.1	From Value Iteration to Q-learning	54
7.1.2	Reinforcement Learning	56
7.2	Experiments	57
7.2.1	Bots	57
7.2.2	Pseudo-code and adaptations	60
7.2.3	Performance assessment	61
7.3	Drawbacks	64
8	Opponent Modeling	65
8.1	From Q-learning to Opponent modeling	65
8.2	Experiments	66
8.2.1	Representation of a state	66
8.2.2	Performances of Opponent Modeling	72
8.3	Drawbacks	73
9	Exploration vs Exploitation dilemma	74
9.1	Introduce exploration by changing the learning policy	74
9.1.1	Greedy - "s s-36"	74
9.1.2	ϵ -greedy	74
9.1.3	Boltzmann distribution - B	75
9.2	Experiments	75
9.2.1	First experiment	75
9.2.2	Second experiment	81
9.3	Drawback	83

10 Opponent Modeling with fitting α and γ	85
10.1 Standard way to tune α and γ	85
10.2 Experiments	85
10.2.1 Influence of α and γ	86
10.2.2 A dynamic model with α and γ	86
10.2.3 Behavior after being trained	91
10.3 Drawback	91
11 Evolved Opponent Modeling with smoothing	93
11.1 Find a suitable smoothing	93
12 Final agent	95
12.1 The algorithm	95
12.2 Final performances	95
12.2.1 Performances against agents and bots	97
12.2.2 Performances against humans	106
13 Conclusion	110
14 Appendices	115
14.1 Game's action table	115
14.2 The simulator	115
14.2.1 Input and output	116
14.2.2 General implementation of each agent	117
14.3 Graphs associated with table 9.2	118
14.4 Heat maps of B-T0.5 against GSR, B-T0.1 and GBR	121
14.5 Graphs associated with table 12.1 and 12.3	127
14.6 Graphs associated with tables 12.2 and 12.4	132

List of Figures

1.1	Moore's Law [1]	6
1.2	State of the Chess AI at the end of the 50s. [2]	7
2.1	Smartphones ownership across the world. [3]	10
2.2	Data Mining Computing Resource Distribution: Processors (cores) vs Memory (GB). Circle size corresponds to the number of responses. [4]	12
2.3	Bar chart show MFLOPS evolution for Smartphones	13
2.4	Bar chart show MFLOPS evolution for video game consoles	13
2.5	Bar chart show MFLOPS evolution for Super Computers	14
2.6	Bar chart showing random (Rnd.) and Sequential (Seq.) write speed in mb/s [5] [6] [7]	15
2.7	Bar chart showing random (Rnd.) and Sequential (Seq.) read speed in mb/s [5] [6] [7]	15
2.8	Load time comparison for RF and CF for each device.	17
2.9	Comparing the runtimes of the connected components and PageRank algorithms, on the iPad mini and the Macbook Pro. [8]	19
2.10	Run time comparison for PageRank and Connected Components for each device.	20
3.1	Game representation	27
4.1	Game Design (1)	42
4.2	Game Design (2)	43
7.1	The reinforcement learning problem (from [9])	56
7.2	Q-learning vs Bot11	63
7.3	Game length of Q-learning vs Bot1	64
8.1	Opponent Modeling vs bot11	73
9.1	s s-36 and GBR vs Bot14	77
9.2	Humans vs smart agents	79
9.3	Humans vs smart agents	80
10.1	Heatmaps for the analyze of the influence of α and γ	88
12.1	Interesting graphs of the final agent against bots	104
12.2	Interesting graphs of the final agent against smart agents	106
12.3	Final agent vs human 1	107
12.4	Final agent vs human 2	108
14.1	Graphs associated with table 9.2	120

List of Tables

2.1	SVM analysis	16
2.2	Random forest analysis [10]	16
2.3	Collaborative filtering analysis [10]	17
3.1	Prisoner's Dilemma Payoff Matrix	25
4.1	No player has reloaded yet.	36
4.2	No player has sufficient power to overcome shield.	37
4.3	One player has sufficient power to overcome shield.	38
4.4	Both players have sufficient power to overcome shield.	39
7.1	Bot vs Bot win ratio.	59
7.2	A cowboy game between 2 players with, as state, a set of status	61
7.3	Win rate of basic Q-learning agent vs bots	62
8.1	Decision of each player at each round	67
8.2	Opponent modeling vs bots for different representation of state	69
8.3	Opponent modeling vs advanced bots for different representation of state	71
8.4	Paired Student's t-test	71
8.5	Win rate of a basic Opponent Modeling agent (s s-36) vs bots	72
9.1	Opponent modeling vs basic bots for different type of exploration	76
9.2	Dual between intelligent agents (see graphs Appendices 14.1)	78
9.3	Agents with dynamic Boltzmann temperature vs bots	82
9.4	Dual between agents with dynamic Boltzmann temperature and intelligent agents	83
10.1	Comparison of B-T0.5 and B-T0.5-D $\alpha\gamma$	90
10.2	Comparison of B-T0.5 and B-T0.5-D $\alpha\gamma$ both previously trained	92
11.1	Battle of agents with different maximum values for the smoothing against smart agents	94
12.1	Final model performances against bots with static strategies without any train (graphs section 14.5)	98
12.2	Final model performances against smart agents with dynamic strategies without any train	99
12.3	Final model performances against bots with static strategies with train	101
12.4	Final model performances against smart agents with dynamic strategies with train	102
14.1	All possible game situations.	115
14.2	Definition of each positional/optional argument of the simulator	116
14.3	Action of each verbosity level on the output	117

Introduction

On the first observation, the famous game rock-paper-scissors looks like a perfectly random game. Nevertheless, the New York Times created in 2012 on its website an artificial intelligence and a simulator for this game. By playing some games against it, it looks like the artificial enemy learns how to win. The Times corroborates this observation : "A truly random game of rock-paper-scissors would result in a statistical tie with each player winning, tying and losing one-third of the time. However, people are not truly random and thus can be studied and analyzed. While this computer won't win all rounds, over time it can exploit a person's tendencies and patterns to gain an advantage over its opponent."

This resumes exactly the purpose of this thesis. Based on a similar game with some more complexity, the objective is to create an artificial player who can understand the tendencies of its opponent and thus learns how to win. The additional constraint is that it must be able to run on a smartphone. The fundamental question is to know if machine learning techniques are applicable to this kind of device and if it is also able to solve the game chosen in the thesis.

For this purpose, the thesis is composed of 2 parts divided into chapters. The first one describes the different elements of the title of this thesis and the chosen game according to different frameworks. The second one is all about the experiments made to accomplish the objectives.

Part I

Machine learning, smartphones and games

The title of this part really describes as plainly as possible its whole content and follow a simple logic. Our main subject is of course Machine Learning but more a particular application of it. This induces that our first keywords are "Machine Learning" but in games. Chapter 1 will introduce it by explaining its evolution through time and various games.

Following that, now that this concept is acquired, it must be recalled that Machine Learning must be used on a smartphone. So it will be interesting to observe the capabilities of such devices and if indeed it can be applied on it or to which extends. This is the purpose of the Chapter 2.

Finally, our thesis would be useless without a game to work on. To embrace our self into this field, chapter 3 will present a framework to talk more formally about this last keyword of the title. This chapter should be seen as an introduction to the formalism and theory of games.

To conclude, the last chapter of this first part will be finally dedicated to apply all the theoretical concepts approached during the first chapters to a concrete game. This latter will be analyzed as a machine learning problem and thanks to the game framework. It will also describe its implementation on smartphones.

As for now let's jump into chapter one.

Chapter 1

Artificial Intelligence and Games

In this chapter, an overview will be given on the history of artificial intelligence (AI) through several known games. This chapter goal is to bring the insight on how artificial intelligence was thought through the last decades and how researchers achieved current models.

The first section will describe how mathematician first came with the idea that machines could learn to beat humans. Once this notion was accepted, engineers and researchers began to develop rudimentary model to try to prove this theory. This phase of history will be part of the second section. The third and fourth section will talk about the first program that beat human and the techniques imagined to achieve this goal. The fifth section is about machine learning, the domain providing these techniques and the subject of this thesis. The last section will raise the topic of the current state of game AI.

The history sum up was largely inspired from the work of Kurenkov Andrey in [11].

1.1 AI vs human

Since the inception of the modern era, people were wondering if one day, computer could outmatch the human intelligence. Since measuring the human intellect is difficult and not really quantifiable, many thought they could answer the question by first making computer programs good at very specific task and challenge this so said human intelligence. One of the main application were obviously strategic games. Since **1949**, Claude Shannon express his belief that computers could play chess [11].

“The chess machine is an ideal one to start with, since: (1) the problem is sharply defined both in allowed operations (the moves) and in the ultimate goal (checkmate); (2) it is neither so simple as to be trivial nor too difficult for satisfactory solution; (3) chess is generally considered to require “thinking” for skillful play; a solution of this problem will force us either to admit the possibility of a mechanized thinking or to further restrict our concept of “thinking”; (4) the discrete structure of chess fits well into the digital nature of modern computers. . . . It is clear then that the problem is not that of designing a machine to play perfect chess (which is quite impractical) nor one which merely plays legal chess (which is trivial). We would like to play a skillful game, perhaps comparable to that of a good human player.” [12]

Claude Shannon

But why chess, over other games. Chess was simply one of the world most popular games and its complexity made it an interesting research domain. Furthermore it is also a perfect information game (meaning that both players know the entire state of the game at any given time), but that doesn’t make it easy for either humans or machines to master [13].

The approach Shannon suggested [12] can be related as today's **Minimax** (developed and proven by John Vonnn Neuman in **1928**, see definition 1.1.1) which is one of the huge influential for AI's improvements (This theorem is important in game theory since the **Minimax** solution of a zero-sum game is the same as the Nash equilibrium - these concepts will be introduced in Chapter 3). This approach is maybe one of the most straightforward when thinking of strategic AI. The idea is to assume that both players will consider all possibles moves until the end of the game and play the move that will bring the player the closest to a winning game state and this optimally. In other words, you should always choose a move such that, even if the opponent chooses the absolute best response to that move and to every future move of yours, at the end, you would still achieve the highest possible score.

Using a computer, this can be achieved really easily since you only need to produce a program that generate from a given state of the game all next state till the end of the game. Then you simply choose the best case scenario.

But in practice, it's impossible due to the number of states to take into account. Only a few step ahead can be considered. Therefore, Shannon expressed the idea that one could use an evaluation function combined with **Minimax**. The means of this function is to rate the quality of a move compared to the optimal solution. This would bring two strategies, either brute-force the game tree of moves or use a move generator that is not only based on the game rule but that create a more plausible subset of the solution space.

Definition 1.1.1. The *Minimax Theorem* can be stated as follow; Let $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$ be convex sets. If $f : X \times Y \rightarrow \mathbb{R}$ is a continuous function that is convex-concave, i.e.,

- $f(\cdot, y) : X \rightarrow \mathbb{R}$ is convex for fixed y , and
- $f(x, \cdot) : Y \rightarrow \mathbb{R}$ is concave for fixed x ,

then,

$$\min_{x \in X} \max_{y \in Y} f(x, y) = \max_{y \in Y} \min_{x \in X} f(x, y).$$

1.2 Early stage of AI

The first implementation was created by Shannon himself on a reduce version of the chess game. Indeed Moore's Law was very crucial and restrictive in the early stage of programming as we can observe on figure 1.1. It's only by 1955 that transistors replaced vacuum tubes in computer designs.

The real first 'complete' implementation of a chess AI was conceived by Allan Turing and named TurboChamp. In **1952**, he tried to implement it on a Ferranti Mark 1 computer, but lacking enough power, the computer was unable to run the program. To overcome this problem, Turing played a game in which he simulated the computer, taking about half an hour per move for finally loosing the game [14].

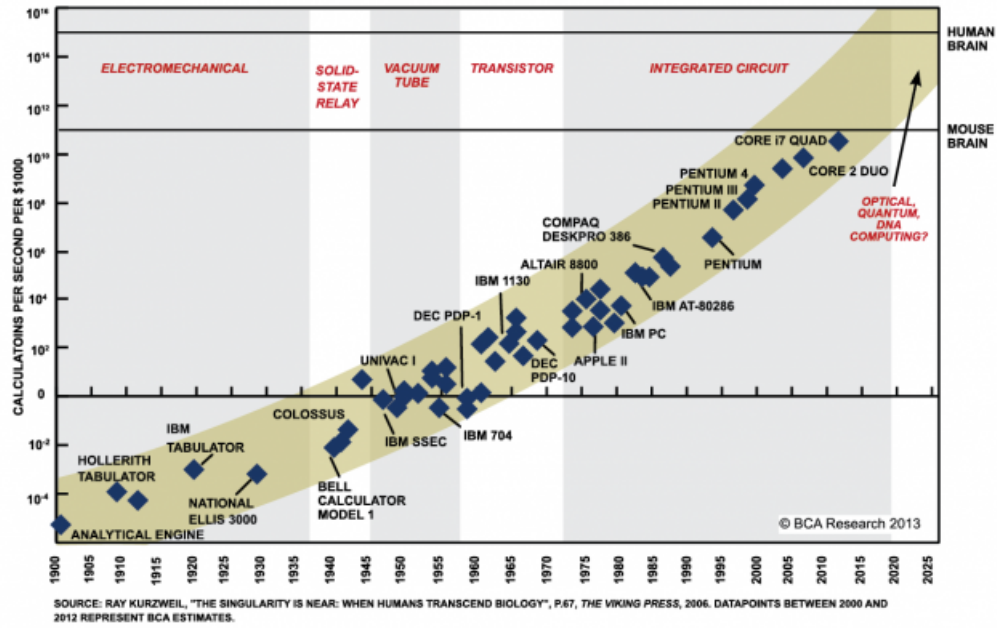


Figure 1.1: Moore's Law [1]

Checker

In **1956**, Arthur Samuel made an attempt to develop an AI that could learn to play the game of Checkers (it has 10^{20} possible board positions, which is simpler than Chess (10^{47}) or Go (10^{250})). Fundamentally, the program was based on **Minimax**, but had an additional hugely important aspect: learning. Expressed as **Rote Learning** and **learning-by-generalization**. The program remembered states of the game he already had seen and stored the final score/reward value associated with it. This ensured that at a know state, the **Minimax** algorithm could compute more depth of prediction for each state, since it already knows some solution states. The program could also adjust its reward by playing games from professional games and games played against himself. Indeed it could modify some parameters and thus modify its evaluation function. This to correct the expected value of the goodness of a move and become closer to the real reward of it [15]. At the end (mid 70's) and with the addition of **alpha-beta pruning** (see further), the program reached the level of a respectable amateur and was the first to play any board game at a sufficient level.

Chess

But who cares about Checker ? The meaningful game was always chess at this time. In **1957**, Alex Bernstein and his team set up a new AI chess player. This later also used **Minimax** but lacked any learning capabilities. It could only look at 2 moves ahead with 7 options per moves. Indeed till the 70s all similar programs would have the same restrictions. The new idea brought by this team was the addition of an heuristic instead of simply using Shannon's brute-force strategy.

Figure 1.2 , shows the evolution of the artificial intelligence throughout the 50s in term of speed, capabilities and AI's level. The last column shows another important contribution to this field of research, the addition of the **alpha-beta pruning**.

TABLE 1 Comparison of Current Chess Programs

	Turing	Kister, Stein, Ulam, Walden, Wells (Los Alamos)	Bernstein, Roberts, Arbuckle, Belsky (Bernstein)	Newell, Shaw, Simon (NSS)
Vital statistics	1951	1956	1957	1958
Date	8 X 8	6 X 6	8 X 8	8 X 8
Board	Hand simulation	MANIAC-1	IBM 704	RAND JOHNNIAC
Computer		11,000 ops./sec	42,000 ops./sec	20,000 ops./sec
Chess program	All moves	All moves	7 plausible moves	Variable
Alternatives			Sequence of move generators	Sequence of move generators
Depth of analysis	Until dead (exchanges only)	All moves	7 plausible moves	Until dead
Static evaluation	Numerical	2 moves deep	2 moves deep	Each goal generates moves
	Many factors	Numerical	Numerical	Nonnumerical
		Material, mobility	Material, mobility	Vector of values
			Area control	Acceptance by goals
			King defense	
Integration of values	Minimax	Minimax (modified)	Minimax	Minimax
Final choice	Material dominates	Best value	Best value	1. First acceptable
	Otherwise, best value			2. Double function
Programming				
Language		Machine code	Machine code	IPL-IV, interpretive
Data scheme		Single board	Single board	Single board
		No records	Centralized tables	Decentralized
			Recompute	List structure
Time	Minutes	12 min/move	8 min/move	Recompute
Space		603 words	7000 words	1-10 hr/move (est.)
Results				Now 6000 words, est. 16,000
Experience	1 game	3 games (no longer exists)	2 games	0 games
Description	Loses to weak player	Beats weak player	Passable amateur	Some hand simulation
	Aimless	Equivalent to human with	Blind spots	Good in spots (opening)
	Subtleties of evaluation lost	20 games experience	Positional	No aggressive goals yet

Figure 1.2: State of the Chess AI at the end of the 50s. [2]

Since at this time the available memory was very limited, the NSS team (Newell, Shaw, and Simon) implemented what is known as the **alpha-beta pruning** algorithm in **1958**. This latter seeks to decrease the number of nodes that are evaluated by the **minimax** algorithm in its search tree. In practice, instead of computing the whole path till the end of the game, the team developed a scoring function based on the position of the board at any given time. [16] The goal of the function was to measure the chance of winning of each player at any position on the board and therefore avoid simulating moves that are clearly bad. This to save computation resources and of course search for better moves. This algorithm became the new leader in AI and dethroned **minimax** alone. As a reminder, all these program were written in binary language. The NSS group was one of the first to use a symbolic language to program its more complex program. It was also the first to ever win a game against a human player (a secretary that learned the rules of chess one hour before the game).

1.3 When the human cannot beat the machine anymore

It's not until three more decades that the computer became the supreme master and, as before, it started with Checker. The first winning program was called CHINOOK. It was developed by a team at the University of Alberta led by Jonathan Schaeffer in **1989**. It was constituted with a database of opening moves, an alpha-beta pruning search tree with an evaluation function based on a linear combination of features, and an end-game database for all positions with fewer than eight pieces [17]. By **1994** the best Checkers player could only reach a draw against him.

Around that time, chess AI also reached another level with "Deep Thought". Developed by a team led by Feng-hsiung Hsu, it includes the same type of improvements as his predecessors but added two main components : custom hardware and moves extension. Deep Thought was as his predecessors mainly build on the idea that brute force computation are the core of the process with some heuristics to choose which moves are worth computing and playing. Therefore, the addition of specific hardware (dedicated circuit board to perform simulation and evaluation) successfully helped the program to be the fastest. The program also included an heuristic that would search after the default depth for promising moves. Another functionality also allowed the program to update its evaluation function based on games between master players. In 1988,

Deep Thought became the best computer Chess champion of the world and beat a chess grand master Bent Larsen. Some years later, an evolution of this program called Deep Blue and built on a custom super computer beat Garry Kasparov the world chess champion in 1997.

1.4 Let the machine think

The performance of the different AI improved over time but it was mainly due to the increase in computer power and some ingenious heuristics/algorithm. The programs still had no real learning and by default their intelligence relied mostly on their creator's evaluation function.

Once the programmers interested themselves in the Backgammon game, it became rapidly clear that a traditional tree pruning technique would be insufficient. Indeed Backgammon possesses a huge branching factor, this prevents to search for many moves ahead and design a good evaluation function.

In **1989**, a duo of researchers, Gerald Tesauro and Terrence Sejnowski, trained a neural network (Neurogammon) to measure the quality of a move depending on the game position [18]. This new type of evaluation function, removed the need to bring human intuition when developing an AI (at least partially at the beginning) since the idea of Neural Network is, "Given a specific task to solve, and a class of functions $\mathbb{F}$, learning means using a set of observations to find $f^* \in \mathbb{F}$ which solves the task in some optimal sense" [19].

As for now, the programs were primarily based on supervised learning, meaning that the learning is done by approximating a function through input and output. Another trend arrived after named reinforcement learning. This method generally learns from past experience by receiving rewards after trying actions in different states. Indeed every transition between states gives a reward and finding a good action allows to reach the highest reward in the long term which is desirable. This thesis will widely make use of this domain later.

A concrete application was done with TD-Gammon in **1992**. It learned just by playing games of backgammon against prior versions of itself. These observations were used to update a neural network to produce a probability of winning for any given state. This was different from Neurogammon, that used databases of moves compiled and human scored. With only a two depth tree search it reached the level of Neurogammon.

1.5 Machine Learning

Some of the applications expressed in the previous sections can be linked to a more general concept, machine learning (ML). From **rote learning** passing through **minimax alpha-beta** to reinforcement learning, ML is used to learn the evaluation function, discover patterns, model the opponent, ... The primary idea of this concept like expressed in Section 1.4, is learning. More precisely, learning for a machine [20] :

We say that a machine learns with respect to a particular task T , performance metric P , and type of experience E , if the system reliably improves its performance P at task T , following experience E .

Tom M. Mitchell

So, a program learns if, according to a performance measure, it improves with the help of the experience to do a task.

In other words, a learning algorithm is the estimation of a mathematical function. The estimation

is done with the experience. The purpose of this algorithm is to find the best solution (according to a performance metric) for the task to solve. To fully understand this notion lets clarify the notions of "Task" (T), "Experience" (E) and "Performance measure/metrics" (P) :

- A task is defined by Collins [21] as "any piece of work". A task is therefore the purpose of the algorithm.
- An experience is described by Collins [21] as "a direct personal participation or observation; actual knowledge or contact".
- For the "performance metrics", it's first interesting to look at the definition of a "metric". According to the lecture [22] : "Metrics are criteria to compare the performances of a system.". They are often related to the speed, the reliability, the precision, ... Consequently, a "performance metrics" is the value used to compare and to assess the performance of a system/agent.

Machine learning is nowadays widely used in various domains from cancer detection to film classification. And there is few chances that its days are numbered when talking about games ML. The conclusion of this chapter shall indeed show some more achievement reached by AI mainly through machine learning.

1.6 Current milestones of AI

Since Shannon, computer have been known to have down many games, using various techniques going from simple brute-force to self-learning algorithms.

The last milestones set by AI was not exclusively achieved on board games. IBM's Watson beat two champions at Jeopardy in 2011 and Google's AI achieved to play dozens of Atari games just from the raw pixel inputs (2014). Lately Google alphaGo even managed to win at GO against one of the best world player.

From all these historical fact it appears quite clearly that AI has reached a sufficient level in gaming.

In this thesis, we will focus on applying ML to games, on hardware much less efficient than those previously cited, smartphones. The next chapter will emphasis the use of smartphones for machine learning and their probable limitation. Furthermore we will only discuss a specific part of AI which is Reinforcement Learning (part of ML) on an unique game described at Chapter 4.1.

Chapter 2

Smartphones era and machine learning

Nowadays, smartphones become more and more popular. Actually, following a report from the "Global Attitude Survey" published in spring 2015 (see Figure 2.1), globally, 43 % of the population own a smartphone. In the top 11 advanced economies (U.S. and Canada, major Western European nations, developed Pacific nations and Israel) the mean of ownership even reach 68% [3].

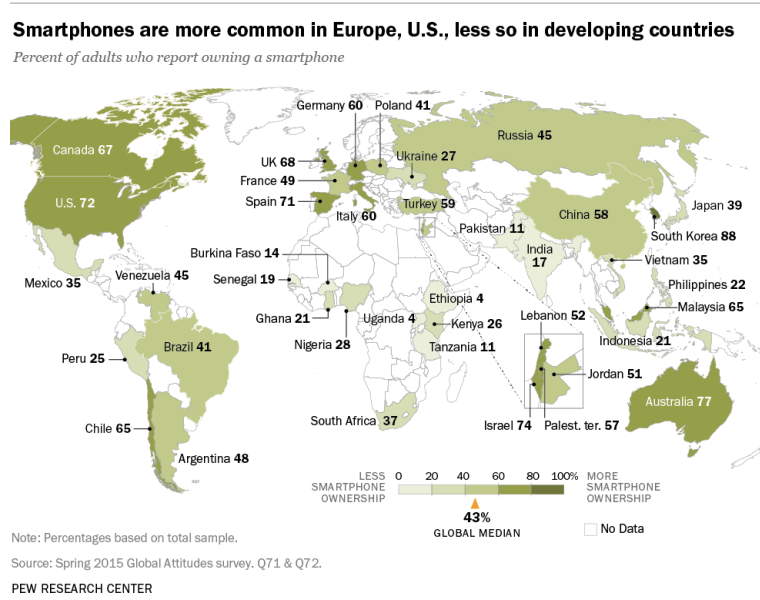


Figure 2.1: Smartphones ownership across the world. [3]

People seems to use them for everything and anywhere now; In the doctor's office. While taking a walk. In meetings. Even when making breakfast. It became something natural, handy and maybe a bit compulsive, but here is not the question. With smartphones we kind of have access to a whole universe, within the reach of an hand, but then one can reasonably ask, can we do anything with a smartphone ?

To answer that question, we'd like to recall a famous event that append in 1969, the moon landing. In this era of our history, Humanity managed to bring a manned expedition on another "planet", with a computer whose power is light years behind of what we have now. Indeed Apollo's guidance system was running at 2 MHz with 4 Kb of RAM. Compared to today's av-

era standard, say an iPhone 6, this processor is 32 600 times faster than the best Appolo era computer and can perform instructions 120 000 000 times faster [23]. Would it be wrong to say that nowadays, a smartphone could control 10K spacecrafts all at the same times ? Actually it's kind of a big leap. Indeed, Appolo's computers were designed specifically for some task and therefore totally optimized for these kind of job but considering only raw computational power, why not.

Thus coming back to the title of this thesis, it is legitimate to ask ourself if a smartphone a.k.a a not so much powerful device would be able to perform complicated task such as machine learning ? Is it only a question of optimisation ? Does a smartphones posses the sufficient computational power/hardware ? What are the motivation and theoritical limits of such an idea ? These are the first subject that will be addressed in this chapter. Thereafter, some experiments retrieved from various papers will be analyzed followed by real world industrial applications and their limits.

2.1 On device machine learning

Bringing machine learning into the device breaks the current model of the smartphones. Indeed, currently "code offload" is the most popular technique used to reduce power consumption, break through the hardware constraints (CPU, memory) of the smartphone and achieve high speed for the task execution of smartphone.

It divides executing tasks into two categories. For small computation requiring less power consuming task, it will execute locally. For large computation requiring heavy power consuming task, it will take advantage of delegating code at powerful remote server to increase execution speed and save battery power [24].

But since this model is widely used nowadays and somehow efficient ; why change it ? Machine learning on devices has many benefits. First of all, a subject that bothers many people, privacy and security. These are by this means better protected. Indeed sensitives or personal datas doesn't need to be uploaded to the cloud to be processed and on the other hand users can still benefit from the experience that the developer wants to provide and not have to risk privacy [25]. A second interest resides on the increase in responsiveness and resiliency since applications don't suffer from the server latency (for small request for example). It also decreases the need of a connection, therefore the data plan usage and network quality isn't a problem anymore for some usages [26]. Another motivation comes from the developers themselves. As the data science website KDnuggets [4] suggests from an intern poll, 80% of its developer use a laptop for their project as one of their machines, 47% of the jobs uses less than 4Gb of RAM and less than 2 cores in 40% of the cases(see Figure 2.2). This shows that many ML jobs doesn't require many hardware anymore and all these ciphers are now in the range of smartphones capabilities.

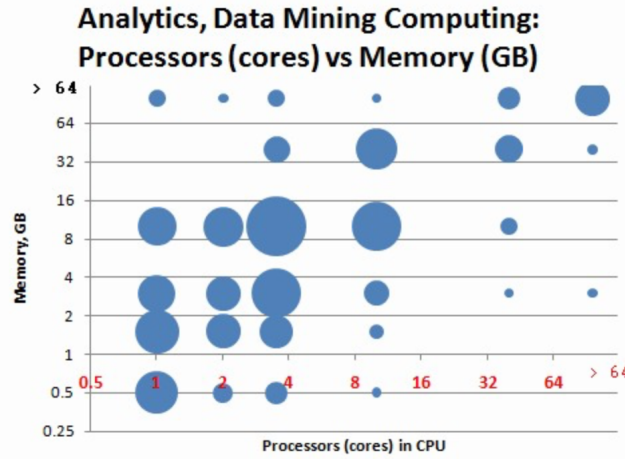


Figure 2.2: Data Mining Computing Resource Distribution: Processors (cores) vs Memory (GB). Circle size corresponds to the number of responses. [4]

Nevertheless, applying server class machine learning to a mobile device isn't without obstacle. Indeed, the implementation has not only to consider constraints in computation speed, memory and power, but also restrictions derived from thermal limits due to the lack of cooling [26]. There exists many other points like scheduling, the OS requirements, ... To put these limits into perspective, the next section will compare smartphone's hardware to more concrete examples. The OS limits or specificities will be mentioned later on.

2.2 Smartphone's hardware

Two of the most common comparison point between computers are processor speed and hard memory speed. To compare CPU speed or more precisely the brute computational processor's power, one of the easiest way is to use either the FLOPS (floating point operations per second) or MIPS (millions of instructions per second) unit. Since we are more in a mathematical/simulation perspective, it's more interesting to use the first one. The most intuitive way to define FLOPS can be given as follow ;

$$FLOPS = \text{Number of Cores} \times \text{Average Frequency (cycles/s)} \times \text{Operations per Cycle}$$

Memory is easier to compare since it only require to check the read or writing speed in mb/s.

2.2.1 Computational performances

The following charts will compare the computation power of top smartphones sold by millions of unit in the last years. These smartphones will be a good means to show the actual level of smartphones compared to other know computer like architectures. Indeed we choose for "high priced" phones since a correct computer for computation are in these range of price. We will confront their respective power to video game consoles which can be seen as "average/high" powerful computer and to supercomputers. Obviously since smartphones are less powerful, their competitors won't come from the same "time" range. Figures 2.3 , 2.4 and 2.5, shows respectively the power for smartphones, consoles and supercomputers based on [27] numbers.

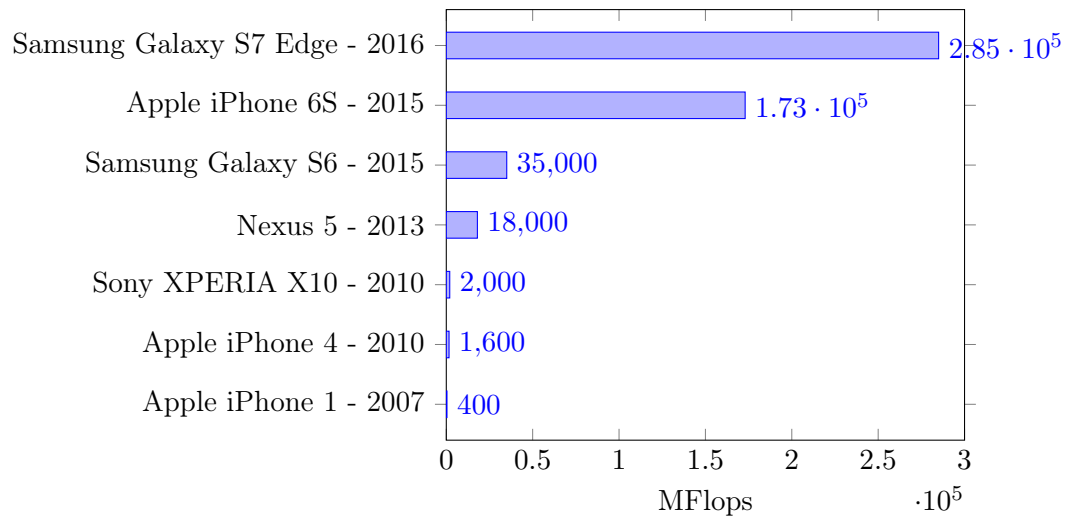


Figure 2.3: Bar chart show MFLOPS evolution for Smartphones

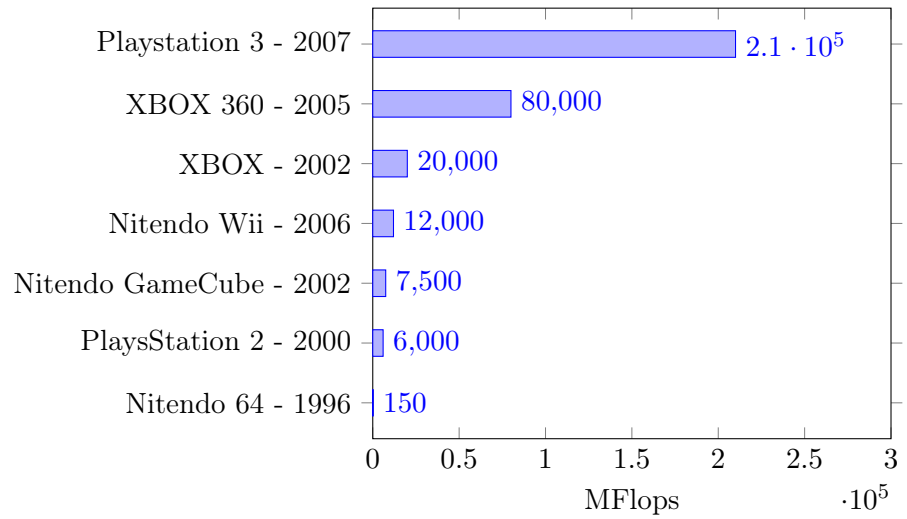


Figure 2.4: Bar chart show MFLOPS evolution for video game consoles

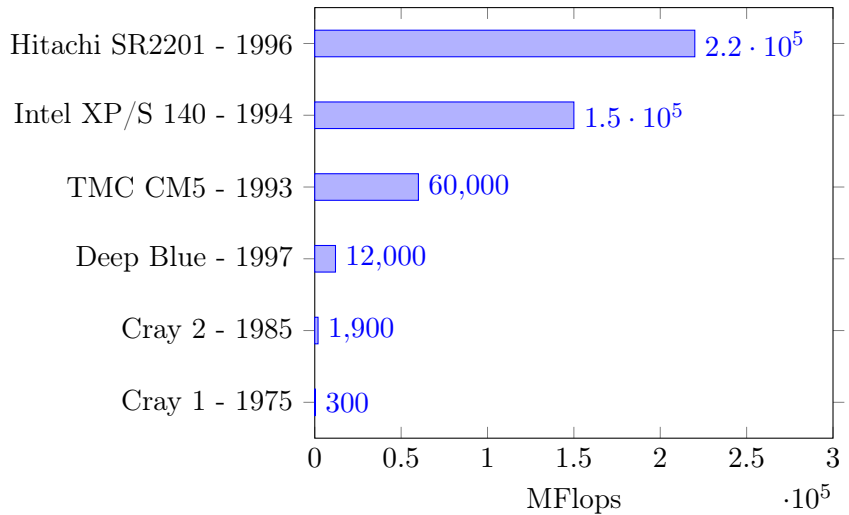


Figure 2.5: Bar chart show MFLOPS evolution for Super Computers

Today's figures are off charts to be compared to smartphones. For example the most powerful console reach 1.8 TFlops and the best super computer reach 40 PFlops. Its obviously non comparable and non significant.

However by analyzing these charts, it becomes quite natural to realize that smartphones actually have a good computational power nowadays. As for many technological sectors, the increase in power is astonishing. Indeed current smartphones can be compared to computer that are less than ten years old or even to supercomputers from 20 years ago (They are also potentially thousand of times lighter in weight). The progress in power is also faster compared to the other "older" sectors.

2.2.2 Storage performances

We previously compared raw computational power and could note that smartphone indeed posses some capabilities in this specific domain. The next step is to assess the storage performances of smartphones.

Smartphones uses two kind of internal storage, either only internal non-removable NAND flash or this latter combined with external SD cards. We will only interest ourself on the first kind (since external storage could be anything the user chooses to be).

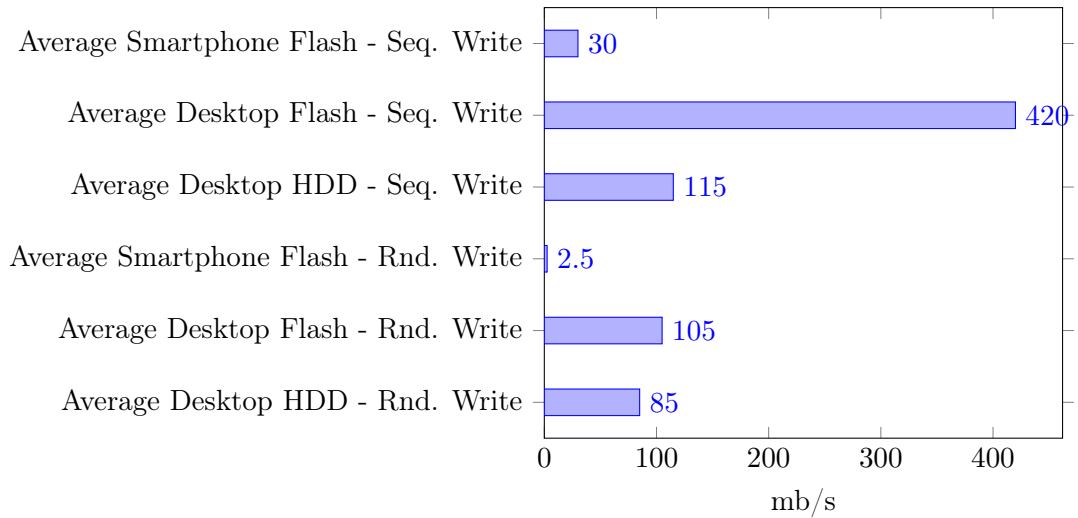


Figure 2.6: Bar chart showing random (Rnd.) and Sequential (Seq.) write speed in mb/s [5] [6] [7]

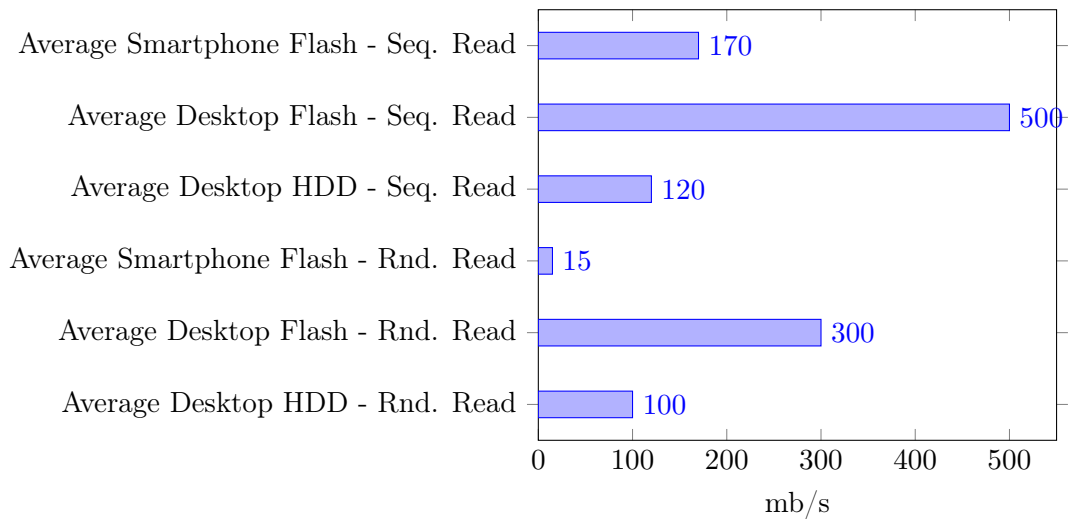


Figure 2.7: Bar chart showing random (Rnd.) and Sequential (Seq.) read speed in mb/s [5] [6] [7]

Since smartphone storage is a flash type, similar performances could be expected. Nevertheless, by comparing the writing and reading throughput, it appears that it is even worse than a standard rotating HDD. Indeed the comparison between sequential and random write speed (see Figure 2.6) with desktop SSD hardware is quite clear. There is a 35 to 40 factor for random writes and a 4 to 10 factor for sequential write. For applications that could require a lot of I/O, smartphones storage is not optimized at all.

This is even true for reads (see Figure 2.7). Smartphones are 3 times slower in sequential reads and 20 times slower in random reads compared to SSD. It only manage to beat classical HDD in sequential reads. This seems to be the only part for which smartphones are good at. This could be explained by the utilization of these devices. Indeed, for a regular use, they should only load applications or assets straight into memory for direct use.

From these graphs, the conclusion is that the internal smartphone storage is a bottleneck in general since they have slow I/O. Nevertheless their computational power seems to be good enough.

Now that these information have been highlighted, the next section will put Android and iOS smartphones to the task and assess if the conclusion depicted before has an high impact on some well known ML algorithms.

2.3 Smartphones in action

To evaluate if applying machine learning to mobile devices is a valid option, the following section will express the experiments given by a paper ¹ from the University of Berkeley [10] and another from the Georgia Institute of Technology (GATECH) [8]. The first one evaluate the following supervised learning models, support vector machines, random forests, and collaborative filtering on two android devices compared to a classical cloud request. And the second one confront an iPad Mini and a Macbook Pro on two common graph algorithms, weakly connected components and PageRank.

2.3.1 Berkeley's experiment

This experiment uses two devices. Firstly, a Nexus 6 phone running Android 5.1.1 with a 2.7GHz Krait 450 processor and 3GB RAM and, secondly, a Galaxy Tab 4 tablet running Android 4.4.2 with a 1.2 GHz Snapdragon 400 processor and 1.5GB RAM. The cloud simulation uses an Amazon EC2 instance which can be compared to a Google request, and takes approximately 200ms for each model.

The paper covered 3 machine learning techniques. The SVM part uses the MNIST digits dataset with LibSVM for its implementation. For Random Forest they used scikit-learn's models with the same dataset. Finally for the collaborative filtering part, MLLib's implementation (a.k.a alternating least squares) was used.

	Galaxy Tab 4	Nexus 6
SVM Prediction (ms)	1107	341.06
SVM Load time (s)	281	53

Table 2.1: SVM analysis

	Galaxy Tab 4	Nexus 6
10 Trees	80	27
50 Trees	110	33
100 Trees	158	41
300 Trees	408	73
500 Trees	770	105

(a) Load time (s)

	Galaxy Tab 4	Nexus 6
10 Trees	0.17	0.04
50 Trees	0.52	0.21
100 Trees	1.11	0.46
300 Trees	4.19	1.72
500 Trees	8.34	3.16

(b) Average prediction time (ms)

Table 2.2: Random forest analysis [10]

The result for each method are listed on Tables 2.1, 2.2 and 2.3. What is interesting to point out is that for any computation (only) of a prediction different than a Random Forest or

¹only the experiment values are used

	Galaxy Tab 4	Nexus 6	size
Rank 10	243	77	55.2
Rank 50	681	221	278.6
Rank 100	1305	404	562.2
Rank 200	7513	1040	1157.6

(a) Load time (s) and model size (MB)

	Galaxy Tab 4	Nexus 6
Rank 10	232.28	49.69
Rank 50	363.05	65.63
Rank 100	393.05	83.25
Rank 200	512.47	158.18

(b) Average prediction time (ms)

Table 2.3: Collaborative filtering analysis [10]

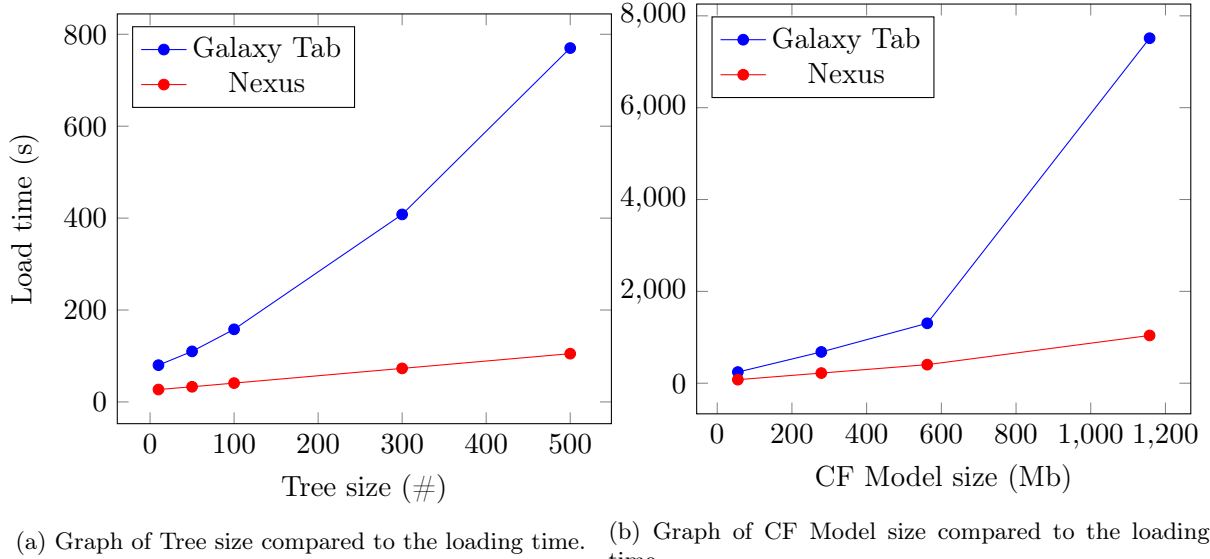


Figure 2.8: Load time comparison for RF and CF for each device.

a small rank for Collaborative filtering, a direct request to a server with loaded resources takes less time. But all of these computations still take less than half a second to be performed with a smartphone. It is still a very acceptable time. Nevertheless the loading time are execrable from 30 s to 18 min for some models, although this depends completely on the complexity of the model. The authors still points out that their loading format wasn't really fully optimized (eg. text files versus a binary format).

Whoever the loading part of any of these problems always outrank the computation time. This is mostly due to the Virtual Machine's heap size for the application processes that are limited. Applications start up with 8 Mb, and the maximum allocation is limited to 174 Mb (for 1Gb RAM) or 348 Mb (for > 2Gb RAM) based on Google specification for Android 5, but these values vary for each constructor. As opposed for example to a computer, the standard initial heap size of a JAVA VM is 256Mb up to 2Gb. This means that it is likely that with larger models, there is a lot of paging occurring, causing the extremely long load times. Another consequence is that the garbage collector is being run more often than normal which also diminish latency.

The figures on 2.8 show the loading time of each model compared to the input size of this latter. For the first figure it depends on the number of trees and on the second on the size of the matrices to load.

The two curves for the Nexus phone are strictly linear for the two situations, whereas it is

between linear and exponential for the first case with the Galaxy Tab and only linear for size till 600 Mb in the second case.

This difference can be explained by the memory difference between the two devices. In the second figure (2.8b) the three first size cause a linear time loading while the fourth isn't. It can be simply explained since the Galaxy has only 1Gb RAM which is less than the 4th model. This implies the longer loading time due to a higher paging.

On the first graph (2.8a) the blue curve tends to be more exponential. Indeed in that case we have to load up one after another several trees instead of an unique model. We can also observe that we don't have the huge explosion in time as in the second figure, this implies that for this scenario, the limitation in speed founds its origin on the disk (random) access time and not on the RAM size.

These two graphs show the weaknesses of smartphones; slow reading speed, even slower reading speed in non sequential read and nor slower writes when the device has to perform swap/paging.

When we compare the two smartphones, the Nexus seems to handle the jobs much better even if it is still globally slow. These results also show that computing on a smartphone isn't a bad idea but only for application that doesn't require the loading of lots of data simultaneously and are more based on processing time. Or maybe very optimized algorithm that limits the I/O requirements and allow to the remain in cache as long as possible.

2.3.2 GATECH's experiment

The primary experiment device is an iPad mini with Retina display, with Apple A7 chip with 64-bit architecture, 1 GB RAM and 32 GB disk space, running iOS 7.1. The laptop computer used is a 13.3" Macbook Pro with Retina display, with 2.8 GHz dual-core Intel Core i7 processor, 16 GB RAM and 512 GB SSD, running OS X 10.9.2.

The experiment consists on applying PageRank and connected components algorithm to four different sized graphs.

The data used for the experiments are stored under a binary edge list files for faster computation. Indeed, a graph can be represented as an edge list *text* file, with each row in the file is an edge, identified by its source node ID and target node ID, both stored as integers. Each integer can be transformed into a 4-byte integers. Working with binary files is in general faster than working with the raw text files, which requires parsing and converting the text.

This format has the advantage that it can be directly mapped into the device memory with the standard paging mechanism shipped with every OS.

Two of the four graphs have binary edge list files that are smaller than the iPad's physical memory (1 GB): a Pokec network [28] with 31 million edges (245 MB) and a LiveJournal network [29] with 69 million edges (552 MB). The other two graphs have binary edge list files that are close to or larger than the iPad's main memory size: an Orkut network [30] with 117 million edges (938 MB) and a snapshot of Google+ social structure [31] with 272 million edges (2.2 GB) [8].

Figure 2.9a and 2.9b shows the run time on both devices. For the two first graphs, the computation of both algorithms on the computer is around 3 times faster than that on the iPad mini.

The third graph is a little slower, it runs around 4.4 times faster on the Macbook. This is due to the limited memory of the iPad, only around 77% of the graph file was mapped into the physical memory at any time [8].

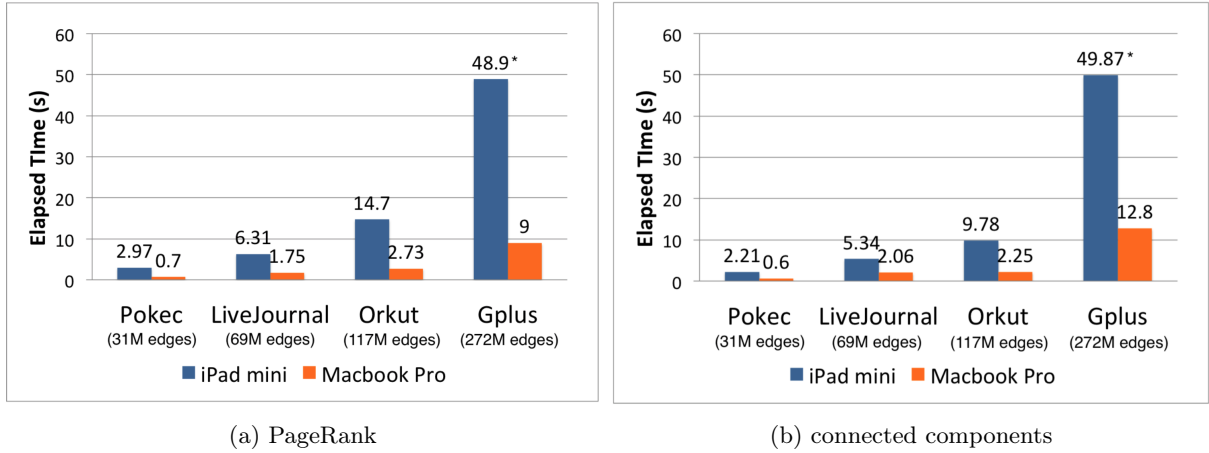


Figure 2.9: Comparing the runtimes of the connected components and PageRank algorithms, on the iPad mini and the Macbook Pro. [8]

The last graph is a little more tricky. Indeed, it can't be run directly on the mobile device, the memory address space of virtual memory for each process on the iPad being limited to 2 GB. This file being greater, it had to be partitioned through software to fit into memory and swapped when needed. Therefore, the result can't be compared. Nevertheless, it gives an idea how some specific tasks have to be specifically optimized to run on mobiles.

Before analyzing these results, it's important to compare the memory management of iOS vs Android. Indeed, as opposed to Android, each app in iOS already has the largest heap the OS can provide. iOS attempts to provide each application with as much of the device's memory as the OS can spare. However each application is limited to the device's physical memory. There is therefore no hard coded limitation of the heap size. Also writable data is never removed from memory by the operating system. Instead, if the amount of free memory drops below a certain threshold, the system asks the running applications to free up memory voluntarily to make room for new data. Applications that fail to free up enough memory are terminated [32]. Also from the I/O hardware point of view, Apple devices are very good for anything sequential (till 3 times better than the competition) and are in the average for random read/write.

For information, the experiment is 15 times slower on Android but on a 1 year older hardware with 2Gb ram. This conclude the little comparison with the previous experiment.

The Figures 2.10 now shows that the algorithm run pretty much in $\mathcal{O}(edge)$ on the Macbook Pro. On the other hand since the computation is equivalent on Mac and iOS (same language, ...), we can see that the 3rd graph takes in the two cases more time on the iPad. The difference in time could be explained by the fact that this graph is nearly the size of the RAM which is 1Gb. As a reminder the 4th graph isn't really representative since it isn't loaded the same way in memory on both devices.

On iPad the limitation still resides on the disk access time but it is more limited. The more restrictive is the total quantity of available RAM and the total VM memory available. So once again the memory seems to be the bottleneck of the device, the processor speed here should only influence the slope of the graph due to the linearity of the problem.

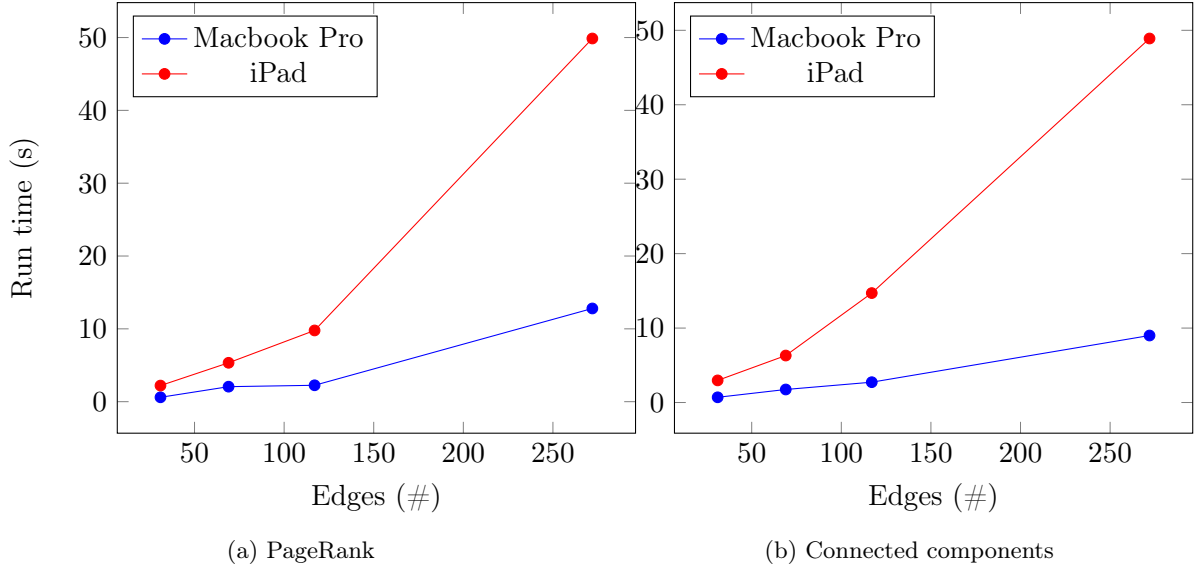


Figure 2.10: Run time comparison for PageRank and Connected Components for each device.

2.4 Application on real world problem

In the previous section we saw some application of machine learning applied to real devices and could conclude that the memory was the real drawback of smartphones. Of courses the limited processor power reduces the computation speed but it isn't such a disadvantage since the computation time is in many cases very reduced compared to the loading time.

To work around these limitations this section will present applications from Google and Qualcomm that brought cloud software to a smartphone.

Google

Google wanted to transpose Google Translate (server) deep learning onto a phone, meaning to do real-time visual translation from an image on the fly [33]. To do so, they had to crunch down their neural network. They needed to develop a very small neural net, and put an upper bound on the density of information it handles. The challenge is to create the most effective training set with the minimum of datas. To achieve real-time, they also heavily optimized and hand-tuned the math operations. Which means using the mobile processor's SIMD instructions and tuning things like matrix multiplications to fit processing into all levels of cache memory. In the end they were able to achieve good result while still being about as fast as the server version.

Qualcomm

Qualcomm created a custom software developer kit (Qualcomm Zeroth Machine Intelligence Engine) to perform some cloud machine learning (image/text recognition, natural language processing, scene detection and classification, ...) on the device through a Snapdragon 820 system-on-a-chip [26].

To develop this software, the researcher team had to optimize and compress networks to $1/10^{th}$ of their size to reduce the memory and computation requirements on the device. They even used 16-bit values instead of 32-bit to reduce the size even more. They also used custom

BLAS api and parallelism mechanism. All of this to achieve limited impact in the measured performance.

While the SDK is designed to learn on device, it still can work with cloud network. This functionality is designed to port down to the devices training and refined data sets, since the phones may not have the power to fully crunch the data to create the networks.

These two real life applications shows compared to the previous experiments which were kinda discouraging in term of "run time" (from Section 2.3) that if applications are heavily optimized, even with the smartphones limitations, good results can be achieved and compared to server like solutions.

2.5 Conclusion

This chapter regrouped many different points which for most of them indicates that smartphones have a good potential to be a receptacle for machine learning. Nevertheless many limitations still exist and specific or additional work as to be performed to achieve similar/equivalent result than on other platform. As for other application like *games* before, the devices were not fully designed for these kind of job, but with time and the increase in interest - devices got specific "gaming" hardware and api. Therefore if machine learning continues to have an high values to these sectors, phones are likely to be provided with specific chips and api for these tasks. Now that the smartphone constraint has been examined, the last keyword of the title will be the subject of the next chapter.

Chapter 3

Game theory

This chapter covers some of the general aspect of game theory. Indeed to fully understand games it is important to acquire the notions associated with them. Although this chapter only explains and defines some of the basic concepts of game theory and could totally find its place in Appendices, we thought it important to leave it in the core text since it covers notions less known in Engineering.

Thus this chapter will first define game theory and how games should be represented depending on their characteristics, followed by the explanation on some typical game type. You would have understand, mostly related to the game we will present later. Finally the last section relates which are the optimal strategies for a game and why game theory while being theoretical can be applied to a human being and its usefulness.

3.1 Game theory notion

Game theory is "the study of mathematical models of conflict and cooperation between intelligent rational decision-makers." [34]. It is a mathematical framework for modeling behavior of rational agents interacting in an environment. It analyzes and prescribes what strategy should agents take in order to achieve their goals, to maximize their expected payoff but the outcomes in question might have been intended by none of the agents [35].

Today's approach of Game Theory's mathematics is mostly associated to John Von Neumann work and it's definition of the **minimax** algorithm (see definition 1.1.1) . But before hand, what is the conceptual idea of Game Theory ? We can take as example a text from Plato, "the Laches and the Symposium". In this latter, Socrate recalls an episode of the Battle of Delium that considers a soldier at the front line in a defense position. This soldier may be entitled to ask himself that if the defense is most likely to hold, then his personal contribution may not be essential. But by staying he run the risk of being killed/wounded for no reason. Now if his army is going to lose anyway, then his chance of dying/being wounded are even greater, and therefore it's quite clear that he has no point to stay. Thereby it is shown that for any soldier fleeing is the best option regardless of the outcome of the battle. This can be applied to every soldier and even more to each army. The greater the soldier's fear of defeat, the likelier they are to flee. And the greater the soldier belief that they would win, the less they would want to fight. In fine, the two armies should disband without a single shot being fired [36].

This conclusion doesn't come from the fact that the soldiers are scared of the dangers. But rather because they discovered what would make sense for them to do depends on what would make sense for the other to do, and all that can be notice by the other too. By this means without loss

of generality we can imagine two armies of brave soldier that would both flee instead of having a fair fight, although this wasn't the result that they all would have wanted. This outcome was produced by the interaction of many individually rational decision-making processes but wasn't intended by no one.

The reasoning of the soldier, and in general for any agents, derives from their situation. The aspect of the environment that is most important to the agent's achievement of their preferred outcomes is the set of expectations and possible reactions to their strategies by other agents. The distinction between acting parametrically on a passive world and acting non-parametrically on a world that tries to act in anticipation of these actions is fundamental [36]. This state the principal implication of Game Theory.

3.2 Basic definition

A game is all the ensemble of situation in which at least one agent wants to maximize his utility by anticipating the response to his actions by (an)other agent(s). Agents in games are called players. Note that, Game Theory doesn't need to be applied when all agents have optimal actions regardless of the actions of other agents (like in conditions of monopoly or perfect competition). In a game each player posses a certain panoply of strategies. A strategy is a high level plan to achieve one or more goals under conditions of uncertainty, meaning a predetermined series of action(s) to answer to whatever other strategy the other player would play.

Game can be defined in two major categories, games in which agent have perfect information or not, when they choose their strategies. Perfect information means that each time a player has to make a decision using its strategy, he has a total knowledge of what happened before hand in the game. This comes in contradiction with imperfect information games, that regroup games where components of the game have an uncertainty chance, are hidden or even simultaneous. These two classes bring an easy ascertainment. From what a player believes or fails to believe about each other's actions/strategy is a key part in the Game Theory analysis. Another important aspect to point out is that (im)perfect information is related to the order of play but is not similar to it. If we compare sequential-move and simultaneous-move games in terms of information, its easy to misinterpret and suppose that the former represent game in which strategies are taken in order and the latter where strategies are taken at the same time. Indeed it is not the order in which the strategy is taken that is important but more when the player is aware of the actions of the other player concerning his own action. E.g., we can imagine two competitor marketing campaigns that are set up with months of interval in term of decision making, but since they are unaware of one another or doesn't know if it will one day exist, it is a simultaneous-move game though the decision were taken at different time. Another attribute can be added to games which is recall. Perfect recall assumes that each player is allowed by the rules of the game to remember everything he knew at previous moves and all of his choices at those moves [37].

These notions defines games but other conclusions can also be retrieved by how a game is visualized. This is covered in the next section.

3.3 Game representation

Depicting a game can be achieved using two principal forms, the normal or the extensive one.

3.3.1 Normal form

In simultaneous game, the strategy must be determined for each player before knowing what the other player will play. Their strategy must be determined depending of the state their opponent is in before playing their next move. For a game with finite moves, meaning that for each play there is only a given amount of action that a player can play according to its strategy, the state of the game can be represented with a $\mathbb{N} \times \mathbb{M}$ matrix where one player has $\mathbb{N}$ possible moves they can play and the other has $\mathbb{M}$ playable moves. This matrix is often called payoff matrix. Indeed by looking at the intersection of the move that each player chooses, e.g. *player one* chooses action x and *player two* chooses the action y , the payoff given at each player if these play/strategies would be chosen can be found in the matrix at the corresponding intersection, meaning the row x and column y .

This simple explanation can be formalized for $\mathbb{P}$ players (definition 3.3.1) :

Definition 3.3.1. A normal (or strategic) form game is a triplet (P, S, U) with the following properties.

- $P = 1, 2, \dots, p$ is a finite set of players
- S_i is the set of pure strategies (see Definition 3.5.2) of player i ; $S = S_1 \times \dots \times S_p$
- $u_i : S \rightarrow \mathbb{R}$ is the payoff function of player i ; $u = (u_1, \dots, u_n)$.

Player $i \in N$ receives payoff $u_i(s)$ when $s \in S$ is played. The game is finite if S is finite. The structure of the game is common knowledge: all players know (P, S, U) , and know that their opponents know it, and know that their opponents know that they know, and so on. [38]

A classical example to explain this type of representation is the Prisoner's Dilemma. This latter was invented by Merrill Flood and Melvin Dresher and formalized by A. W. Tucker [39] :

"Two members of a criminal gang are arrested and imprisoned. Each prisoner is in solitary confinement with no means of communicating with the other. The prosecutors lack sufficient evidence to convict the pair on the principal charge. They hope to get both sentenced to a year in prison on a lesser charge. Simultaneously, the prosecutors offer each prisoner a bargain. Each prisoner is given the opportunity either to: betray the other by testifying that the other committed the crime, or to cooperate with the other by remaining silent. The offer is:

- If A and B each betray the other, each of them serves 2 years in prison
- If A betrays B but B remains silent, A will be set free and B will serve 3 years in prison (and vice versa)
- If A and B both remain silent, both of them will only serve 1 year in prison (on the lesser charge)"

The latter situation we'll describe a simultaneous "game", and typically is a non zero-sum game but we will come back to this point later. This game can typically be represented with a matrix, see Table 3.1.

		B	
		Silent	Confess
A	Silent	1, 1	4, 0
	Confess	0, 4	2, 2

Table 3.1: Prisoner's Dilemma Payoff Matrix

Each value in the matrix represents the number of year the specific player will pass in prison. A rapid analyze show that regardless of the action of B, it's always more interesting to confess than to remain silent (for its own interest obviously). Indeed, if B is **Silent**, if A's action is **Silent** he stays one year, but if he **Confess**, he is free. Likewise, if B **Confess** and A is **Silent** he stays four years or 2 years if he **Confess** too. This also yield for B. The optimal strategy will therefore be, to always confess. This equilibrium is called Nash equilibrium (see Definition 3.5.7). However, the payoffs for (**Silent**, **Silent**) are both less than those for (**Confess**, **Confess**). Therefore (**Silent**, **Silent**) is more Pareto-efficient (see Definition 3.5.8) than (**Confess**, **Confess**).

3.3.2 Extensive form

The normal form introduced before is generic enough to describe games with finite strategies. On the other hand, this representation is kinda noisy for large games. This is even more true for sequential games where player has knowledge of the other player's previous action when he chooses his strategy. Extensive form is therefore another representation that allows to show more/different informations.

An extensive form displays the sequencing of player's possible moves, their choices at every decision point, the (possibly imperfect) information each player has about the other player's moves when he makes a decision, and his payoffs for all possible game outcomes. Extensive form games also allow representation of incomplete information in the form of chance events encoded as "moves by nature" [40]. Games in the extensive form are most of the time visualized as game trees (see Figure 3.1b). The concept was defined as follow by Hart [41];

Definition 3.3.2. A (finite) *extensive form game* is,

- A finite set of p (rational) players
- A rooted tree, called the game tree
- Each terminal (leaf) node of the game tree has an p -tuple of payoffs, meaning there is one payoff for each player at the end of every possible play
- A partition of the non-terminal nodes of the game tree in $p+1$ subsets, one for each (rational) player, and with a special subset for a fictitious player called Chance (or Nature). Each player's subset of nodes is referred to as the "nodes of the player". (A game of complete information thus has an empty set of Chance nodes.)

- Each node of the Chance player has a probability distribution over its outgoing edges.
- Each set of nodes of a rational player is further partitioned in information sets (see Definition 3.3.3), which make certain choices indistinguishable for the player when making a move, in the sense that:
 - there is a one-to-one correspondence between outgoing edges of any two nodes of the same information set—thus the set of all outgoing edges of an information set is partitioned in equivalence classes, each class representing a possible choice for a player’s move at some point, and
 - every (directed) path in the tree from the root to a terminal node can cross each information set at most once the complete description of the game specified by the above parameters is common knowledge among the players

Definition 3.3.3. An *information set* is a set that, for a particular player, establishes all the possible moves that could have taken place in the game so far, given what that player has observed. If the game has perfect information, every information set contains only one member, namely the point actually reached at that stage of the game. Otherwise, it is the case that some players cannot be sure exactly what has taken place so far in the game and what their position is [42].

A game is then the path through the tree from the root following the edges corresponding to the different players actions and ending to a final leaf. At any Chance node, the outgoing edge is chosen following to the probability distribution previously defined. At any player node, the player must choose one of the edge depending on the information set of the node ; this derives an unique edge. In case of imperfect information, the player can’t know which edge is followed (an omniscient observer, knowing the nature and all players moves can determine the current edge).

It’s interesting to mention that the extensive form can also be used to describe simultaneous games, by using information sets. These latter, are usually represented by a dashed line uniting two nodes (see Figure 3.1a). This means that the player does not know in which node he is, which implies imperfect information, like when using the strategic form. Moreover to totally come back to the definition of normal form (Definition 3.3.1), we can also define the pure strategy (Definition 3.5.2); for a player it consists of selecting precisely one class of outgoing edges for every information set (of his).

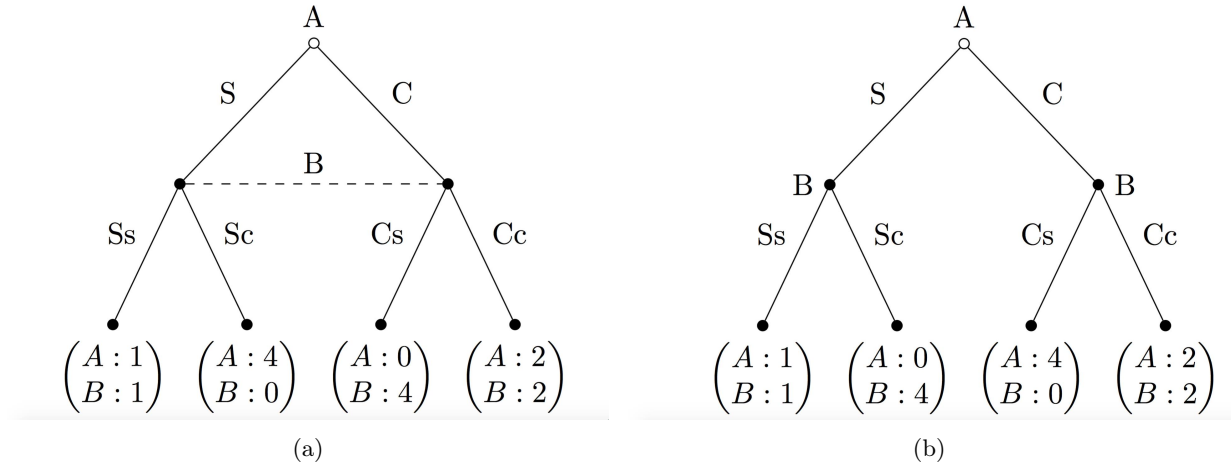


Figure 3.1: Game representation

3.4 Type of games

Now that we have defined how we can represent a game, it's also important to understand against which type of game we deal with. Indeed depending on his kind, the strategies applied can be very different.

3.4.1 Cooperative and Non Co-operative Games

Cooperative game theory focuses on how much players can appropriate given the value each coalition of player can create, while non-cooperative game theory focuses on which moves players should rationally make. [43] In a cooperative game, the participating players are allowed to communicate freely and bargain with one another to try and obtain the best payoff they can. Simple examples to illustrate these two types are a bargaining between a merchant and a buyer or a Tic-Tac-Toe game. Indeed in bargain, the merchant wants to sell at the higher price but don't want the buyer to go away and conversely the buyer wants/need the product but at the lowest price possible. Tic-Tac-Toe is a non-cooperative simply because winning is the only meaningful rule.

3.4.2 Zero- and non-zero-sum games

A zero-sum game is a game in which each player's gain or loss of utility (measure of preferences over some set of goods and services) is exactly balanced by the losses or gains of the utility of the other participant(s) [44]. Therefore the combined total of gain and loss of all players is equal to zero. A simple example is a pie division. If one person cuts a larger portion, the others will have less. This example is only a valid zero-sum game if every eater values the pie equally and want to maximize their utility.

On the other hand, a non-zero-sum (or general-sum) describe a game in which the players combined sum of gain and losses is non-zero (either less or more). In a non zero-sum game it is possible for all players to gain or suffer together. Participants of a non zero-sum game usually have a common interest at heart whilst still being in conflict over other issues [45].

A zero sum-game is called a strictly competitive game while a non-zero-sum game can be either competitive or non-competitive.

Definition 3.4.1. A *strictly competitive game* is a two-player strategic game $(\Pi_i, \Pi_{-i}, p_i, p_{-i})$ for any two joint strategies (see Definition 3.5.1) π and π' :

$$p_i(\pi) \geq p_i(\pi') \text{ iff } p_{-i}(\pi) \leq p_{-i}(\pi').$$

That is, a joint strategy that is better for one player is worse for the other player. This formalizes the intuition that the interests of both players are diametrically opposed and explains the terminology. By negating both sides of the above equivalence we get

$$p_i(\pi) < p_i(\pi') \text{ iff } p_{-i}(\pi) > p_{-i}(\pi').$$

So an alternative way of defining a strictly competitive game is by stating that this is a two-player game in which every joint strategy is a Pareto efficient (see Definition 3.5.8) outcome. [46]

A zero-sum game is an extreme form of a strictly competitive game in which whatever one player wins, the other one loses (tie is also admitted). The simplest example is the Roshambo game. The Prisoner's Dilemma, on the other hand, is a non-zero-sum game.

In 1944, Von Neumann and Morgenstern proved that any non-zero-sum game for n players is equivalent to a zero-sum game with $n + 1$ players; the $(n + 1)^{th}$ player representing the global profit or loss. [47]

3.4.3 Stochastic Games

Stochastic games were introduced by Lloyd Shapley [48]. They represent dynamic games played by one or more players. The game is played iteratively, meaning that at each state of the game, each player select an action and receive a payoff that depends on the current state of the game and all the actions played. The game transition to a new random state whose distribution depends on the previous state and the action chosen. This is repeated infinitely or until the game ends.

Definition 3.4.2. A *stochastic game* consists of a tuple $= (P, \Pi, S, A, R, s_0)$.

- P is the set of players. $n = |P|$
- Π , the set of joint-actions
- S is the set of states
- $A : S \times \Pi \times S \rightarrow [0, 1]$ is a transition function which depends on the actions of all players.
- $R : S \times \Pi \times S \rightarrow \mathbb{R}$ is a reward function representing the expected value of the next reward, which also depends on the actions of all players.
- $s_0 \in S$ is the initial state [49]

It's important to notice that stochastic games generalize both Markov decision processes (see Definition 6.1.4) and repeated games. Indeed an agent starts in an initial state followed by repeatedly choosing actions, receiving rewards, and transitioning to new states. These concepts will be used later on.

Now that the basis of games has been set, a good idea would be to understand which are the best strategies and why. This is the topic of the next section.

3.5 Equilibrium and optimality concepts

Equilibrium refers to the possible outcome of games, considering the goal of each player, strategy and the type of game considered. First of all the concept of strategies need to be defined; The concept of strategy can easily be misinterpreted as being a move. A move is the action played by a player at a given time in the game. A strategy on the other hand is a complete algorithm for playing the game, telling a player what to do for every possible situation throughout the game [40].

Definition 3.5.1. A *strategy profile* (sometimes called a strategy combination or joint strategy) is a set of strategies for all players which fully specifies all actions in a game. A strategy profile must include one and only one strategy for every player.

Definition 3.5.2. A *pure strategy* provides a complete definition of how a player will play a game. In particular, it determines the move a player will make for any situation he or she could face.

Definition 3.5.3. A *mixed strategy* is an assignment of a probability to each pure strategy. This allows for a player to randomly select a pure strategy. Since probabilities are continuous, there are infinitely many mixed strategies available to a player. (A pure strategy can also be seen as a mixed strategy where a given strategy is selected 100 percent of the time and the others are never chosen.)

Definition 3.5.4. A totally mixed strategy is a mixed strategy with full support, i.e., one in which each strategy is used with a strictly positive probability.

Definition 3.5.5. Let π_i, π'_i be strategies of player i . We say that π_i *strictly dominates* π'_i if $\forall \pi_{-i} \in S_{-i}, p(\pi_i, \pi_{-i}) > p(\pi'_i, \pi_{-i})$. Further, we say that π_i is *strictly dominant* if it strictly dominates all other strategies of player i [50].

3.5.1 Nash Equilibrium (NE)

The Nash Equilibrium is a solution state in a non-cooperative game with n players, in which each player is assumed to know the equilibrium strategies of the other players, and no player has anything to gain by changing only his own strategy [51].

If each player has chosen a strategy and no player can benefit by changing strategies while the other players keep theirs unchanged, then the current set of strategy choices and the corresponding payoffs constitutes a Nash equilibrium [52].

To study this latter, we will need some assumptions (given in [53]);

Strategic (see Definition 3.3.1) games are studied under the following basic assumptions :

- players choose their strategies simultaneously; subsequently each player receives a payoff from the resulting joint strategy,
- each player is rational, which means that his objective is to maximize his payoff,
- players have common knowledge of the game and of each others' rationality.

Definition 3.5.6. A strategy s_i of player i is called a best response to a joint strategy π_{-i} of his opponents if,

$$\forall \pi'_i \in \Pi_i, p_i(\pi_i, \pi_{-i}) \geq p_i(\pi'_i, \pi_{-i})$$

.

Definition 3.5.7. Next, a joint strategy s is called a Nash equilibrium if each π_i is a best response to π_{-i} , that is, if

$$\forall i \in \{1, \dots, n\} \forall \pi'_i \in \Pi_i, p_i(\pi_i, \pi_{-i}) \geq p_i(\pi'_i, \pi_{-i})$$

.

So a joint strategy is a Nash equilibrium if no player can achieve a higher payoff by unilaterally switching to another strategy. Intuitively, a Nash equilibrium is a situation in which each player is a posteriori satisfied with his choice.

3.5.2 Pareto optimal

Pareto efficiency is a state in which it's impossible to increase one player's utility without worsening another's. This equilibrium is a minimal notion of efficiency, but doesn't result in a fair distribution of the utility. There is no idea of equality or global fair optimum. Concepts defined below, can however state the social efficiency of a solution.

Definition 3.5.8. We call a joint strategy π a *Pareto efficient* outcome if for no joint strategy π'

$$\forall i \in \{1, \dots, n\}, p_i(\pi') \geq p_i(\pi) \text{ and } \exists i \in \{1, \dots, n\}, p_i(\pi') > p_i(\pi).$$

That is, a joint strategy is a Pareto efficient outcome if no joint strategy is both a weakly better outcome for all players and a strictly better outcome for some players. [54]

Further, given a joint strategy π we call the sum $\sum_{j=1}^n p_j(\pi)$ the *social welfare* of π . Next, we call a joint strategy π a *social optimum* if the social welfare of π is maximal. Clearly, if π is a social optimum, then π is Pareto efficient. The converse obviously does not hold. [54]

Using once again the "Prisoner's Dilemma" (see Section 3.3.1), the previous theory can be applied. All but the joint strategy that choose to both **Confess** is Pareto efficient. Indeed you can't change one's strategy to increase one's utility without decreasing someone's else. This strategy is said to be Pareto dominated by (**Silence**, **Silence**). This latter is in fact the social (Pareto) optimal strategy. The social welfare is equivalent for all the strategies that are not the optimal one. Interestingly enough the optimal strategy is not the Nash Equilibrium (**Confess**, **Confess**) has it would have been expected.

These optimality concepts are great but quite theoretical and do they still work when humans are involved ?

3.6 Game theoretic models and Human nature

There exist variety of games where the best strategy is known. For example, in the Roshambo game, playing perfectly random is theoretically the best way to play. Indeed, it is impossible to gain an advantage over a truly random opponent. However, by exploiting the weaknesses of nonrandom opponents, it is possible to gain a significant advantage. This is even more true since human players tend to be nonrandom.

In fact what is commonly accepted is the idea that humans are prone to mimicking the behavior and gestures of others, an idea called "automatic imitation". So humans are unconsciously biased by their opponent. An experiment conducted by the UCL university [55] on the Roshambo game indicated that if two players were blindfolded during the game, the win, lose and tie probabilities were exactly 1/3. Whereas if only one side is blinded, the number of games ending in draws was significantly higher than a third, suggesting that the sighted player was copying the gestures of the blindfolded one – even when it was not in their interests.

Another ascertainment was made by another research team under the work of Zhijian Wang at China's Zhejiang University [56]. They have demonstrated that people actually follow a pattern when playing Roshambo.

Players who win tend to stick with the same action while those who lose switch to the next action in a clockwise direction (where $R \rightarrow P \rightarrow S$ is clockwise). They theorize that since players are generally aware that a random strategy minimizes their chances of losing, the patterns players fall into are a kind of acquired reflex. In this case, it's following the name of the game.

These findings show that even on a game where total randomness is the key to success game theory can be applied to study human tactics. And more importantly, machine learning that infer the opponent behavior can be used and will "theoretically" be effective.

The next section will present the game we choose to analyze. This game like Roshambo seems to have a predisposition for randomness. But experiments from Section 7.2.1 will show otherwise. Indeed, randomness is something good in the game but not at each state of it. And as demonstrated before humans tend to be nonrandom, therefore applying machine learning to this game is something viable and interesting enough to be studied.

Chapter 4

The game

This master thesis obviously needs a game. The game choice was based on the reference article [57] for this thesis which was about a Rock-Paper-Scissor AI. This article explained that a truly random game of rock-paper-scissors would result in a statistical tie with each player winning, tying and losing one-third of the time like explained in section 3.6. However, since people are not truly random, a person's tendencies and patterns can be exploited by computers to take advantage.

To describe the chosen game, this chapter will depict what is a game and what makes a game something interesting for us to work on. Followed by the game description itself, how game theory analyses it and its implementation as a smartphone game.

4.1 A game ?

According to Salen & Zimmerman, the formal definition of a game [58] is:

" A system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome."

Salen & Zimmerman

Combining this statement with the definition given by Wolfgang Kramer in his book [59], a game must contain the following particularities :

Game rules which are the main part of the game and must successfully regulate the game flow and termination conditions.

Goals which are nothing else than the victory condition or the quantifiable outcome.

A game should also include a competition so that there are winners and losers or in other words an artificial conflict.

And one last but important component, chance so that the course of the game is as much as possible never the same.

These elements are the main components of a game. but to chose one, we personally have some more criteria;

1. The best player for the chosen game can't be a random player because the final agent must be smart and consequently not simply play randomly.
2. It mustn't be too difficult to understand and implement. The purpose of this thesis is the machine learning part and not the implementation of the game.

3. It must be playable on a smartphone meaning, intuitive enough to have a simple UI to be played on a mobile.

With all these conditions, we also wanted a game that was close to the original article game. Thus something simple but still interesting.

In the following sections, we will define our choice and describe it as a "game with rules" (according to [59]) and as a "Machine Learning problem" (see section 1.5).

4.2 "Cowboy game"

The "Cowboy game" (found on [60]), also called "Shotgun game" or "007", is similar to "rock-paper-scissors" (a.k.a. Roshambo). Indeed, it is basically a two players simultaneous turn based hand game but conversely to Roshambo, it's an iterative game. Therefore each action played at the previous round is reported, if it has any incidence, to the next round until the victory condition are met. Since the game is simultaneous, it also includes the concept of **chance** because the player can't guess the action chosen by his opponent.

The game has a simple set up. Initially, players have the same status. They are defined by the following characteristics :

- An activable shield with a solidity of 9
- a power of 0
- 1 health point

The **victory condition** is simple : the first player reaching no health point dies and consequently loses the game. The other player, therefore, automatically wins so the game is **competitive**. If both player dies simultaneously, which is possible, a tie is reached. At the beginning, both players have the same chance/propensity to win.

At each iteration of the game, the player can choose between 3 actions (if available), **Reload**, **Shield** or **Shoot** to modify his or its opponent characteristics. Simply speaking the ultimate goal is to **Shoot** directly toward the opponent health point. But to be able to **Shoot**, a player must **Reload** and a player can protect itself from this attack by activating his **Shield** for one turn.

A more precise description of the actions is given here after :

1. Reload : increase your power by 1. The maximum value for this attribute is 10.
2. Shoot : the power of the shoot can be 1,3,5 or 10 but the player must have at least a power of this amounts. After a shoot, the power of the player decreases by the power of the shoot. Each direct hit regardless of the power (without the opponent using the action **Shield**) is deadly.
3. Shield : used to block a possible shoot of the opponent. To be able to block a **Shoot**, the solidity of the shield must be superior or equal to the shooting power. As a result the shield is decreased by the power of the **Shoot**. If the opponent plays **Shoot** but the solidity is strictly less than the power, the bullet hits the health point of the player and he therefore dies/loses. If **Shield** is played versus another action than **Shoot**, nothing happens for this player.

Consequently, a player loses when the opponent shoots while he reloads or when his shield isn't sufficient. Indeed each player has only 1 health point and a **Shoot** has a minimum value of one. If both players shoot at the same time/iteration, the game reach a tie.

If the **game rules** are not clear enough, a summary table can be found in appendices at Table 14.1.

4.3 Cowboy game as a ML problem

As expressed in section 1.5, let's first define the three main notions and observe how they fit the main definition.

In our case, the Task is to create an agent who plays against a human and can learn how to win at the "Cowboy game". It means :

- Learns the good/bad combination of actions (for example, the action "shield" at the beginning of a game is useless because of the null initial value of the player's power)
- Adapt its strategy when it loses or wins
- Finally find an optimal strategy, if it exist.

Ultimately the role of the Task is, at each round, to output the next action the agent should play to maximize its chance to win the game against its human opponent with some Experience if any (from bots or reals humans). This Experience could be acquired either by observing games or by playing against the agent or both accrued.

But how would the quality/performance of this agent be assessed ?

A good agent should achieve a certain win rate. Against bots this win rate should be (much) higher than 50% since by definition they are predictable, whereas against humans who tends to play more intelligently, a win rate near 50% is acceptable. Win rate is computed as follow : +1 for every win, +0 for loses and tie are excluded (see equation 4.1).

$$\text{win rate} = \frac{1 \times \text{wins} + 0 \times \text{loses}}{\# \text{games} - \# \text{ties}} \quad (4.1)$$

Indeed, if we include tie as +0.5 for example it has a tendency to smooth to 50% scores which is bad for comparison. It is more interesting to separate win/lose score and tie score to have more granularity. But this has a drawback, matches with a high tie rate can't be reliable due to the reduced (win/lose) sample size used to compute the score.

It's also important to consider on how much game this rate is computed. A player should not play 1000 games such that the agent learns enough to be efficient. As by personal experience a fair amount of game to play is near 80-100 which is equivalent to 10-15 minute game time. This number reflects two aspects. Firstly, less than a quarter of an hour play time seems acceptable. Secondly and maybe more importantly, the game has a high tie rate, approximately 30-40%, and the (bad/good) luck factor must not be neglected. For these two last reasons a number of game below 50, is highly influenced by these factor and may not be representative. In the experience we will present scores at 50, 100 and 500 games to show short terms tendencies that are highly variable in general (at 50), the main score at 100 games and the global tendency at 500.

The game length is also an issue. A classical game length between humans is near 10-15 moves per game. Agents even against bots that tends to have long game length compare to other (with same win rate) should be considered less efficient. Indeed either they don't adapt fast enough or don't really know which is the "good" action to play which could mean that the agent as fallen in a local optimum (can happen against bots for example).

Another significant factor is the capability of the agent to adapt himself to various kind of player and this the most efficiently as possible. The goal of the agent is not to be the perfect hard counter of a specific agent but rather to be able to face various opponent equally. Therefore the mean victory is more important than "local" very high scores. Most of these factors which makes the performance measure can't be really rated with a score and therefore are a more a personal evaluation.

4.4 Game Theory applied to Cowboy

By using the game theory defined by Chapter 3, the "Cowboy Game" has the two common attributes of being a two-person competitive zero sum and non co-operative stochastic game. Indeed, as for Roshambo, zero sum implies that if one player wins the other loses with ties admitted and the non co-operative part comes from the same deduction. The game is also stochastic since it is an iterative game with actions influencing one's utility.

To define the game we also have that each player knows what all the players did in the previous iterations, but not what they're doing in the current iteration especially due to the simultaneity of the game. Thus, we have an imperfect-information game with perfect recall.

Now that the game is defined through his main characteristics, let's study its equilibrium and optimal strategies. To make it easier, instead of analyzing the entire game at once, we will only consider 4 typical game states which can be repeated at each iteration of the same round :

1. no player has power
2. both player don't have have power sufficient to bypass the opponent shield
3. one player has a power sufficient to bypass the opponent shield
4. both player have a power sufficient to bypass the opponent shield

For each case, the game theory concept presented in the previous chapter can be applied. To study each state, the expected utility (EU) of each action compared to the other has to be determined. These will be resumed in Table 4.1 to 4.4.

4.4.1 State 1

The first state is equivalent to the beginning of the game when no player has ever **Reload**'ed. Table 4.1 resume the EU. Indeed if both players **Reload**, it's good for both because they unlock their third action which is desirable. **Reload**'ing while the opponent **Shield** is even better because you unlock the **Shoot** action while the opponent stays in the same state and therefore yield a better score. Both players **Shield**'ing is advantageous for no one.

From these observations, we can output that every pair of actions which contains **Reload** are a Pareto equilibrium. Indeed a player can't switch from one strategy to another to increase his

utility without decreasing someone's else.

Now the Nash equilibrium of this game state and optimum strategy for both players is given by both playing the pure strategy **Reload**. Indeed it is always more interesting to play **Reload** than **Shield** in any case. This respect the equilibrium since no player can benefit by changing strategies while the other players keep theirs unchanged.

	Reload	Shield
Reload	1,1	2,0
Shield	0,2	0,0

Table 4.1: No player has reloaded yet.

4.4.2 State 2

The second state is a bit more complex. Now both player can **Shoot** but not kill their opponent when they **Shield**. Therefore **Shoot** on a **Shield** is bad but blocking is good obviously. Whereas **Reload**'ing on a **Shield** is the opposite. The last two new pairs of actions (**Shoot**, **Reload**) and (**Shoot**, **Shoot**) are the win and tie state.

This state and the following being most complex we will only study the Nash equilibrium. Since the game is a finite game it has an equilibrium and one of this three rules must be true for any equilibrium :

- Player 1 plays a pure strategy.
- Player 1 mixes between exactly two pure strategies.
- Player 1 mixes among all three pure strategies.

The same must be true for player 2.

The first statement is not possible since playing only one action is not viable and can be counter-played each time. Playing only two strategies is also not possible. If a player chooses for example to only play a mixed strategy between **Reload** and **Shoot**, the second player can obtain a strictly positive utility by only playing **Shoot**. (In red in Table 4.2). The same yields for a mixed strategy of **Shield** and **Shoot** (in green) and for **Shield** and **Reload** (in blue). For these reason only a combination of the three actions is viable.

The distribution of each strategy can be found with this system based on the table 4.2 :

$$\begin{pmatrix} U_{reload} \\ U_{shield} \\ U_{shoot} \end{pmatrix} = \begin{pmatrix} p_r & p_s & 1 - p_r - p_s \end{pmatrix} \begin{pmatrix} 1 & 1 & -10 \\ -1 & 0 & 1 \\ 10 & -1 & 0 \end{pmatrix}$$

with $U_{reload} = U_{shield} = U_{shoot}$

This gives for both players (since the utility matrix is symmetrical),

$$\begin{cases} p_{reload} = 0.083 \\ p_{shield} = 0.827 \\ p_{shoot} = 0.09 \end{cases}$$

This shows that to reach an equilibrium the players plays a lot of shield, because the win utility score is high (10) and to reduce this influence **Shield** is played a lot. It is also a safe action to play due to the high value of one's player shield. Now, if we reduce for example win utility score from 10 to 1, which means that winning is equivalent to reloading for example, the relation gives,

$$\begin{cases} p_{reload} = 0.33 \\ p_{shield} = 0.22 \\ p_{shoot} = 0.45 \end{cases}$$

With these mixed strategy probabilities, **Shoot** is more efficient and this shows how everybody can tweak his strategy depending of his objective. In this game *a priori*, winning is still the best option and therefore yield more utility.

	Reload	Shield	Shoot
Reload	1, 1	1,-1	-10, 10
Shield	-1, 1	0, 0	1,-1
Shoot	10,-10	-1, 1	0, 0

Table 4.2: No player has sufficient power to overcome shield.

4.4.3 State 3

This state is much comparable to the previous one. Only this time, the second player can kill the first player over his shield, which means that **Reload**'ing more isn't worth it anymore for player 2 and **Shoot** on a **Shield** is now good. Looking for a draw is also valuable for player 1 (but much less than a victory) since he may lose at any moment now.

As before, one equilibrium rule must be true. This time, **Shield** for player one gives always a negative utility which means it will never use it see blue at Table 4.3. For player, it's the action **Reload** (see red). The solution is therefore a mixed strategy of 2 actions this time. Like before the system can be written as,

$$\begin{pmatrix} U_{reload} \\ U_{shoot} \end{pmatrix} = \begin{pmatrix} p_{2s} & 1 - p_{2s} = p_{2kill} \end{pmatrix} \begin{pmatrix} 1 & -10 \\ -1 & 1 \end{pmatrix}$$

with $U_{reload} = U_{shoot}$

This gives for player 2,

$$\begin{cases} p_{2 \text{ shield}} = 0.84 \\ p_{2 \text{ killing shoot}} = 0.16 \end{cases}$$

Now for player 1, the system is,

$$\begin{pmatrix} U_{shield} \\ U_{killingshoot} \end{pmatrix} = \begin{pmatrix} p_{1r} & 1 - p_{1r} = p_{shoot} \end{pmatrix} \begin{pmatrix} -1 & 1 \\ 10 & 0 \end{pmatrix}$$

with $U_{reload} = U_{shoot}$

which gives,

$$\begin{cases} p_{1reload} = 0.08 \\ p_{1shoot} = 0.92 \end{cases}$$

This results show that for this Nash equilibrium player two will tend to protect himself a maximum to place supposedly the killing blow at the best moment, whereas player one wants to kill or trade kill his opponent with much more probabilities.

	Reload	Shield	Killing Shoot
Reload	1, 0	1,-1	-10,10
Shield	-1 ,0	0 ,0	-10 ,10
Shoot	10, -10	-1,1	1,0

Table 4.3: One player has sufficient power to overcome shield.

4.4.4 State 4

This last state happens when both players can kill each other with the **Shoot** action regardless of the opponent action. In this case, only winning is worth utility and so does a tie (but less), cfr. Table 4.4.

In this relation every action is a Pareto optimum, since like previously a player can't switch from one strategy to another to increase his utility without decreasing someone's else. Nevertheless any combination of **Reload** and **Shield** are Pareto dominated by both playing **Shoot**. This last pair of action is also the Nash equilibrium. Indeed, once again no player can benefit by changing strategies while the other players keep theirs unchanged in this state.

The best action is simply to **Shoot** with the high probability that both players will die.

	Reload	Shield	Shoot
Reload	0,0	0,0	-10,10
Shield	0,0	0,0	-10,10
Shoot	10,-10	10,-10	1,1

Table 4.4: Both players have sufficient power to overcome shield.

4.4.5 Interpretation

All of the previous values are strictly theoretical meaning that the Nash equilibrium is only valid when both players follows it. Therefore playing exactly as theory suppose is not valid if both player doesn't follow the same rule. For example in Roshambo playing randomly each action one third of the time is the Nash equilibrium but if the opponent only plays says **Scissors**, this last statement doesn't yield anymore.

If one can forecast their opponent action accurately, they may be able to do much better than the Nash equilibrium strategy. Also why won't the other agents use their Nash equilibrium strategies ? Because they may be trying to forecast your actions too.

Thus this part is meant to give a real understanding of what the best reaction should be following theory but in practice this should not be followed blindly.

This conclude the little game analysis of the game. The next section will show how by using the game concept we created a design for everyone to enjoy it on smartphone.

4.5 The mobile application

One of the outcome of this thesis was the production of the game. You learned the basic concepts of the rules earlier. Now is the time to apply them to something more visual and entertaining, an Android application.

This section will summarize how the game was designed and with which tool and will also depict some of the limitations imposed by a gaming framework and smartphone specifically for our game.

4.5.1 The application

To develop this application, since it is first of all game based, we had to select between some known or maybe less know game framework/engine. Some example can be cited; First of all, Unity¹. This engine gained popularity these last years mostly by "Indie developers", since the engine is free when the user revenue doesn't excess US\$100,000. This framework uses C# and is multiplatform. Another mobile (/multiplatform) engine in JAVA is LibGDX². This latter is totally open-source compared to Unity, but is, as for many open-source software, still in alpha/beta phase. These two software described alone the two larges families of engines, which principally holds in commercials and non-commercials ones.

For the sake of simplicity, stability and the quality of the editor, we choose Unity between many

¹unity3d.com

²libgdx.badlogicgames.com

candidates (for not exhaustively citing all of them).

Unity development is based on a scene approach, meaning that each game screen (different levels, menus, ...) are individually loaded. We made use of two scenes, one for the game, strictly speaking, and another for the main menu. This latter is composed of 3 choices, launch the game, expose the game rules or the credits. A visualization of the menu can be found on Figure 4.1a. As for the game, the gameplay was implicitly dictated by the rules. Therefore we kinda just had to imagine a theme/concept that could encompass the rules. Since the game is designed as a versus fight between an IA and a human, we thought that an alien-learning invasion theme would be appropriate. With that we added a pinch of piracy, because, why not. This led to the game design shown at Figure 4.1b. The pirate is played by the player and the alien/octopus is played by the AI.

Now, how are the actions portrayed. Conceptually, the shooting power of the players are represented by the unveiling of the canons of each ship (respectively 1,3,5 and 10 as shown on the side of the ships) and by the star shaped number on top of the screen (see Figure 4.1b). The shields are pictured as the whale back water stream for the player and a magnetic field force for the AI (see Figure 4.2a). Shields respective values are also written on the shield shaped number on top of the screen. Reload is designed as the floating "Ammo Crate" for the player and the symbol "Ammo" on the Alien screen. A +1 is also shown on the top bar (see Figure 4.2b). Finally any shooting action is depicted with specific projectiles for each power. To conclude this description of the game design, for readability sake, available actions are highlighted with dashed circles.

4.5.2 The constraints

This game was developed with the intent to demonstrate that we could apply machine learning to smartphones and that this hardware is not a limitation in itself. Nevertheless develop on smartphones poses technical constraints. First of all in our case we used a game engine, which imply that everything is designed with a gaming/visual background idea. When you want to apply machine learning you don't have standard matrix functions for example or libraries specifically design for this kind of job, but you still can code it yourself since the framework stays a C# compiler but a simplified one. Indeed you don't have access to all the function that you could use on a desktop version of .NET.

Another limitation is, since we have a game, a fraction of the resources of the device is used to render graphic on the screen, which decrease computation power and processor time allocation for a ML job. In our case, this has nearly no impact since as you will discover later in the chapter describing our final model (Chapter 12) we don't need much computation between each move of a player.

This last concept brings another constraint, responsiveness. You can't expect a player to wait forever between each move to let the AI think, this has a direct implication to the Agent model design.

One more constraint is "backgrounding". Indeed in modern mobile OS, we can have multiple apps opened at the same time. If the main app displayed is too resources consuming, the OS will kill backgrounded application to free up memory and free processor time. For example we made the following test on a Nexus 7 (Android 4 - 2012), launch a Skype video call during the launching of the game (pressing play) causes the OS to close Skype. On newer devices and OS

version (Samsung S6 edge - Android 6 - 2015) this doesn't happen anymore.

One last but somewhat important constraint is I/O. On mobile, it seems to be a blackhead. Reading files is less a problem whatsoever, but writing really is, recall section 2.2.2. For android device, many doesn't have dedicated storage and need external SD memory (with possibly bad performance) or have inside storage with really poor capabilities. Once again recent device seems to improve in this matter. With the same devices as mention before (and internal memory), 40mo of raw data takes 35 second on the older device and merely 3.5 on the other. For these reasons saving time are sometimes a bit long on older device. So was the loading part too, since it needed to load all the graphical part of the game and set up the AI, but thanks to some tricks (a main menu for example) we manage to "hide" some of these to make the application look faster.

This concludes some of the highlights of mobile development. This section demonstrated the mobile application, but what also interesting is to know and understand how the AI inside the game was designed. This is the subject of Part II. Therefore let's jump to the conclusion this part and find out what lies ahead.

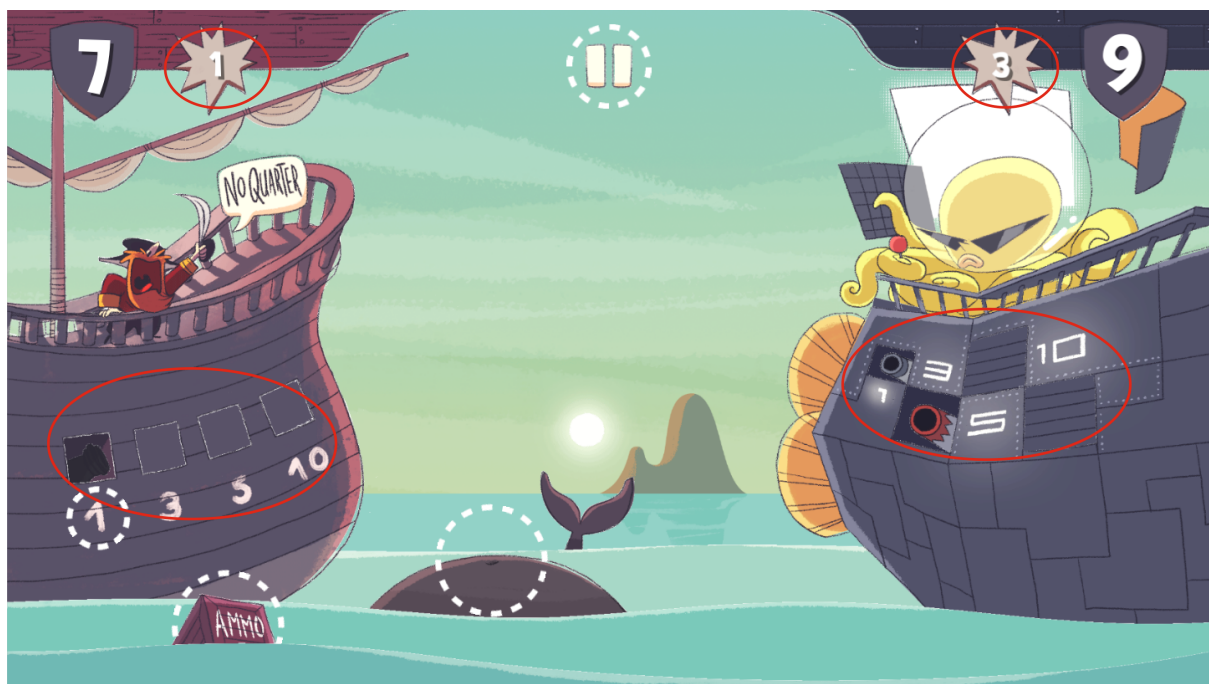
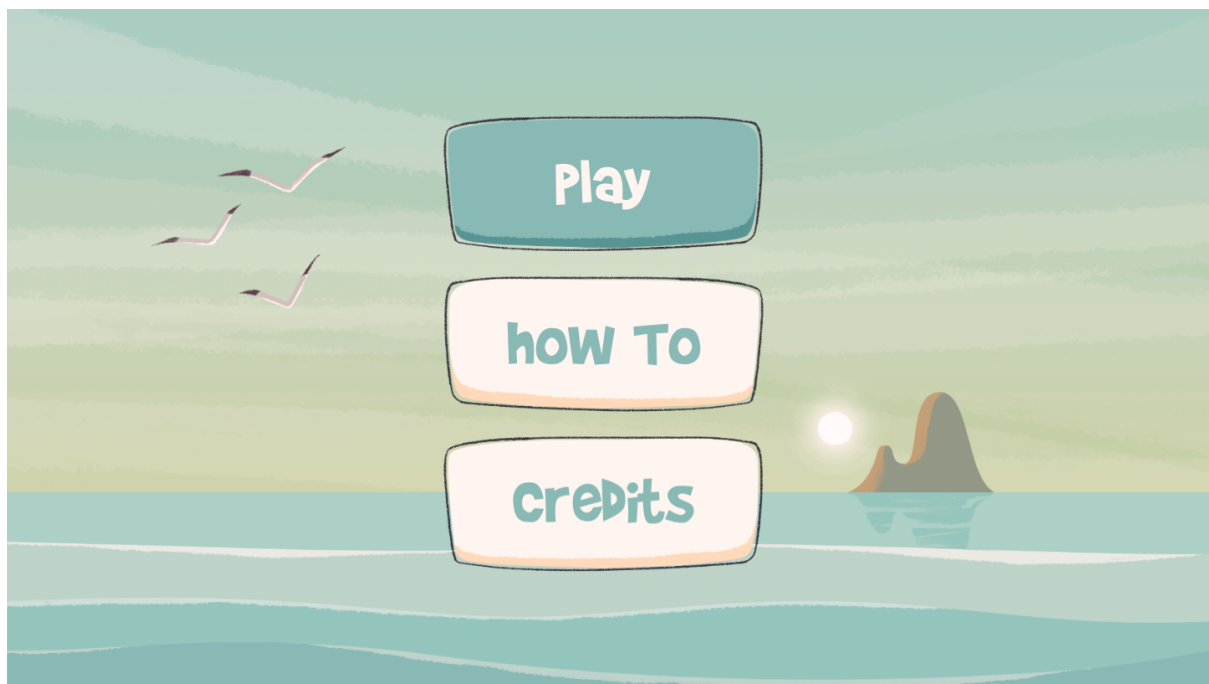
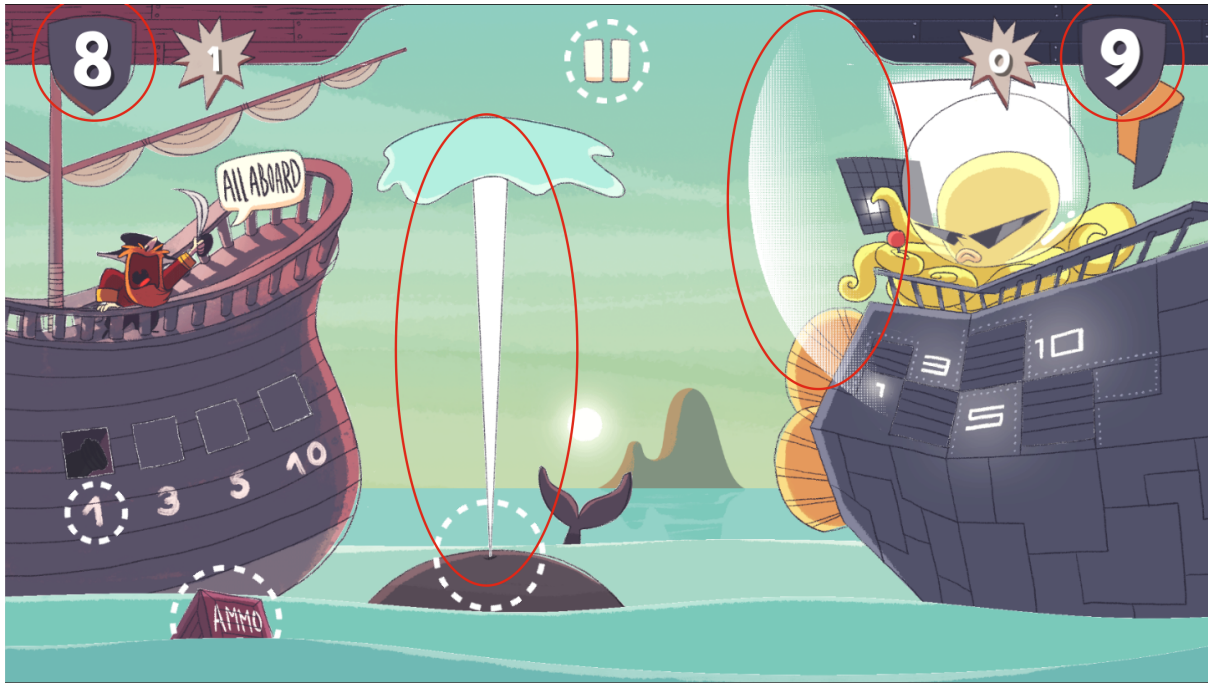
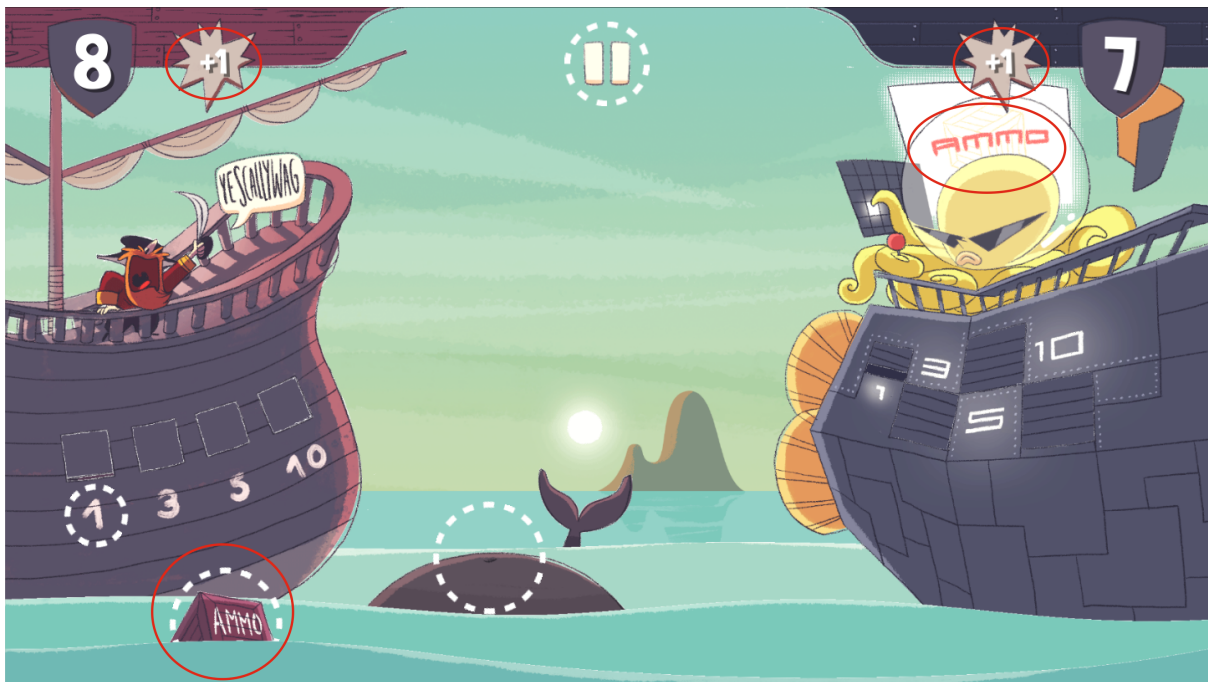


Figure 4.1: Game Design (1)



(a) Game : Shields



(b) Game : Reload

Figure 4.2: Game Design (2)

Chapter 5

Conclusion

After this first part, the basis of the thesis has been set up. Indeed to be able to develop and analyze the concept of machine learning for smartphone game, we needed to put mobile devices, games and machine learning somehow into perspective.

From these it could be learned that creating a virtual challenger for the human brain has been a long and incredible journey that started with Claude Shannon's belief and continued till today and for many years to come.

Also we stumbled on a new kind of device, the smartphone, and discovered that although it is relatively new, mobile device possesses some interesting capabilities in terms of computation. Indeed it is still unable to achieve jobs as easily as a computer but even with its restrictions which can be summarized as slow (hard) memory, limited heap space, restricted background processes, heat threshold, limited battery and the requirement of high optimization, they can still be useful in many cases. Phones also have advantages like, on device data access, latency, nomadism, internet access if necessary or simply said a reduced version of a laptop. The jobs that most successfully suits smartphones are those who benefit a maximum from only RAM access and limited resources from disk or swap. Meaning that either a program has basically these characteristics or has to be optimized to fit these requirements by reducing their complexity or adjusting the computation/loading to only fit in cache if possible. Successful examples like Google's or Qualcomm's already exist but point out the complexity of it as much as the rewards.

Last but not least, the thesis hopped in the game world through game theory. This latter was used to analyze thoroughly the game we choose which is as a reminder the *Cowboy Game*. The outcome of the analyses gave the optimal strategy to follow in 4 main stages of the game and helped to understand the quality of each move depending of the situation. This strategy although optimal is only meaningful if both players use it which is unlikely since the opponent and ourselves may try to forecast the enemy's moves.

This brings us to the continuity of this thesis, be able to predict and respond accurately to the opponent strategy. This motive introduces subtly machine learning as a means to study the adversary and use this knowledge to beat him. We already set the basis by applying the basic notion of ML to the game and clarified how an agent could achieve good performances. The next part will cover machine learning but more specifically a subpart of it, reinforcement learning. The ultimate goal being to create an agent able to compete against a human being. Once achieved this agent purpose is to be implemented in the smartphone game presented before and be playable by everyone equipped with an Android smartphone. This application is also a means

to prove that the method/model is applicable to mobile phone with the additional constraint that a graphical interface must be supported too.

Let's now dive in the second part which will this time apply machine learning to the *Cowboy Game* and thanks to an iterative process reach an agent able to compete against pre-programmed (semi-)static strategies and humans.

Part II

Algorithms and experiments

At this step, we precisely defined our task and how to prove the quality of an agent with the performance metrics but a part of the experience is missing. Indeed, we don't know the action to play at each state yet and consequently we need an efficient model to determine it. This agent mustn't be a real time constraint and consequently shouldn't use a lot of memory since this is as shown before the principal problem of smartphones.

To do so, Each next chapter of this part describes an improvement of the previous algorithm in 3 sections. The first one introduces theoretically the technique used. Then, the second one challenges the algorithm to assess its performances according to the so called performance metrics (section 4.3). This section isn't relevant in chapter 6. This fact will be explained later. Finally, The last section analyses the drawbacks to find some enhancements. The enhancements leads to the next algorithm.

The selected starting point and subject of chapter 6 is a derivative of Markov Chain called Markov decision processes (MDP). This choice can simply be explained because MDP is often used to easily represent systems containing non determinism. In this game, it is caused by concurrency (a player doesn't know the next action of its opponent).

Chapter 6

Markov decision process

Remind that the ultimate goal of a game is obviously to win. Therefore a player wants to find the best way to reach this goal state from the initial state of the game. One of the famous model to show such a transition is the Markov Chains theory. Moreover, according to the fact we have a stochastic game, this model is easy to apply. In fine we will upgrade this definition to reach a more complete and complex theory called Markov decision process. The transition between these 2 concepts is the purpose of section 6.1. After this theoretical point, we talk about the limitation of MDP according to the cowboy game in section 6.2.

6.1 From Markov chain to Markov decision process

6.1.1 Markov chain

Let's first introduce the process called Markov Chain, since it's the basis of the reasoning to obtain our algorithm. Assuming a finite set of (discrete) states, $Q = \{q_0 \dots q_{n-1}\}$. $X_t = q_i$ (with $i \in [0, n-1]$) means that the process/system is in state q_i at time t . It can be fully defined with the help of [61].

Definition 6.1.1. A discrete time finite Markov Chain of order p is a stochastic process $\{X_t | t \in \mathbb{N}\}$ where the random variable X takes its value at any discrete time t in Q and such that:

$$P(X_t = q_i | X_0, \dots, X_{t-2}, X_{t-1}) = P(X_t = q_i | X_{t-1} = q_j) \triangleq P(q_i | q_j)$$

This definition makes 2 assumptions :

- The next state depends only on the current one and not all the previous states (Markov property)
- Stationary : $P(q_i | q_j) = P(X_t = q_i | X_{t-1} = q_j)$ doesn't depend of t .

This said it is easy to represent it as a finite state machine (an abstract machine that can be in one of a finite number of states). $P(q_i | q_j)$ being the transition probability between states $q_i \in Q$ and $q_j \in Q$. This simply define the transition matrix $A = [P(q_i | q_j)]$. This matrix has 2 properties :

- $\forall q_j \in Q, \sum_{q_i \in Q} A_{ij} = 1$. In other words, for each row the sum of the elements is 1
- The transition probabilities are also stationary. They don't depend of t .

The associated initial probability vector π can be defined with

$$\forall q_i \in Q, \pi_i = P(X_0 = q_i)$$

This matrix is associated with the property that,

$$\sum_{q_i \in Q} \pi_{q_i} = 1.$$

Thus, a finite state machine can be fully described by (Q, A, π) defined in definition 6.1.2.

Definition 6.1.2. The formal definition of a Markov Chain model is :

- Q a set of discrete states
- A a $|Q| \times |Q|$ transition probability matrix
- π an initial probability distribution

Definition 6.1.3. From the known Bayes rules, the product rule can be inferred :

$$P(A_0, A_1) = P(A_1|A_0)P(A_0)$$

The *chain rule* extends this formula to k variables:

$$P(A_0, A_1, \dots, A_k) = P(A_k|A_{k-1}, \dots, A_0) \dots P(A_2|A_1, A_0)P(A_1|A_0)P(A_0)$$

Thanks to the chain rule (see Definition 6.1.3 adapted from the website [62] of the Queen Mary University of London) and the Markov chain assumption, the probability of a succession of k states $(s_0, \dots, s_k) \in Q$ can be computed like this :

$$\begin{aligned} P(X_0 = s_0, \dots, X_k = s_k) &= P(s_0^k) \\ &= P(s_0)P(s_1|s_0)P(s_2|s_1) \dots P(s_k|s_{k-1}) \\ &= P(s_0) \prod_{i=1}^k P(s_i|s_{i-1}) \\ &= \pi[s_0] \prod_{i=1}^k A[s_{i-1}, s_i] \end{aligned} \tag{6.1}$$

With s_0 the initial state, s_1 the second state and so on. This definition can be used to predict the next state given a succession of states.

6.1.2 Markov Decision Process

The Markov Chain although being a formidable tool doesn't allow us to fully represent our game/agent, since the idea is to produce an AI capable of decision. Therefore it needs a means to weight his decisions between several actions.

Consequently, we need to introduce a new model, the Markov Decision Process. It is a Markov Chain (it follows the same hypothesis) but with actions and rewards (useful to make the best possible choice). If you have a Markov Decision Process with one action and if all rewards are equivalent, this model is a Markov Chain.

Definition 6.1.4. More formally, a Markov Decision Process (MDP) is (adapted from Wikipedia) :

- Q a set of discrete states
- *Actions* a set of discrete actions. For each state q_i , $act_{q_i} \subseteq Actions$ is the set of available actions at this state.
- One transition probability matrix (A) for each $a \in act_{q_i}$ at state q_i :
 $A_a[q_i, q_j]$ is the probability that action a in state q_i at time t will lead to the state q_j at time $t + 1$ (Markov assumption)
- One reward matrix (R) for each $a \in act_{q_i}$ at state q_i :
 $R_a[q_i, q_j]$ is the immediate reward in state q_i at time t and in q_j at time $t + 1$ and by taking action a .

To form the MDP model it also need to define the policy which is the strategy an agent will deploy at a given state to reach a goal.

A policy π is thus a set of actions, one for each state which specify the action to play at each state. $\pi(q_i)$ is the action to play at state q_i .

In a MDP the goal is to reach from some initial state s_0 the goal state s_d . This latter has two properties :

- $\forall a \in act_{s_d}, A_a[s_d, s_d] = 1$ (absorbing state)
- $\forall a \in act_{s_d}, R_a[s_d, s_d] = 0$ (no more rewards)

The purpose of a MDP algorithm is to find the best policy π^* (best action for each state) to reach s_d . To do that, it uses a cumulative function of the rewards to maximize. Typically, this function is defined as $V_\pi(s_0)$ (see equation 6.2), and is referenced as the total expected infinite-horizon discounted reward following a policy π .

This definition includes γ , the discount factor. It gives importance to future rewards according to immediate rewards. If $\gamma = 1$, it means that each reward has the same importance independently from the time. If $\gamma = 0$, more importance is given to immediate rewards.

$$V_\pi(s_0) = V_\pi(X_0 = s_0) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R_{\pi(X_t)}[X_t, X_{t+1}] \middle| X_0 = s_0, \pi \right], \gamma \in [0, 1] \quad (6.2)$$

Other models exist but this one is adequate for a game without a fixed game length because it tries to maximize the total sum of the reward and can give more importance to immediate rewards. This model is the most common used in reinforcement learning because it can bound the sum (purpose of γ since it is between 0 and 1 and t is increasing over time).

With this expression, π^* can be found by maximizing the total expected rewards. This define the equation 6.3 of $V^*(s_0)$

$$V^*(s_0) = \max_{\pi} V_{\pi}(s_0) \quad (6.3)$$

The application of equation 6.2 to 6.3 is resumed at equation 6.4

$$V^*(s_0) = \max_{\pi} \left(\mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R_{\pi(X_t)}[X_t, X_{t+1}] \middle| X_0 = s_0, \pi \right] \right) \quad (6.4)$$

To simplify this equation 6.4, the definition of the expectation can be invoked (if X can have values $x_1, x_2, \dots, x_n$ with probabilities $p_1, p_2, \dots, p_n$, we have $\mathbb{E}[X] = x_1 p_1 + x_2 p_2 + \dots + x_n p_n$) resulting to the equation 6.5. In this latter, $PATH_{\pi}(s_i)$ is the set of available path following the policy π from s_i to s_d . A path is a succession of states.

$$V^*(s_0) = \max_{\pi} \left(\sum_{(s_1, s_2, \dots) \in PATH_{\pi}(s_0)} \left(P(X_1 = s_1, X_2 = s_2, \dots | X_0 = s_0, \pi) \left[\sum_{t=0}^{\infty} \gamma^t R_{\pi(s_t)}[s_t, s_{t+1}] \right] \right) \right) \quad (6.5)$$

To explicit this equation even more, the conditional probability definition can be used to derive $P(a, b|c) = P(a|b, c) * P(b|c)$ with

$$\begin{aligned} a &: X_2 = s_2, X_3 = s_3, \dots \\ b &: X_1 = s_1 \\ c &: X_0 = s_0 \end{aligned}$$

In our case, equation 6.6 is obtained.

$$V^*(s_0) = \max_{\pi} \left(\sum_{(s_1, s_2, \dots) \in PATH_{\pi}(s_0)} \left(P(X_2 = s_2, X_3 = s_3, \dots | X_1 = s_1, X_0 = s_0, \pi) \right. \right. \\ \left. \left. P(X_1 = s_1 | X_0 = s_0, \pi(s_0)) \left[\sum_{t=0}^{\infty} \gamma^t R_{\pi(s_t)}[s_t, s_{t+1}] \right] \right) \right) \quad (6.6)$$

Using the fact that $P(X_1 = s_1 | X_0 = s_0, \pi(s_0)) = A_{\pi(s_0)}[s_0 | s_1]$ (see definition 6.1.4), the Markov Property and then basic algebra, the bellman equation for s_0 can be deduced at 6.7.

$$\begin{aligned}
V^*(s_0) &= \max_{\pi} \left(\sum_{(s_1, s_2, \dots) \in PATH_{\pi}(s_0)} \left(P(X_2 = s_2, X_3 = s_3, \dots | X_1 = s_1, \pi) A_{\pi(s_0)}[s_0 | s_1] \right. \right. \\
&\quad \left. \left. \times \left[\sum_{t=0}^{\infty} \gamma^t R_{\pi(s_t)}[s_t, s_{t+1}] \right] \right) \right) \\
&= \max_{\pi} \left(\sum_{(s_1, s_2, \dots) \in PATH_{\pi}(s_0)} \left(P(X_2 = s_2, X_3 = s_3, \dots | X_1 = s_1, \pi) A_{\pi(s_0)}[s_0 | s_1] \right. \right. \\
&\quad \left. \left. \times \left[\gamma^0 R_{\pi(s_0)}[s_0, s_1] + \sum_{t=1}^{\infty} \gamma^t R_{\pi(s_t)}[s_t, s_{t+1}] \right] \right) \right) \\
&= \max_{\pi(X_0)} \max_{\{\pi(X_1), \pi(X_2), \dots\}} \left(\sum_{s_1 \in Q} A_{\pi(s_0)}[s_0 | s_1] \sum_{(s_2, s_3, \dots) \in PATH_{\pi}(s_1)} P(X_2 = s_2, X_3 = s_3, \dots | X_1 = s_1, \pi) \right. \\
&\quad \left. \times \left[R_{\pi(s_0)}[s_0, s_1] + \sum_{t=1}^{\infty} \gamma^t R_{\pi(s_t)}[s_t, s_{t+1}] \right] \right) \\
&= \max_{\pi(X_0)} \left(\sum_{s_1 \in Q} A_{\pi(s_0)}[s_0 | s_1] \times \left[R_{\pi(s_0)}[s_0, s_1] + \right. \right. \\
&\quad \left. \left. \max_{\{\pi(X_1), \pi(X_2), \dots\}} \sum_{(s_2, s_3, \dots) \in PATH_{\pi}(s_1)} \left(P(X_2 = s_2, X_3 = s_3, \dots | X_1 = s_1, \pi) \sum_{t=1}^{\infty} \gamma^t R_{\pi(s_t)}[s_t, s_{t+1}] \right) \right] \right) \\
&= \max_{\pi(X_0)} \left(\sum_{s_1 \in Q} A_{\pi(s_0)}[s_0 | s_1] \times \left[R_{\pi(s_0)}[s_0, s_1] + \gamma \right. \right. \\
&\quad \left. \left. \underbrace{\max_{\{\pi(X_1), \pi(X_2), \dots\}} \sum_{(s_2, s_3, \dots) \in PATH_{\pi}(s_1)} \left(P(X_2 = s_2, X_3 = s_3, \dots | X_1 = s_1, \pi) \sum_{t=0}^{\infty} \gamma^t R_{\pi(s_{t+1})}[s_{t+1}, s_{t+2}] \right)}_{V^*(s_1)} \right] \right) \\
&= \max_{a \in act_{s_0}} \left(\sum_{s_1 \in Q} A_a[s_0 | s_1] (R_a(s_0, s_1) + \gamma V^*(s_1)) \right) \\
&= \text{Bellman equation for } s_0
\end{aligned} \tag{6.7}$$

The optimal action for each state verifies exactly the Bellman equation. Thus, for each state s_i , the relation 6.8 is valid :

$$V^*(s_i) = \max_{a \in act_{s_i}} \left(\sum_{s_{i+1} \in Q} A_a[s_i | s_{i+1}] (R_a(s_i, s_{i+1}) + \gamma V^*(s_{i+1})) \right) \tag{6.8}$$

To solve these equations, a common way is to use dynamic programming as the value-iteration procedure (algorithm 1) which simply repeats the second equation of system 6.9 for each state $q_i \in Q$ until convergence (with $\gamma = 1$).

$$\begin{cases} \hat{V}_0(q_i) = 0 \text{ if } q_i \neq \text{destination} \\ \hat{V}_{k+1}(q_i) = \max_{a \in \text{act}_{q_i}} \left(\sum_{\text{available next states, } q_k} A_a[q_i, q_k] \left(R_a(q_i, q_k) + \hat{V}_k(q_k) \right) \right), \text{ if } q_i \neq \text{destination} \\ \hat{V}(\text{destination}) = \text{Final reward} \end{cases} \quad (6.9)$$

Algorithm 1: Value-Iteration algorithm

Data: maxIter, ϵ , R, A
Result: $\hat{V}, \hat{\pi}$ for each state q_i

```

1   $t \leftarrow 0$ ;
2  repeat
3    for each state  $q_i$  do
4      if state is the destination (goal) then
5         $\hat{V}_t(q_i) = \text{final reward}$ ;
6      else
7         $\hat{V}_t(q_i) = \max_{a \in \text{act}_{q_i}} \left( \sum_{k' \in \text{next states}} A_a[q_i, k'] \left( R_a(q_i, k') + \hat{V}(k') \right) \right)$ ;
8         $\hat{\pi}_t(q_i) = \arg \max_{a \in \text{act}_{q_i}} \left( \sum_{k' \in \text{next states}} A_a[q_i, k'] \left( R_a(q_i, k') + \hat{V}(k') \right) \right)$ ;
9       $t++$ ;
10 until  $t = \text{maxIter}$  or  $\max_{q_i \in Q} |\hat{V}_t(q_i) - \hat{V}_{t-1}(q_i)| < \epsilon$ ;
11 return  $\hat{V}_t, \hat{\pi}_t$ ;
```

The value-iteration algorithm goal is to determine the optimal strategy to apply for a given problem. But this algorithm can't be used for our problem. The next section will state why.

6.2 Drawback

The main problem with this Algorithm 1 is that it needs the rewards matrix and transition matrix as input. So, a MDP needs the entire model of the system. This means a static representation of the problem/game. This could be provided if we'd knew our adversary and how each action in each state should be scored reward wise, but this is not the case.

Furthermore the means of this model is to predict how an entire game will happen for every possible state and iterate till it reaches convergence. This require a lot of computation and time which are not much available in our device (a smartphone) and we face here a pretty quick and responsive game.

All these facts lead to the fact that we need a more responsive model which requires less computation and can evolve against his opponent behavior. Of course, all these qualities comes with defaults like the lost of optimality. The next chapter will introduce the Q-learning notion to fill some of the loopholes of value-iteration.

Chapter 7

Q-learning

Due to the previous drawback, a solution to approximate the Markov decision process formula is needed. This approximation must consider the fact that the transition matrix is unknown. The purpose of the section 7.1 is to discuss the theoretical aspects of one of these solutions. Q-learning has been chosen as a starting point. The next section (7.2) is all about challenging this Q-learning agent and section 7.3 will highlight the reason of the troubles seen during the experiments according to the studied game.

7.1 Q-learning algorithm and Reinforcement learning

This section addresses the theoretical aspects to understand the Q-learning method. To do so, the first subsection 7.1.1 is all about the transition from the Value Iteration algorithm to the famous approximation called Q-learning. This demonstration is made with the help of 2 courses given in EPL : INGI2263 [61] and SINF2275 [63]. The final method is part of a specific branch of Machine Learning called Reinforcement Learning. This domain will be explained at subsection 7.1.2.

7.1.1 From Value Iteration to Q-learning

The first step leading towards Q-learning is the introduction of a new quantity called the Q-value (equation 7.1), $Q(q, a)$. It is the expected reward when choosing action a at state q and then following a policy π .

$$Q(q_i, a) = \sum_{\text{available next states, } q_k} A_a[q_i, q_k] (R_a(q_i, q_k) + \gamma V(q_k)) \quad (7.1)$$

With it the relation 7.2 can be extracted from the Bellman formula (equation 6.8).

$$V(q_i) = \max_{a \in \text{act}_{q_i}} Q(q_i, a) \quad (7.2)$$

Combining equation 7.2 in 7.1 and the expectation formula, the equation 7.3 can be found.

$$\begin{aligned} Q(q_i, a) &= \sum_{\text{next states, } q_k} A_a[q_i, q_k] \left(R_a[q_i, q_k] + \gamma \max_{a' \in \text{act}_{q_k}} Q(q_k, a') \right) \\ &= \mathbb{E}[R_a[q_i, q_k] + \gamma \max_{a' \in \text{act}_{q_k}} Q(q_k, a')] \end{aligned} \quad (7.3)$$

Therefore, the Value-Iteration algorithm can be rewritten thanks to equation 7.3 and it gives relations 7.4:

$$\begin{cases} \hat{Q}_0(q_i, a) = 0 \text{ if } q_i \neq \text{destination} \\ \hat{Q}(q_i, a) = \mathbb{E}[R_a[q_i, q_k] + \gamma \max_{a' \in \text{act}_{q_k}} Q(q_k, a')], \text{ if } q_i \neq \text{destination} \\ \hat{Q}(\text{destination}, a) = \text{Final reward} \end{cases} \quad (7.4)$$

Lets now introduce a stochastic approximation to find an approximation of the expected value in equation 7.4. Indeed our goal is to make the transition matrix disappears from these equations and for now, it is still needed to compute the expectation.

For that lets imagine you want to find $\mathbb{E}[z]$, the expectation of a random variable z based on empirical realizations $z(t)$ appearing sequentially at each t (one observation by time step) instead of with the mathematical formula. The mean at time t , $m(t)$, can be defined as in equations 7.5.

$$\begin{aligned} m(t) &= \frac{1}{t} \sum_{k=1}^t z(k) \\ &= \frac{1}{t} \left(z(t) + \sum_{k=1}^{t-1} z(k) \right) \\ &= \frac{1}{t} z(t) + \frac{1}{t} \sum_{k=1}^{t-1} z(k) \\ &= \frac{z(t)}{t} + \frac{t-1}{t} \frac{1}{t-1} \sum_{k=1}^{t-1} z(k) \\ &= \frac{z(t)}{t} + \frac{t-1}{t} \left(\frac{1}{t-1} \sum_{k=1}^{t-1} z(k) \right) \\ &= \frac{z(t)}{t} + \frac{t-1}{t} m(t-1) \\ &= \frac{z(t)}{t} + m(t-1) - \frac{1}{t} m(t-1) \\ &= m(t-1) + \frac{1}{t} (z(t) - m(t-1)) \end{aligned} \quad (7.5)$$

And so the update rule (with $0 < \alpha \leq 1$ because $t > 0$) equation 7.6 can be extracted.

$$\hat{m} \leftarrow \hat{m} + \alpha(t)(z(t) - \hat{m}) \quad (7.6)$$

An important observation is that if $\alpha(t) = \frac{1}{t}$ is selected, this equation computes the exact mean. So, $\alpha(t)$ must decrease with t and be between 0 and 1. Finally, from 7.6 and 7.3, the final result

independent of the transition matrix follows equations 7.7.

$$\begin{cases} \hat{Q}(q_i, a) \leftarrow \hat{Q}(q_i, a) + \alpha \left((R_a[q_i, q_k] + \gamma \max_{a' \in \text{act}_{q_k}} Q(q_k, a')) - \hat{Q}(q_i, a) \right), & \text{if } q_i \neq \text{destination} \\ \hat{Q}(\text{destination}, a) \leftarrow \text{Final reward} \\ 0 < \alpha \leq 1, \forall t, q_i \\ 0 \leq \gamma < 1, \forall t, q_i \end{cases} \quad (7.7)$$

7.1.2 Reinforcement Learning

As said before, Q-learning is one of the most common algorithms for reinforcement learning. Here is the formal definition of Reinforcement learning as found in [64] :

Reinforcement learning (RL) is learning by interacting with an environment. An RL agent learns from the consequences of its actions, rather than from being explicitly taught and it selects its actions on basis of its past experiences (exploitation) and also by new choices (exploration), which is essentially trial and error learning. The reinforcement signal that the RL-agent receives is a numerical reward, which encodes the success of an action's outcome, and the agent seeks to learn to select actions that maximize the accumulated reward over time.

F. Woergoetter and B. Porr

The figure 7.1 explains this definition. Agent interacts with its environment with actions. As a consequences of each chosen action it receives a reward and a state. And so the agent can learn and acquire experience with the help of these two feed-backs from the environment. It describes RL with a single agent but it can be extrapolate for multi-agents (each agent communicates with the environment).

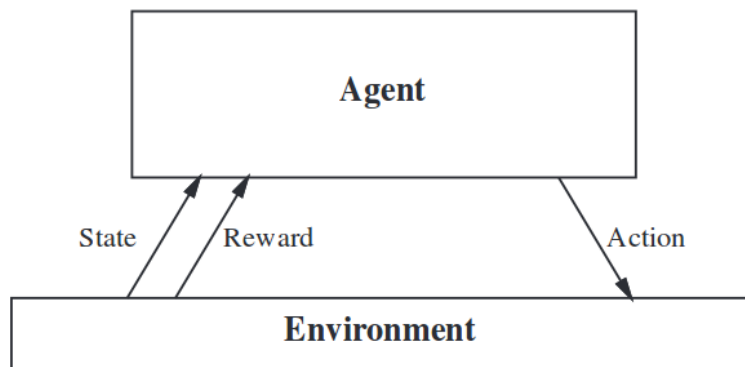


Figure 7.1: The reinforcement learning problem (from [9])

There is 2 classes of algorithms which solve this constraint in reinforcement learning :

- Model-based : the algorithm approximates the model and then uses a method like dynamic programming to estimate the optimal solution/action for each state.

- Model-free : the algorithm approximates directly the optimal solution/action for each state.

It is easy to understand that Q-learning is a model-free method. It is the chosen starting point because of this fact. Indeed it isn't necessary to build a model and so we thought it is basically easier to implement. Moreover, literature on reinforcement learning talks a lot about this promising method.

Now that we have access to a functioning model it's time to try it out.

7.2 Experiments

This section contains multiple parts to eventually assess the performance of this first algorithm. First, it seems logic that a huge number of games has to be played to properly consider an agent as good or bad. Consequently, a simulator and bots are needed. Concerning the simulator, it is described in appendices. The subsection 7.2.1 talks about the first wave of created bots. Then, the subsection 7.2.2 will be about the adaptations needed to use equations 7.7 in the framework of the cowboy game and will describe the pseudo-code used for the experiments presented under subsection 7.2.3. These experiments will be used to observe how well this first agent behaves.

7.2.1 Bots

As stated before, bots are a crucial ingredient in the testing phase of an AI agent. Indeed since they are made from very simple to more complex patterns, it is really interesting to observe how an agent adapts itself against such behavior. Moreover, it is mandatory to simulate a huge number of games (unless you have a lot of time to waste) and to compare multiple agents on the same basis.

Different things are important to assess the performances of an agent. First of all, in the early phases of development, the primary interest is to state if the agent can understand the game rules. For this purpose simple very patterned bots are sufficient, e.g. always play some moves in the same order. With these version, it is intuitive that there is an optimal strategy and that a smart agent can't loose once it understands the "rules" and how the opponent plays.

The following subsection will describe the bots used and some of their characteristics/restrictions.

Bots description

A succinct description of the bots is given thereafter in List 7.1. Table 7.1 also represent how bots behave against themselves. This table holds all the statistical wins of every bot on 1000 games. Scores are computed according to the equation 4.1.

From the table it is easy to point out bots/strategies that seems better. It also proves that bots are well distributed between many wins, many loses or 50/50. They don't really outperform each other, they are just very different.

Unlike some other games like Roshambo, a complete random agent seems not to be a good idea. Indeed the bot will most likely play action that will not improve his current state, for example shield many times without having a threatening power (no **Reload**). But when you weight the random toward the winning action **Shoot** (**Bot11**), it significantly improve his win rate. Nevertheless the medal of the "best" strategy, is still awarded to another kind of bot that

abuse of the **Shield** action (**Bot14**). Indeed since the shield has a high value, it's a safe action to spam. This also consistent with the theoretical analysis of the game at Section 4.4. Still these bots are not perfect.

Among all these, the smarter one with the most dynamic strategy depending on the opponent is **Bot6**. This is also an important thing to remember because it is the most human like for now.

- **Bot1-2** (not_)last_move_agent : Play or don't play last opponent move.
- **Bot3-4** (max/min)_played_agent : Play min/max opponent played action.
- **Bot5** basic_pattern_agent : Counter play on longest adversary matching pattern.
- **Bot6** pattern_agent : Size based matching pattern.
- **Bot7** play_last_best_game_agent : Play the last won game of the enemy.
- **Bot8** random_agent : Simple random on playable moves.
- **Bot9-11** weighted_(MOVE)_agent : Biased random on one move = {**reload**, **shield**, **shoot**} at 60 percent.
- **Bot12** reload_shoot_agent : **reload** then **shoot**.
- **Bot13** rand_reload_shoot_agent : n random $\times$ **reload(s)** then **shoot**.
- **Bot14** RSS_agent : **reload** followed by n random $\times$ **shield(s)** and then **shoot**, repeat.

List 7.1: Bot description

vs	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1		20	33	50	1	50	50	48	40	40	48	50	50	84
2	80		99	85	58	45	1	61	74	62	57	50	77	20
3	67	1		1	50	1	1	30	11	34	7	1	1	20
4	50	15	99		69	1	43	65	79	69	56	24	30	30
5	99	42	50	31		1	1	45	60	50	35	50	44	21
6	50	55	99	99	99		97	47	80	60	43	90	83	39
7	50	90	99	57	99	3		68	80	50	60	50	81	50
8	52	39	70	35	55	53	32		66	55	42	25	45	26
9	60	26	89	21	40	20	20	34		42	25	13	30	16
10	60	38	66	31	50	40	50	45	58		36	23	44	32
11	52	43	93	44	65	57	40	58	75	64		32	56	28
12	50	50	99	76	50	10	50	75	87	77	68		83	31
13	50	23	99	70	56	13	19	55	70	56	44	27		19
14	16	80	80	70	79	61	50	74	86	68	72	69	81	

Table 7.1: Bot vs Bot win ratio.

Bot restrictions

Although they are very useful to get an insight of the agents, bots suffer from various limitation. Indeed the objective is to create an AI that reacts to the human behavior who is an adaptive one. Bots are limited because, except **Bot6**, they always play the same pattern and never react to what's happening to them or improve themselves (still some bots are less predictable like, playing the agent last game or more randomized strategies). One other characteristic is that for most of them there is a counter strategy that should obtain 100% win rate.

7.2.2 Pseudo-code and adaptations

First of all, the algorithm 2 is the pseudo-code of the Q-learning agent. Firstly, α and γ are considered as constant over time.

Algorithm 2: Q-learning	
1	$S_0 \leftarrow$ initial state of the game
2	$A \leftarrow$ set of actions
3	$\alpha \leftarrow$ the learning rate ($0 < \alpha \leq 1$)
4	$\gamma \leftarrow$ the discount factor ($0 \leq \gamma < 1$)
5	foreach <i>game</i> do
6	Initialize $S=S_0$
7	foreach <i>round i</i> do
8	if <i>State S_i not already explored</i> then
9	Initialize $Q(S_i,A)$ to 0
10	$B_i \leftarrow$ subset of playable actions in A at round i
11	Take playable action $a \in B_i$ maximising $Q(S_i,B_i)$, observe reward R and next state S_{i+1}
12	$Q(S_i,a) \leftarrow (1-\alpha)Q(S_i,a) + \alpha(R + \gamma \max_{b \in B_{i+1}} Q(S_{i+1},b))$

By analyzing it, 2 things remain to be defined for this game. This is the purpose of this subsection. The first one is the representation of a state of the game and the second one is the immediate reward R observed after each played action.

Representation of a state S

It exists plenty of representation of a state but it will be later proven by experience (chapter 8) that the representation described below is the best one among all those we tried.

The current state of the game will be defined with 4 integers :

- The power of its shoot
- The solidity of its shield
- The power of the opponent shoot
- The solidity of the opponent shield

The table 7.2 shows an example of the evolution of the state during a game. The table on the right contains the decision taken by each player for each round. The other table is a representation of each state of this game.

	Player 1		Player 2	
<i>State</i>	<i>Reload</i>	<i>Shield</i>	<i>Reload</i>	<i>Shield</i>
S_0	0	9	0	9
S_1	1	9	0	9
S_2	0	9	0	8
S_3	1	9	1	8
S_4	0	9	2	8
Player 1 wins				

(a) The definition of each state

Round	Transition	Decision P1	Decision P2
1	$S_0 \rightarrow S_1$	<i>RELOAD</i>	<i>SHIELD</i>
2	$S_1 \rightarrow S_2$	<i>SHOOT</i> ₁	<i>SHIELD</i>
3	$S_2 \rightarrow S_3$	<i>RELOAD</i>	<i>RELOAD</i>
4	$S_3 \rightarrow S_4$	<i>SHOOT</i> ₁	<i>RELOAD</i>
Player 1 wins			

(b) The decision of each player at each round

Table 7.2: A cowboy game between 2 players with, as state, a set of status

Immediate reward R

The only rule this reward must respect is that it can't be influenced by the developer assumptions. The immediate reward function is described algorithm 3.

Algorithm 3: Immediate reward function

```

1 if Agent wins then
2   | return +10
3 else if Agent loses then
4   | return -10
5 else if Agent puts its power to 0 without killing the enemy then
6   | return -1
7 else if Agent uses reload and opponent shield then
8   | return +1
9 else if Agent uses shield and opponent reload then
10  | return -1
11 else
12  | return 0

```

The 2 first rules doesn't need to be explained. The third one is logical because if the power of the agent becomes 0, it means he has only 2 possible actions for the next round and so it becomes really predictable. Moreover, the opponent knows he shouldn't use his shield. So it's clearly a disadvantage. The two last rules exist because a shield is useless against a reload. This function is used because it isn't too restrictive and doesn't make any assumption about the behavior of the opponent and how to play.

7.2.3 Performance assessment

It is now time to assess this agent and consequently analyze how it reacts against the bots of the list 7.1. Table 7.3 gathers the average wins rate (according to the performance metrics

described section 4.1) of this algorithm against each bot after 20/100/500/1000 games without any training. The simulation is repeated 5 times and averaged. Therefore for a result to be meaningful only values for which the tie rate is below 50% are kept. This performance check is not an optimization state. Therefore α and γ are fixed to 0.5 because the only things known about these parameters are their boundaries (between 0 and 1).

	Number of games			
	20	100	500	1000
Bot1	46	85	99	99
Bot2	11	49	78	84
Bot3	86	96	98	98
Bot4	55	89	97	98
Bot5	40	80	97	98
Bot6	27	26	29	28
Bot7	8	79	96	97
Bot8	28	46	62	69
Bot9	40	70	80	84
Bot10	20	37	46	53
Bot11	11	33	/	/
Bot12	/	/	/	/
Bot13	65	92	98	99
Bot14	54	91	97	99
<u>Mean</u>	38	67	81	84

Table 7.3: Win rate of basic Q-learning agent vs bots

Some insight can be outputted from the analysis of the table. First of all, for a low number of games, here 20, the agent nearly never manage to have a positive win rate. From 100 games, the agent has a majority of wins and after 500 only potentially struggles against Bot6 and Bot10. It can also be noted that the result for Bot12 and Bot11 for 500 and 1000 games aren't available because of the tie rate (more than 50%). To really visualize whats going on, the figure 7.2 shows the duel between the Q-learning agent and Bot11. The structure of this kind of graph is always as follow :

- the x-axis is the number of games
- the y-axis contains 3 important information which summarized the simulations (the "-r" argument) :

1. The maximum (orange)/minimum (blue)/mean (green) of the win rates (in percent) for the first player.
2. The maximum (purple)/minimum (brown)/mean (pink) of the win rates (in percent) for the second player.
3. The mean of the tie rates in red (in percent)
4. The mean of the game lengths in blue

As a reminder, the win rate ignores ties (see formula section 4.3).

In this case, it is useful to see the evolution of the tie rate. Another observation is the number of games needed to reach 50% of win rate (more than 200).

To produce the result in bold, a little trick had to be used. Indeed, without any random, agents sometimes fall into an infinite "shield-shield" loop ; at each round, both plays the action shield and so the game never ends and can't produce any results. Thus, introduce a chance of doing a random action when the number of consecutive "shield-shield" round becomes (very) huge seems to be useful. The "shield-shield" loop can be observed, for example, on figure 7.3 at the beginning of each simulation. It doesn't influence much the results but avoids infinite loop.

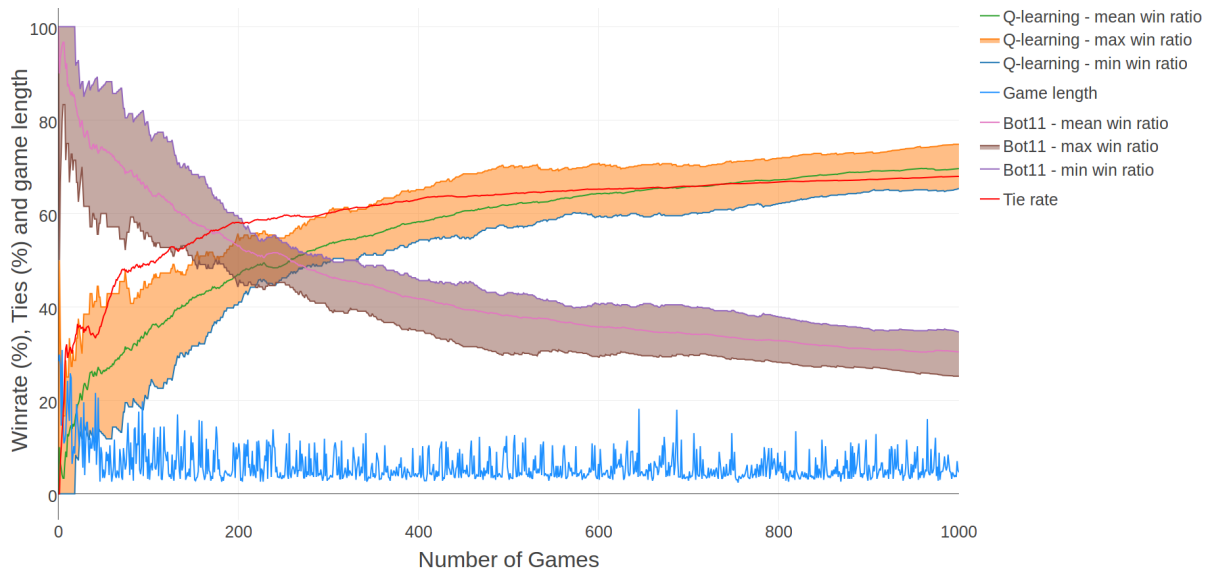


Figure 7.2: Q-learning vs Bot11

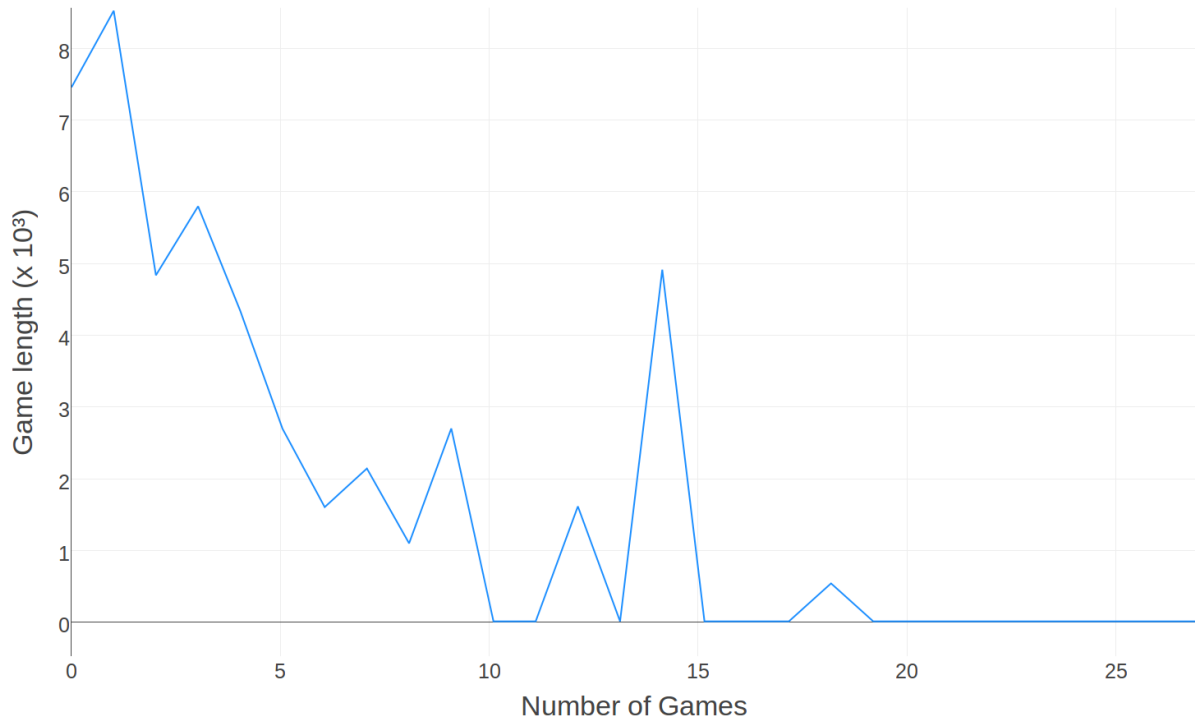


Figure 7.3: Game length of Q-learning vs Bot1

7.3 Drawbacks

The most important drawback is the poor performance of this agent. This algorithm is in fact already bothered by these kind of easy bots. Indeed a human player that would win on average at 65 % after 20 games (without changing his strategy) will most likely stop playing. The table shows that this agent needs sometimes more than 100 games to obtain 50 % of win rate. It isn't realistic to assume a human always plays more than 100 games.

Moreover many bots have an optimal counter if it doesn't play totally randomly (like Bot8-11). Therefore it should be easy to obtain greater scores. The trouble here is that the agent doesn't approximate the probability of each action for the opponent according to a state. It only tries to find the best move at each state but doesn't consider the strategy of the opponent. That's why an enhancement of this algorithm is introduced next chapter.

Chapter 8

Opponent Modeling

As stated before, the framework figure 7.1 can be extrapolate for several agents and consequently Q-learning is a good basis because the transition probabilities is unknown. The only difference and also the issue is, as defined by the report [65] and raised in the previous chapter, that each agent can have an effect on the environment and so an action can have an outcome depending on the actions of the other agents. Consequently, with multiple agents, the environment isn't stationary anymore but the reinforcement learning technique shown before (Q-learning) assumes this fact. To counteract this problem, the RL algorithm needs to approximate the policy of the enemies. That's why a well known adaptation of Q-learning called Opponent Modeling is introduced in this chapter.

Each section follows the usual structure. The first one compare the two algorithms. In addition, the second section also describes the (dis)advantages of different representation of a state S and assess the performance of each of them to determine the best one for the rest of this thesis.

8.1 From Q-learning to Opponent modeling

As said in the introduction, the purpose of opponent modeling is to make an explicit model of the current opponent player and more precisely its behavior. This type of algorithm is called "joint action learner" because it takes the action of each player into account. It's proven that the theoretical optimal solution isn't always the best solution in practice. Indeed, if an agent can exploit the weaknesses of the opponent, he can become stronger and obtain an highest reward. In other words, the optimal policy depends on the behavior of the opponent. This is the purpose of this model. A basic version is shown at Algorithm 4 [66]. The difference between Q-learning and Opponent modeling is the introduction of a new function " $N(s, a)$ " which, in fact, returns the number of times the opponent selects the action a at state s . This function can be used to predict the probability of each opponent action at each state :

$$p(s, a) = \frac{N(s, a)}{N(s)}$$

$N(s)$ is the number of times the opponent had to make a choice at state s . Consequently, Q values are now weighted by p .

Algorithm 4: Opponent modeling

```
1  $S_0 \leftarrow$  initial state of the game
2  $A \leftarrow$  set of  $\langle action, action \rangle$  tuple
3  $Q \leftarrow$  Q learned from the previous games
4  $N \leftarrow$  N learned from the previous games.
5  $\alpha \leftarrow$  the learning rate ( $0 \leq \alpha \leq 1$ )
6  $\gamma \leftarrow$  the discount factor ( $0 \leq \gamma < 1$ )
7 foreach game do
8   Initialize  $S=S_0$ 
9   foreach round i do
10     if State  $S_i$  not already explored then
11       Initialize  $Q(S_i, A) = 0$ 
12       Initialize  $N(S_i, A) = 0$ 
13       Initialize  $N(S_i) = 0$ 
14      $B_i \leftarrow$  subset of playable actions in A at round i
15      $C_i \leftarrow$  subset of playable actions in A at round i for the opponent
16
17     Take playable decision  $a \in B$  maximising  $O(S_i, B)$  with
18      $O(S_i, a) \leftarrow \sum_{c \in C_i} \frac{N(S_i, c)}{N(S_i)} Q(S_i, \langle a, c \rangle)$ 
19
20     Observe reward R, next state  $S_{i+1}$  and decision takes by the opponent  $opp_i$ 
21
22      $V = \max_{b \in B_{i+1}} \sum_{c \in C_{i+1}} \frac{N(S_{i+1}, c)}{N(S_{i+1})} Q(S_{i+1}, \langle b, c \rangle)$ 
23
24      $Q(S_i, \langle a, opp_i \rangle) \leftarrow (1-\alpha)Q(S_i, \langle a, opp_i \rangle) + \alpha(R + \gamma V)$ 
25
26      $N(S_i, opp_i) ++$ 
27      $N(S_i) ++$ 
```

8.2 Experiments

Unfortunately, as shown on the pseudo-algorithm, this model isn't an out of the box solution. It is first needed, as for Q-learning, to define a representation of a state specifically for our game. Consequently, this section is divided in 2 parts. The first compare multiple representation of a state S and the second one is used to compare the basic Q-learning technique (chapter 7) with this new algorithm.

8.2.1 Representation of a state

Two possible solutions were explored :

1. Define a state as the set of the previous moves (abbreviation : "p|p")
2. Define a state as a set of status (see section 7.2.2 - abbreviation : "s|s")

The first idea was to define a state as all the decision taken by the two players since the beginning of the game. An easy to understand example is shown below. The table 8.1 is an example of a game between P1 and P2.

Round	Transition	Decision P1	Decision P2
1	$S_0 \rightarrow S_1$	<i>RELOAD</i>	<i>SHIELD</i>
2	$S_1 \rightarrow S_2$	<i>SHOOT₁</i>	<i>SHIELD</i>
3	$S_2 \rightarrow S_3$	<i>RELOAD</i>	<i>RELOAD</i>
4	$S_3 \rightarrow S_4$	<i>SHOOT₁</i>	<i>RELOAD</i>
Player 1 wins			

Table 8.1: Decision of each player at each round

Consequently, the representation of each state can be easily derived :

$$S_0 = P1 : \{\} P2 : \{\}$$

$$S_1 = P1 : \{RELOAD\} P2 : \{SHIELD\}$$

$$S_2 = P1 : \{RELOAD, SHOOT_1\} P2 : \{SHIELD, SHIELD\}$$

$$S_3 = P1 : \{RELOAD, SHOOT_1, RELOAD\} P2 : \{SHIELD, SHIELD, RELOAD\}$$

$$S_4 = P1 : \{RELOAD, SHOOT_1, RELOAD, SHOOT_1\} P2 : \{SHIELD, SHIELD, RELOAD, RELOAD\}$$

The trouble with this representation is that the number of possible states is infinite because the rules of the game don't consider a maximum game length. So, the learning will be slow because of the great number of state and the agent will always have unexplored states (states where it doesn't know what to do). Moreover, intuitively, a human never consider what was played, for example, 15 rounds before.

Thus, a solution can be to look only at the K (maximum) previous decision of the players since the beginning of the game. Here are states from the table 8.1 with $K = 2$

$$S_0 = P1 : \{\} P2 : \{\}$$

$$S_1 = P1 : \{RELOAD\} P2 : \{SHIELD\}$$

$$S_2 = P1 : \{RELOAD, SHOOT_1\} P2 : \{SHIELD, SHIELD\}$$

$$S_3 = P1 : \{SHOOT_1, RELOAD\} P2 : \{SHIELD, RELOAD\}$$

$$S_4 = P1 : \{RELOAD, SHOOT_1\} P2 : \{RELOAD, RELOAD\}$$

Now the number of states is finite but problems are that agent loses a lot of information and it's really difficult to fix the value of K (it depends for example of the opponent).

In conclusion, it seems a priori not a good solution at to represent the experience like this but is for the experience to decide. The challenger of this representation is the one introduced in section 7.2.2.

Before tackling to the experiment, there is in fact, another parameter that can be defined in two ways to create more representations, **Q**. **Q** which holds the Q value of actions according to a previous combination of $\langle \text{actions}, \text{actions} \rangle$ can be modified. Two version were created. One where both actions from the agent and his opponent (abbreviation : "36" because 6 actions for each players and so 6×6 Q values) are hold and another that only keeps the last move of the agent (abbreviation : "6"). Even if it seems strange prima facie, the outcome is interesting.

Now, to settle which one is the most appropriate, the easiest idea was to make a simulation between each representation but it quickly appeared that it wasn't possible. Indeed, the agents only play more or less 20 games before agreeing to a repeating game finishing in a tie. It seems logic because of the lack of randomness and the fact that the immediate reward of a tie is null (it isn't a bad or a good thing). Consequently, the sample of meaningful games isn't significant at all. So the idea is to simply confront the Algorithm 4 (opponent modeling) with different state against bots. The previous discussion infers 4 different version, but since the `p|p` type implies a pattern length it has been decoupled in two sizes 5 or 99. These values were chosen since 5 (abbreviation : `pl5`) seemed to give good results and increasing even more than 99 (abbreviation : `pl99`) the length isn't beneficial anymore (and can be considered as a full history of each game).

In summary, given all these abbreviations and structures, 6 agents will be compared. They all use the Opponent Modeling algorithm. To be clear and explicit, `p|p-36-pl99` is the Opponent Modeling agent considering both actions and using the first representation with a pattern length limit of 99.

The results of the confrontation is resumed in Table 8.2. Each cell contains the average win ratio of the opponent modeling technique after 50, 100 and 500 games repeated 5 times with, again, α and $\gamma = 0.5$. The last row contains the mean of the obtained results for each agent.

	s s-6	s s-36	p p-6-p15	p p-6-p199	p p-36-p15	p p-36-p199
Bot1	99/99/99	96/99/99	98/99/99	98/99/99	95/98/99	98/99/99
Bot2	74/82/94	73/81/94	87/93/98	74/84/96	87/92/98	88/94/99
Bot3	97/98/99	98/99/99	98/99/99	99/99/99	98/98/99	95/96/99
Bot4	65/70/86	69/74/89	64/72/89	60/61/69	61/68/88	52/58/69
Bot5	90/95/99	84/90/98	84/92/98	89/95/99	88/94/99	83/90/98
Bot6	42/50/61	45/60/57	61/49/41	27/30/32	49/38/40	61/48/44
Bot7	/	/	/	/	/	/
Bot8	65/67/78	62/70/79	64/70/77	67/73/80	68/69/76	70/73/78
Bot9	82/86/90	86/89/93	89/90/91	88/90/92	88/91/92	90/90/90
Bot10	54/57/66	56/57/62	69/63/67	59/60/68	59/56/64	60/62/66
Bot11	49/55/73	70/80/86	54/64/78	57/66/77	64/64/78	60/65/76
Bot12	96/98/99	97/99/99	97/99/99	93/97/99	96/98/99	94/97/99
Bot13	92/96/99	93/97/99	96/98/99	94/97/99	91/96/99	93/97/99
Bot14	98/99/99	94/97/99	59/73/91	37/43/64	66/76/91	53/59/70
<u>Mean</u>	77/81/88	78/84/89	78/82/87	72/76/83	78/80/86	77/79/84

Table 8.2: Opponent modeling vs bots for different representation of state

The results against **bot7** are discarded since the tie rate reach $\pm 99\%$ and bold results are produced with the trick introduced at section 7.2. Based on this table (by looking vertically or with the means), it seems that column 4 and 6 perform somewhat less than the others and column 1 and 2 appears to be slightly better. But by conducting a paired Student T-test between each state variation, it doesn't reach a significant level to justify a difference. Now if by comparing the results with some more insight on what the bots do some results can still be outputted. Indeed, the worst results (and biggest differences) of these agents are against **Bot14**

which is the only bot with a basic meaningful strategy according to the rules and against Bot6 which is the bot who plays with a counter-play idea. This shows somehow that our bots don't have sufficient complex strategies to settle the best agents.

Therefore to differentiate the representation, it is needed to extend the list of bots by introducing new ones with more interesting strategies. These strategies are still static but follow a logic a real player could want to follow. It means a strategy we judge more efficient and with less repeating pattern. These new bots are described in List 8.1.

- Bot15 : **reload**, n random $\times$ **shield**, **reload**, m random $\times$ **shield**, **reload**, r random $\times$ **shield**, **shoot** 3, t random $\times$ **shield**, repeat.
- Bot16 : **reload** or random **shoot** when the power of the enemy is 0 else **shield**.
- Bot17 : Like Bot16 but can also sometimes **shoot** or **reload** when the power of the enemy isn't 0.
- Bot18 : When the power of the enemy is 0, **reload** until 10 then **shoot** at 10. Otherwise, **shield** a lot (at least 6 times if Shield is available) and sometimes **reload** or **shoot**.
- Bot19 : Like Bot18 but tries only to surpass the Shield of the opponent and not obtain a power of 10.

List 8.1: Advanced bot description

Table 8.3 was created by making simulations against these "advanced" bots to again compare the representation. Same parameters are used and same values (50/100/500 games) are reported than for Table 8.2.

	s s-6	s s-36	p p-6-p15	p p-6-p199	p p-36-p15	p p-36-p199
Bot15	86/92/97	87/92/97	67/65/84	71/71/82	67/67/83	65/67/80
Bot16	93/97/99	95/98/99	92/96/99	92/96/99	93/96/99	91/96/99
Bot17	80/88/95	84/91/97	41/41/58	29/34/41	46/53/63	35/37/53
Bot18	93/97/99	95/97/99	98/99/99	99/99/99	99/99/99	94/97/99
Bot19	90/95/99	90/95/99	86/87/89	90/95/99	93/97/99	93/97/99
<u>Mean</u>	88/94/98	90/95/98	77/78/86	76/79/84	80/82/89	76/79/86

Table 8.3: Opponent modeling vs advanced bots for different representation of state

From the mean value and with the observation made before, it appears that the state representation outperform the pattern representation. To be sure of this proposition, a paired Student T-test was applied between s|s-36 (which seems to have the best results) and the others versions on the 18 bots win rate. The result are stored in Table 8.4. After 50 games, the test isn't significant but from 100+ games, the null hypothesis (that the results are equivalent) can be rejected. This can be stated for all the pattern version of the Algorithm with a significance level of 0.1 till 0.05 for some results.

Therefore we fixed that the "Status" representation is better. This could have been predicted since, only by comparing the two choices, the selected one simply holds more crucial game information than only knowing a game pattern.

	s s-36 @50	s s-36 @100	s s-36 @500
s s-6	0.2411	0.4015	0.1231
p p-6-p15	0.3065	0.0563	0.0811
p p-6-p199	0.0742	0.0423	0.0445
p p-36-p15	0.2419	0.0602	0.0465
p p-36-p199	0.1929	0.0331	0.0586

Table 8.4: Paired Student's t-test

Because of these experiments and the theoretical consideration, the final representation we choose for all the following agents is s|s-36.

8.2.2 Performances of Opponent Modeling

To have an easy understanding of the improvements, Table 8.5 references the same data as for Q-learning (Table 7.3) to compare the 2 algorithms.

	Number of games			
	20	100	500	1000
Bot1	97	99	99	99
Bot2	61	84	96	98
Bot3	98	99	99	99
Bot4	40	74	89	90
Bot5	80	90	98	99
Bot6	53	53	54	54
Bot7	95	99	99	99
Bot8	55	69	80	84
Bot9	74	87	91	93
Bot10	40	57	63	66
Bot11	68	80	86	89
Bot12	93	99	99	99
Bot13	81	97	99	99
Bot14	83	98	99	99
<u>Mean</u>	73	85	89	91

Table 8.5: Win rate of a basic Opponent Modeling agent (s|s-36) vs bots

From this table, it appears that Opponent Modeling is really an improvement in this case. This agent outperforms all the bots and the Q-learning algorithm only beats this new agent against Bot4. The opponent modeling agent wins more than a half of the bots with more or less 100% (it loses only at the beginning when he searches for a strategy). The remaining bots are those with a lot of randomness (Bot8-11) and Bot6 which is, as stated before, a bit different (contains learning). The comparison between figure 8.1 and figure 7.2 shows a better tie rate (60% vs 40%) and a quicker convergence for this new agent. Indeed, Q-learning needs more than 200 games to outperform Bot11 and opponent modeling less than 20. The game length is also a bit smaller. These observations can be done against a lot of bots.

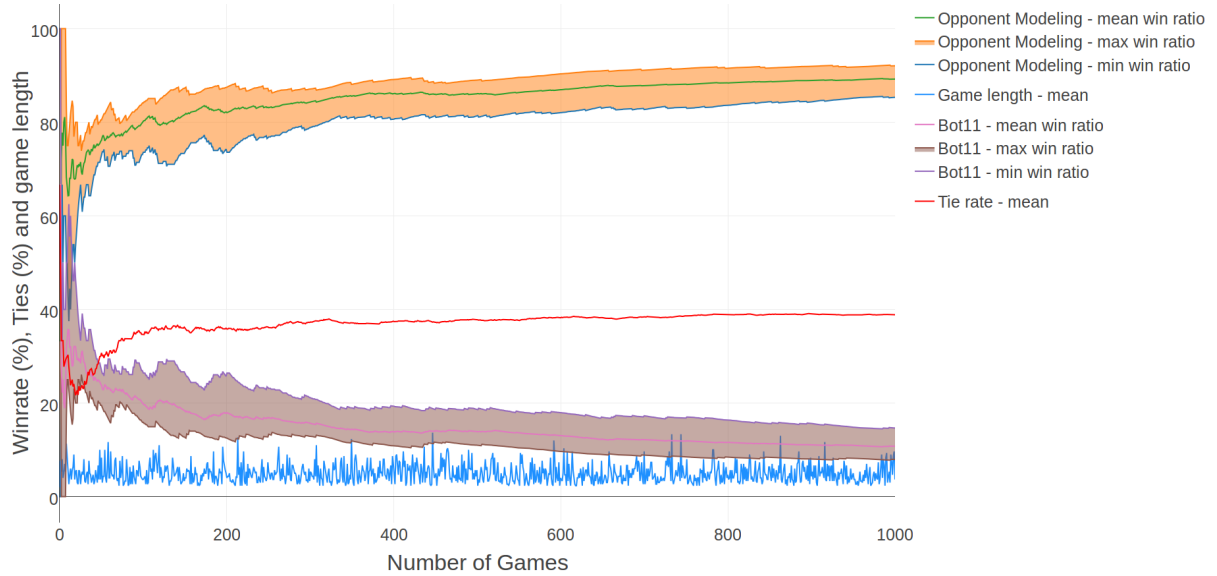


Figure 8.1: Opponent Modeling vs bot11

It is important to point out that the experiment for the representation was made on this algorithm because Q-learning wasn't promising enough even with the best representation.

8.3 Drawbacks

The first problem which appears by analyzing this algorithm is the default learning policy. Indeed, the algorithm always chooses the action associated with the greater O . This is called a "greedy" policy. The drawback of this choice is that you haven't got any exploration when a state is already visited and so the agent can fall into a local optimum. Moreover, it is easy to understand that the basic opponent modeling agent will become predictable for a human because he will almost always play the same action for a given state. A human being the final opponent, this algorithm needs to be modified to be less predictable.

Chapter 9

Exploration vs Exploitation dilemma

The previous chapter raises an important issue about the basic Opponent Modeling algorithm and more precisely the **s|s-36** agent. Indeed, the agent needs more exploration to avoid being predictable. Without it, it can't compete with human. This is the purpose of this chapter. The first section talks about some well known techniques to introduce exploration into the previously found agent (**s|s-36**). The remaining of this chapter compares the different techniques to diminish the predictability of **s|s-36**, to find a better solution and then discuss about the drawbacks of this new less predictable agent.

9.1 Introduce exploration by changing the learning policy

For now, exploration is only available when the state is unknown. The most common way to introduce it is to change the learning policy (LP). The learning policy is the way the agent chooses its next action according to its knowledge. To find a good learning policy (LP), the first step is to analyze the most famous ones. It exists a great number of LP but the 3 most commons are described below [67].

9.1.1 Greedy - "s|s-36"

The default LP of the opponent modeling algorithm. The action a is chosen by taking the action with the maximum associated O values for a given state. It means :

$$a = \arg \max_a O(s, a)$$

The opponent modeling agent following this LP is the previously created agent called **s|s-36**

9.1.2 ϵ -greedy

This LP uses a parameter ϵ ($0 < \epsilon < 1$) which is the probability the agent doesn't take the action associated with the greatest O . So, this policy can be written like this if $p(a|s)$ is the probability of taking available action a for a given state s :

$$\forall a, p(a|s) = \begin{cases} 1 - \epsilon, & \text{if } a = \arg \max_a O(s, a) \\ \frac{\epsilon}{\text{Number of available actions} - 1}, & \text{otherwise} \end{cases}$$

9.1.3 Boltzmann distribution - B

This LP uses also a parameter T strictly greater than 0 called the temperature. The probability $p(a|s)$ can be computed like this :

$$\forall a, p(a|s) = \frac{e^{O(s,a)/T}}{\sum_{a' \in \text{available action}} e^{O(s,a')/T}}$$

By analyzing this function, it appears that a greater T means more exploration. This policy seems to be a refinement of the previous one because ϵ -greedy uses only the O values to attribute an higher probability to the best action. The other probabilities aren't dependent of these values. On the other side, Boltzmann uses the O values to compute the probability of each action. Therefore, we didn't make any experiment on the ϵ -greedy policy and prefer be concentrated on the Boltzmann distribution. In the next section, the agent using this LP is denoted "B".

9.2 Experiments

The purpose of this section is to find a compromise for the exploration and exploitation dilemma. As said before, the initial agent **s|s-36** doesn't have any exploration except the first time it is in a state, meaning randomly choose between the action with equal probability. It will be proven that it isn't sufficient against an agent which tries to predict its behavior (more human-like) and so an alternative using the Boltzmann equation to add exploration during the game will be compared with **s|s-36**.

Another way to add exploration is to only add randomness to the initial Q values at the initialization of a state. It means the first time the agent discovers this state. This is different from the actual version, since it will influence the decision process of all action of a given state for multiple iteration and on multiple action by ranking them. Consequently, in the first subsection (9.2.1), 2 methods of exploration will be compared with the initial agent.

Then because of the results of the first experiments, some variations of the previous techniques will be also assessed. This testing phase is the subject of subsection 9.2.2.

9.2.1 First experiment

For the first tables, the goal is to compare the initial agent without any random (**s|s-36**), with an agent containing small (close to 0) /big random initialization (between 0 and 10 - ten being the reward of a win) (**GSR/GBR**) of Q or following the Boltzmann equation (**B**) with different static values for the parameters T .

Firstly, this comparison can be based on the 19 bots defined before (14 described in list 7.1 and 5 described in list 8.1). For the value of T , the simulation were made for $T = 0.1/0.5/1/5$ (the resulting agents are **B-T0.1**, **B-T0.5**,...). The table 9.1 gathers the results obtained after 5 simulation of 500 games against each bot with $\alpha, \gamma = 0.5$ for each agent. Considering the fact the means between 500 and 1000 games are similar to Table 8.5, it is useless to make the simulation on more than 500 games because it stays mostly constant. Only the simulation with less than 50% of ties are reported. The mean is computed without considering **Bot7** cause the value isn't available for all agents. Finally, cells (without considering means) in red are when the value is under 45% after 50 games and in orange when between 45% and 50%.

	s s-36	GBR	GSR	B-T0.1	B-T0.5	B-T1	B-T5
Bot1	96/99/99	99/99/99	98/99/99	99/99/99	99/99/99	99/99/99	93/95/98
Bot2	73/81/94	67/78/92	72/83/93	73/79/93	57/69/91	61/69/83	37/41/51
Bot3	98/99/99	99/99/99	99/99/99	99/99/99	99/99/99	99/99/99	99/99/99
Bot4	69/74/89	63/77/95	82/92/99	78/90/98	65/80/97	69/77/95	41/54/63
Bot5	84/90/98	81/90/98	86/93/99	87/93/99	88/94/97	77/85/95	65/67/74
Bot6	45/60/57	37/41/48	57/55/51	61/57/52	45/46/56	58/58/64	56/60/60
Bot7	/	58/69/84	70/75/76	/	72/72/77	60/62/74	38/41/47
Bot8	62/70/79	68/77/85	72/77/85	75/76/84	70/79/84	73/75/82	56/59/62
Bot9	86/89/93	90/90/93	89/91/94	82/85/91	90/93/95	84/86/93	74/77/81
Bot10	56/57/62	73/76/78	72/72/75	64/67/77	66/66/75	60/65/73	59/60/64
Bot11	70/80/86	57/69/85	72/76/87	65/74/86	65/76/89	64/69/83	43/47/56
Bot12	97/99/99	98/99/99	96/98/99	94/98/99	95/97/99	89/94/99	47/59/74
Bot13	93/97/99	91/95/99	98/99/99	99/99/99	89/94/99	97/98/99	65/68/77
Bot14	94/97/99	54/77/96	84/92/99	70/85/97	55/75/94	39/52/74	23/23/31
Bot15	87/92/97	77/83/94	83/86/95	82/85/93	82/86/94	81/86/92	70/74/79
Bot16	95/98/99	59/79/96	94/97/99	94/97/99	72/85/95	44/56/83	20/19/30
Bot17	84/91/97	23/27/52	47/52/54	45/52/55	33/39/52	20/24/39	19/15/17
Bot18	95/97/99	95/98/99	99/99/99	99/99/99	98/99/99	98/98/99	88/89/95
Bot19	90/95/99	64/82/97	91/95/99	92/97/99	70/85/95	43/59/83	29/29/29
<u>Mean</u>	82/87/92	72/80/89	83/87/91	81/85/90	75/81/89	70/75/86	57/57/63

Table 9.1: Opponent modeling vs basic bots for different type of exploration

Concerning Table 9.1, the first important interpretation is that the best results are agents with a small part of exploration (best in green, second in orange and third in red). Consequently, it naturally appears that increasing T decreases the performances since T means more exploration (see 9.1.3). In fact, it seems logic because most of the bots have a fixed strategy and so doesn't require a lot of exploration.

Because of the results, it isn't interesting to consider T greater than 5. Indeed, $T = 5$ degrades a lot the performances with only 63% of win rate after 500 games. Moreover, they have 4 or more red cells each.

Then, as shown on figure 9.1a the max/min of the win rate of **GBR** at the beginning of a simulation (less than 100 games) is too huge to consider it as reliable. This graph compared to 9.1 also shows that **GBR** eventually finds the optimal solution but it can take a lot of time (depending on the opponent) due to the initial randomness which can give a good score to a wrong move and inversely. On the other hand, by observing the game length, we can make the assumption that it can quickly be retrieved. In fact, the game length rapidly becomes almost constant (after between 0 and 30 games) and it means the agent found the strategy. In contrast, the game length of **GBR** is smaller than for **s|s-36**. As explained section 4.3, it illustrates the fact that **s|s-36** is in a local optimum.

As said before, the smarter bot is **Bot6**. It adapts its strategy in function of the opponent. Against him, the strength of exploration can be noticed but it isn't sufficient to conclude anything.

Therefore we need to test all these bot against something smarter. Fortunately now, because of the exploration, battle between agents without infinite tie is now possible. To test their reliability the easiest is thus to make them compete against each other.

Results of these showdowns are in table 9.2 and associated graphs are in Section 14.3. For this experiment, 5 simulations are made for each dual with $\alpha, \gamma = 0.5$ on 500 games and the table reports, as before, the win rates of the agent of the column versus the agent of the row after 50/100 and 500 played games when there is less than 60% of ties. 40% of games without ties are considered as sufficient to highlight the power of exploration against smarter agents.

To test them even more we made them also compete against two real humans on 100 games each. Figures 9.2 and 9.3 reference these results.

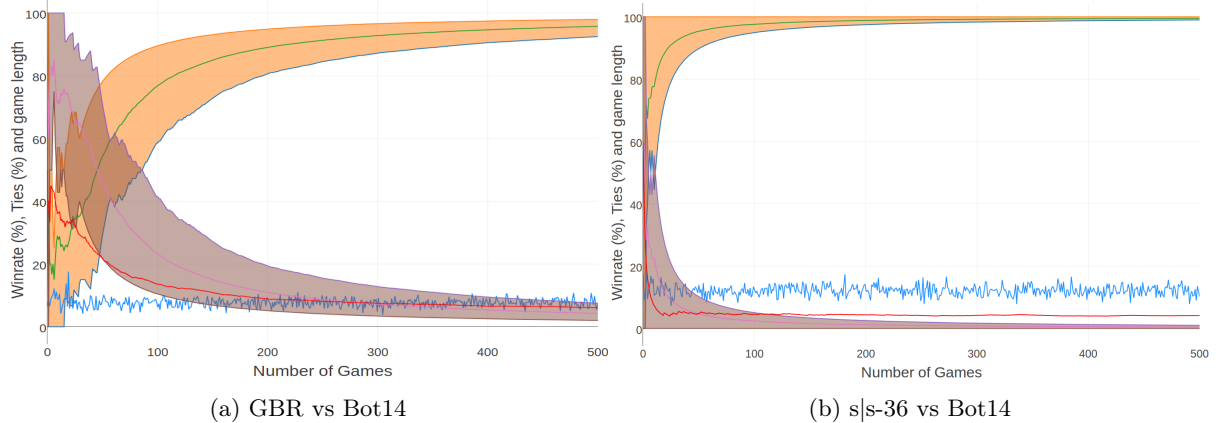


Figure 9.1: s|s-36 and GBR vs Bot14

	s s-36	GBR	GSR	B-T0.1	B-T0.5	B-T1	B-T5
s s-36		61/63/57	/	/	57/59/60	48/53/58	41/42/40
GBR	39/37/42		42/42/38	50/46/42	52/48/47	41/45/44	32/30/30
GSR	/	58/58/62		54/53/54	47/53/53	53/53/54	44/41/37
B-T0.1	/	50/54/58	46/47/46		54/55/53	50/49/50	42/38/30
B-T0.5	43/41/40	48/52/53	53/47/47	46/45/47		45/45/47	34/29/31
B-T1	52/47/42	59/55/56	47/47/46	50/51/50	55/55/53		40/38/33
B-T5	59/58/60	68/70/70	56/59/63	58/62/70	66/71/69	60/62/67	
<u>Mean</u>	48/46/46	57/59/60	49/48/48	52/51/53	55/57/56	50/51/53	39/36/34

Table 9.2: Dual between intelligent agents (see graphs Appendices 14.1)

From table 9.2 and 9.1, B-T1 and B-T5 can definitely be discarded because they aren't in top 3 against bots or smart agents and they have a lot of red cells again.

Concerning s|s-36, it has, as expected, really poor performances against smart agents. The same conclusion can be made for GSR.

Table 9.2 and its top 3 really highlight the fact that exploration is mandatory against adaptive opponents and so against humans. In this case, the best result is obtained by GBR which isn't in the top 3 of the first table. The second best score is achieved by B-T0.5.

Figures 9.2 and 9.3 allow us to definitely discard s|s-36. Its performance is decreasing with time and by analyzing the game length, the predictability can be observed. For example, the game length between 80 and 100 games on figure 9.2a in blue is constant and in favor of the human.

These graphs also prove the poor performances of GBR and GSR and contrariwise the globally good performances obtained with the Boltzmann distribution. Indeed, the mean of the two humans decrease with time against these bots. The top performances are clearly obtained by B-T0.5. The performances of GSR and GBR can be explained because they will eventually converge to s|s-36 and so become predictable. In fact, they have only exploration at the beginning but this exploration is recovered by the update of the Q values.

In conclusion of this first experiment, Boltzmann seems to be the best solution to improve s|s-36. The T value must apparently be between 0 and 1 (0 being also the equivalent of GSR) and T=0.5 has good results in all situation. It is important to remember that a human can also play a fixed strategy if he wants, which means the agent should perform good in every category.

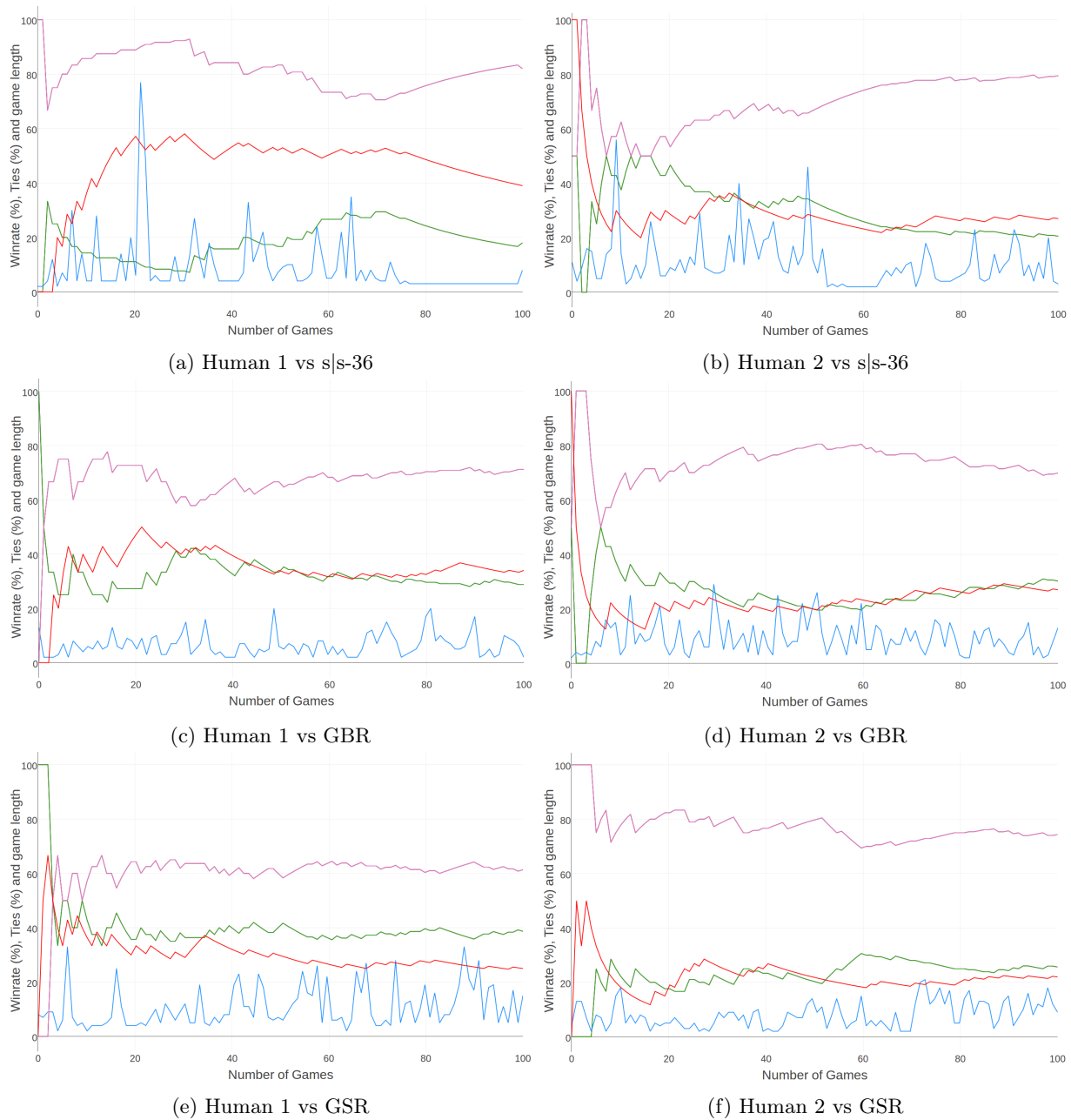


Figure 9.2: Humans vs smart agents

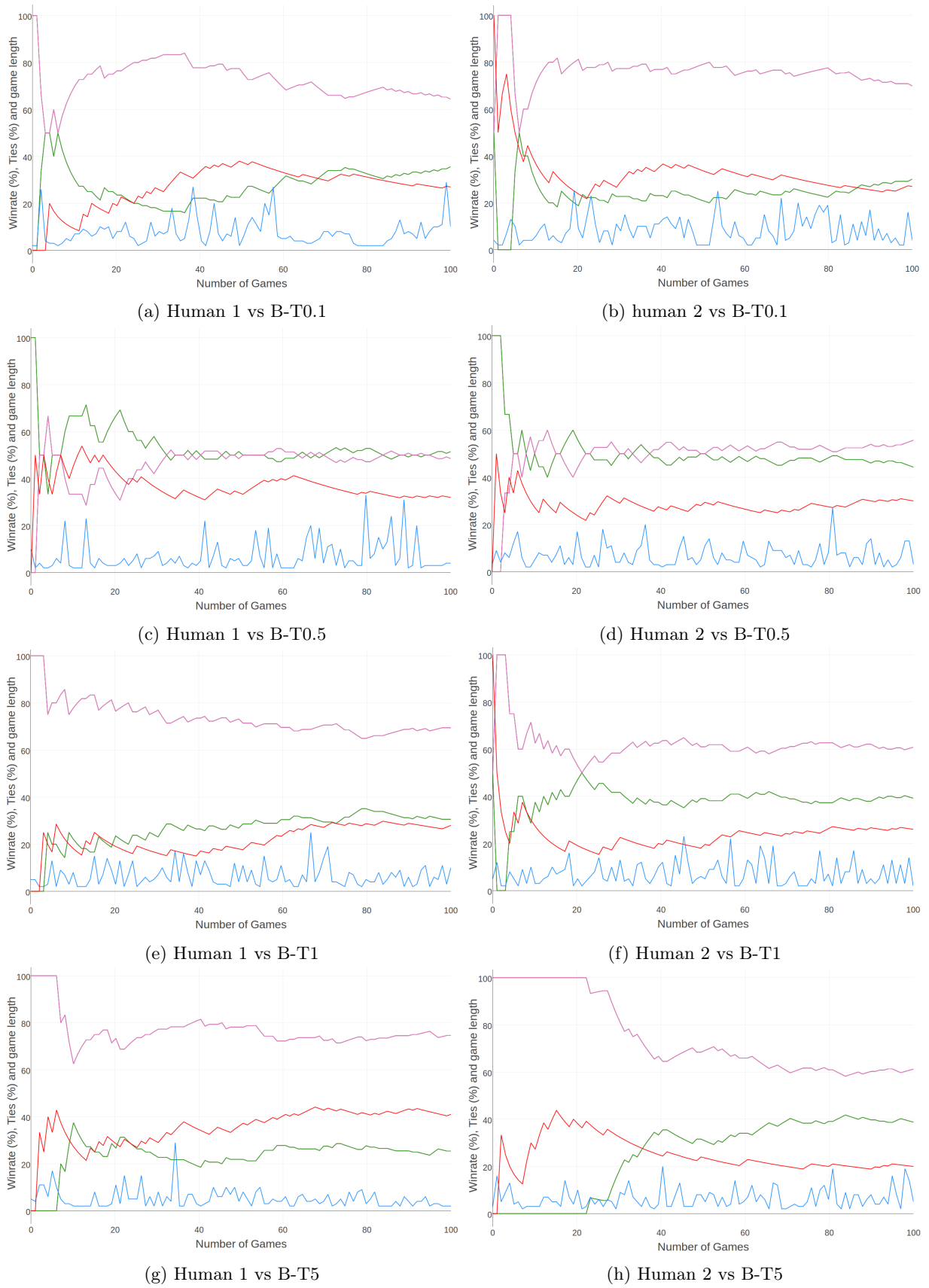


Figure 9.3: Humans vs smart agents

9.2.2 Second experiment

Because of the previous conclusion, the next idea is consequently to use a dynamic T (between 0 and 1) to reach the performances of **GSR** against bots (and so against fixed strategy) and the performance of **B-T0.5** when the opponent is smarter. So the purpose is to find a good trade-off between exploitation and exploration in function of the opponent. The optimal case is a lot of exploration when the strategy changes a lot and few exploration otherwise.

This technique can also be coupled with random initialization (**GBR**) which has good performance against smart agents but poor against bots. So the previous results will be comparing 4 new agents :

- **BR-DT-5 (BR-DT-10)** : an agent with a big random initialization and a dynamic T updates at each multiple of 5 games (10 games).
- **DT-5 (DT-10)** : an agent with a dynamic T updates at each multiple of 5 games (10 games)

Tables 9.3 and 9.4 aggregate the results obtained with the same logic than for experiments of subsection 9.2.1.

	DT-10	DT-5	BR-DT-10	BR-DT-5
Bot1	99/99/99	98/99/99	98/99/99	98/99/99
Bot2	72/81/93	63/71/90	60/74/92	63/77/92
Bot3	99/99/99	99/99/99	99/99/99	99/99/99
Bot4	90/95/98	80/85/95	69/79/95	66/79/96
Bot5	88/94/99	93/96/99	82/90/98	81/89/98
Bot6	61/59/55	53/55/51	33/48/56	57/52/52
Bot7	74/87/93	98/90/93	64/69/82	63/68/82
Bot8	72/77/84	73/78/86	67/74/84	70/75/85
Bot9	94/92/94	89/90/92	85/88/93	89/92/94
Bot10	64/63/70	64/64/69	67/70/76	68/71/77
Bot11	72/76/86	72/79/86	54/68/85	65/73/87
Bot12	93/97/99	94/97/99	86/94/99	92/96/99
Bot13	96/98/99	87/93/99	90/94/99	96/98/99
Bot14	44/73/95	37/64/92	49/77/96	48/76/96
Bot15	84/90/96	86/90/96	84/87/95	76/84/94
Bot16	70/85/97	74/81/96	37/50/89	36/60/93
Bot17	34/38/48	34/36/51	18/19/33	17/23/44
Bot18	97/99/99	98/99/99	96/98/99	96/98/99
Bot19	69/81/95	83/89/97	44/63/92	38/58/90
<u>Mean</u>	78/83/89	78/82/89	68/76/88	69/78/88

Table 9.3: Agents with dynamic Boltzmann temperature vs bots

	DT-10	DT-5	BR-DT-10	BR-DT5
s s-36	/	/	58/59/68	61/62/61
GBR	57/52/39	41/41/36	51/50/51	48/52/51
GSR	49/46/56	47/51/57	52/56/61	53/54/63
B-T0.1	52/50/48	47/50/51	57/56/58	49/52/53
B-T0.5	56/55/47	53/48/42	50/54/54	56/57/53
B-T1	53/48/48	45/48/47	50/54/57	54/54/55
B-T5	64/62/64	63/61/61	67/65/71	64/64/67
<u>Mean</u>	55/52/50	49/50/49	55/56/60	55/56/58

Table 9.4: Dual between agents with dynamic Boltzmann temperature and intelligent agents

From table 9.3, it seems clear that a random initialization reduces the performances against a fixed strategy. These results remain globally good but are clearly not an improvement. They are more or less equivalent to B-T1 which we already discarded. Moreover, they have often less than 50% of win rate after 50 games.

Best results are obtained by DT-10 and DT-5 (78/82.5/89 % of win rate) and are between the results of B-T0.1 (81/85/90) and B-T0.5 (75/81/89).

On the other side, from table 9.4, it appears that the bad ones against bots, BR-DT-10 and BR-DT-5 are the best agents against more intelligent ones. In fact, they are a tiny bit better than B-T0.5 but are worst than GBR. This difference is more negligible than anything else though they manage to beat (in versus) B-T0.5 but not GBR. In these simulation it can also be noted that DT-* have decreasing performances.

In conclusion, it doesn't seems clear if the dynamic T agents are a real enhancement of the other agents since they are a bit better in one case but clearly less in the other case. According to the performances against bots, smart agents and human, the best agent is still on average B-T0.5.

This track although seems to have some promises but maybe the way the T value is updated isn't the right one. This could be an improvement for a future work.

9.3 Drawback

Up to now, agents were only assessed without train, against one agent at a time and without change of the strategy. It means the agent knows nothing before the beginning of a simulation.

In the final application, the agent will have to rapidly adapt its strategy if, for example, the opponent changes and it will also have a train of all the previous simulation. The next 2 chapters will be used to describe the way we used to build an adaptive agent.

Chapter 10

Opponent Modeling with fitting α and γ

In the previous experiments, the α and γ values were considered as constant and fixed at 0.5. Nevertheless, these parameters can be tuned to enhance the performances of an agent. Indeed, changing these values can turn an agent into a more adaptive one. The first next section (10.1) will define theoretically these two important parameters. As usual, the second section (10.2) will make some experiments on the current agent (B-T0.5) with the new adaptation to evaluate it. Finally, the section 10.3 will talk about the remaining negative point of the freshly created agent.

10.1 Standard way to tune α and γ

From chapters 6 and 7, we already know that $0 < \alpha \leq 1$ and $0 \leq \gamma < 1$. Moreover, α must decrease with time (principle of stochastic approximation). It seems logic because it defines the learning rate and obviously agent needs to learn a lot at the beginning but finally converge and then only follow what it has already learned. The trouble in our case is that the opponent can change and so the learning rate must quickly become larger again to converge to another solution (the new optimal one).

On the other hand, remind that γ gives importance to future rewards according to immediate rewards. $\gamma = 1$ gives the same importance to each reward and $\gamma = 0$ gives more importance to immediate rewards. Theoretically, γ should increase with time because future rewards will be more and more reliable.

These considerations are what we want to consider next section.

10.2 Experiments

Two important parts need to be explored concerning α and γ . The first one is the best initial/final α and γ values when the agent has a new opponent and the importance of these values. In particular, there are two main questions :

1. Are these values important for our agent or is him independent of them ?
2. If they are significantly important, how to tune them ? Is the standard way presented section 10.1 the right way in our case ?

This is the purpose of the subsection 10.2.1. The other subsection (10.2.3) concerns the analyze of the transition between 2 opponents to observe if the new agent can adapt the strategy faster than the previous agent (B-T0.5).

10.2.1 Influence of α and γ

To show the possible influence of α and γ , we firstly decided to select 3 smart and really different agents : **GSR** which has good results against bots but some difficulties against the other agents, **B-T0.1** which has still good results against fixed strategies and medium results against smarter agents and **GBR** which contains bad results against bots but best results otherwise. We also selected these bots because **GSR** has almost no exploration while **GBR** has a lot of randomness and **B-T0.1** is a good trade-off.

For each agent, 5 simulations of 500 games were made against B-T0.5 (the agent we choose previous chapter as the best agent) for each value of α between 0.1 and 1 and γ between 0 and 1 by step of 0.1 (it isn't meaningful to make simulations for $\alpha = 0$ because of the way the Qs are updated - it just gives bad results). The results are 3 heat-maps with the mean of performances for each agent for each combination of α , γ after 50, 100 and 500 games. Results are in Appendices 14.4. Moreover, the mean of the results obtained against the 3 agents after 50, 100 and 500 games are also given in figures 10.1a to 10.1c.

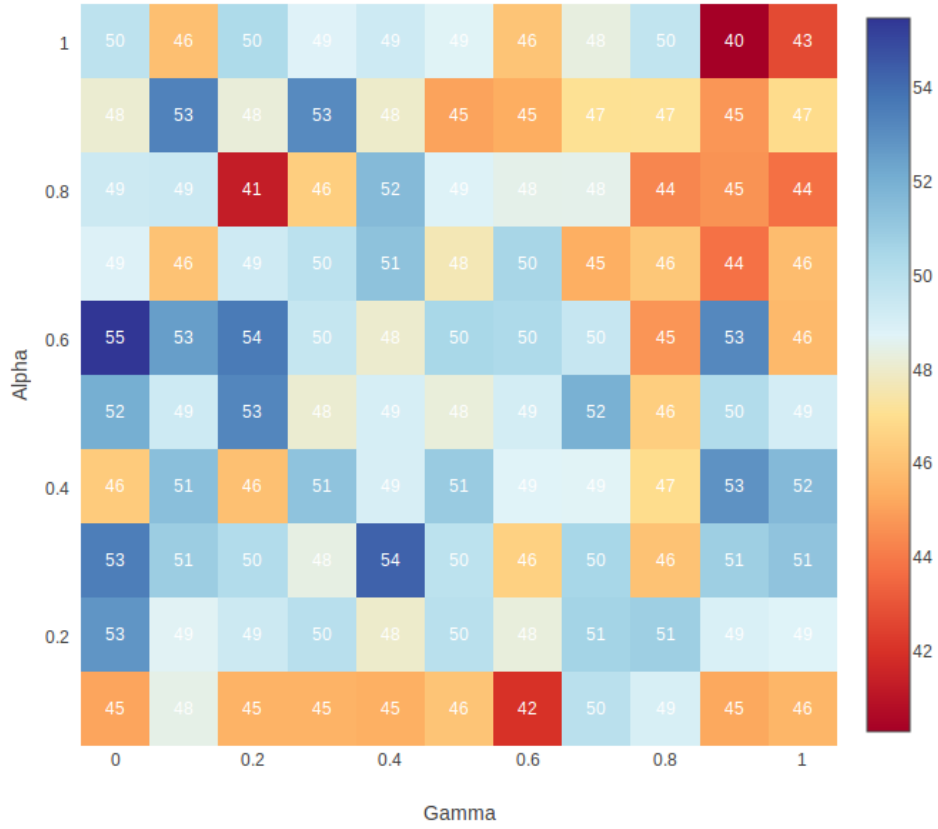
From the first figure (10.1a), the best result is with ($\gamma = 0, \alpha = 0.6$) and the worst with ($\alpha = 1, \gamma = 0.9$). The difference between these two values is 15%. The good results are mainly in the left part of the figure. It matches with the theoretical expected value of γ . On the other hand bad result are achieved for $\alpha = 0.1$ and the upright corner isn't interesting at all. The conclusion is that the initial value for γ must be really low, but nothing can be said about α . Still it can be observed that the value of these parameters are really important and changes the performances at the beginning of a game.

The second figure (10.1b) is more clear. Indeed, the upright corner isn't again a good choice for the parameters at all. The best score are achieved for values lying between $\alpha = 0.5$ and $\alpha = 0.2$. The extreme score for these are 49% and 57% (the second value is 54%). So, taking a low α value seems a good choice at this step. The best value is with ($\alpha = 0.2, \gamma = 0.9$) as expected by the theory, a lower α than before and an increase for γ . The previous best value ($\gamma = 0, \alpha = 0.6$) isn't so good anymore with 49%. A dynamic α, γ seems consequently necessary - the γ value must increase with the number of games and α value must decrease.

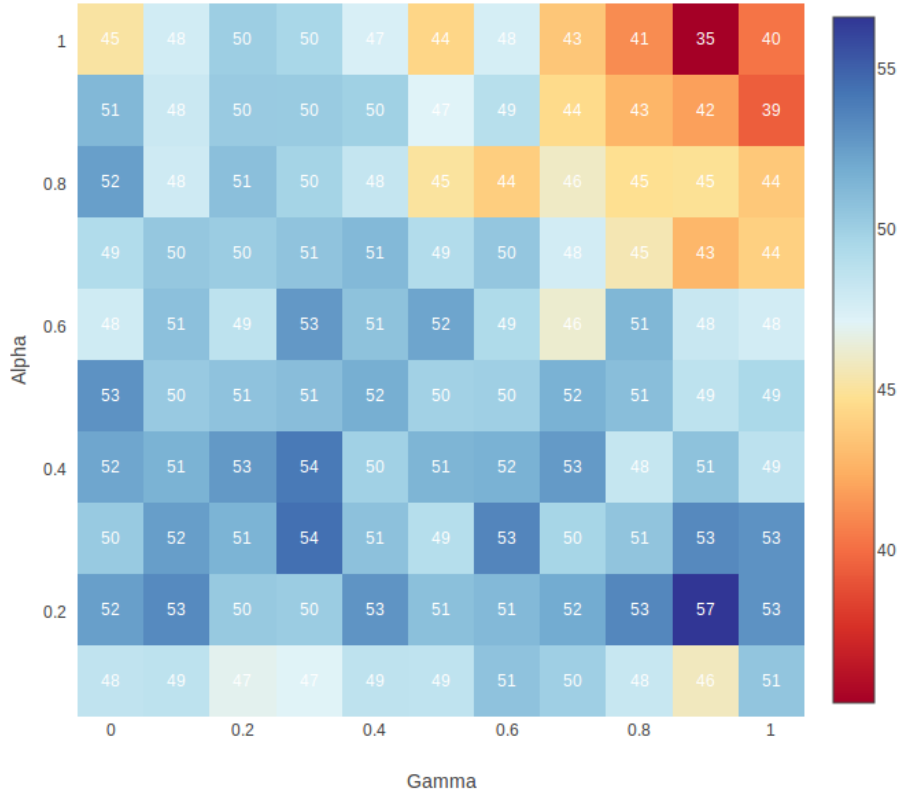
The last figure (10.1c) is only useful to prove the previously made assumptions. The best results are in the interval $0.2 \leq \alpha \leq 0.4$. The influence of γ is negligible but the interval $0.4 \leq \gamma \leq 0.9$ seems a bit better.

10.2.2 A dynamic model with α and γ

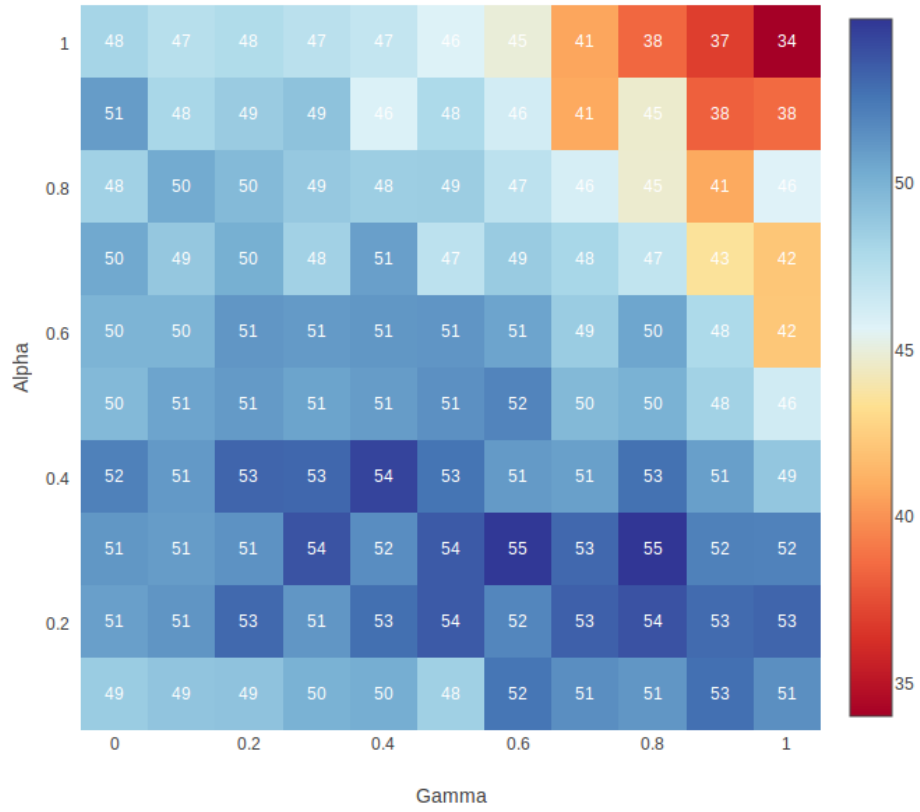
According to these 3 figures and the theoretical aspect, a simple model is to start with an initial value ($\alpha = 0.6, \gamma = 0.1$) and transition to a final value ($\alpha = 0.2, \gamma = 0.9$) obtained after 100 games with a constant step. Values are updated after each game. Even if choosing these value this way seems reasonable, a big influence on the performances doesn't have to be expected. Indeed, the initial choice ($\alpha = 0.5, \gamma = 0.5$) is still more or less



(a) B-T0.5 mean against GSR/B-T0.1/GBR on 50 games



(b) B-T0.5 mean against GSR/B-T0.1/GBR on 100 games



(c) B-T0.5 mean against GSR/B-T0.1/GBR on 500 games

Figure 10.1: Heatmaps for the analyze of the influence of α and γ

adequate although totally arbitrary.

Nevertheless, the tables 10.1 show as usual the win rates (on 5 simulations of 500 games) with this enhanced agent called **B-T0.5-D α γ** against the bots and the smarter agents (with **B-T0.5** removed from the smarter agents set). The values of **B-T0.5** are also reported again to facilitate the comparison.

Notice that here **Bot7** is also used to compute the mean. That's why it isn't the same mean than table 9.1 for **B-T0.5**. Tables 10.1a and 10.1b highlight the fact that the difference between the two agents isn't significant. The new agent is better against bots but not against smarter agents. Anyway, the main objective of this chapter is to have a more adaptive agent. That's why the next experiments using history are useful.

	B-T0.5	B-T0.5-D $\alpha\gamma$
Bot1	99/99/99	97/99/99
Bot2	57/69/91	63/74/92
Bot3	99/99/99	99/99/99
Bot4	65/80/97	72/86/97
Bot5	88/94/97	84/92/99
Bot6	45/46/56	60/59/65
Bot7	72/72/77	54/72/83
Bot8	70/79/84	75/80/85
Bot9	90/93/95	86/89/94
Bot10	66/66/75	67/69/76
Bot11	65/76/89	74/80/88
Bot12	95/97/99	92/96/99
Bot13	89/94/99	90/95/99
Bot14	55/75/94	53/75/96
Bot15	82/86/94	85/87/95
Bot16	72/85/95	62/80/96
Bot17	33/39/52	31/37/54
Bot18	98/99/99	97/99/99
Bot19	70/85/95	72/86/97
<u>Mean</u>	74/80/88	74/82/90

(a) B-T0.5 and B-T0.5-D $\alpha\gamma$ vs bots

	B-T0.5	B-T0.5-D $\alpha\gamma$
s s-36	57/59/60	55/53/58
GBR	52/48/47	48/46/46
GSR	47/53/53	51/54/56
B-T0.1	54/55/53	46/44/51
B-T1	55/55/53	48/49/56
B-T5	66/71/69	57/66/69
<u>Mean</u>	55/57/56	51/52/56

(b) B-T0.5 and B-T0.5-D $\alpha\gamma$ vs smarter agents

Table 10.1: Comparison of B-T0.5 and B-T0.5-D $\alpha\gamma$

10.2.3 Behavior after being trained

This section wants to compare which agent is the most adaptive between **B-T0.5** and **B-T0.5-D $\alpha\gamma$** . To do so, they were firstly trained by observing 100 games of a human. After that, the same dual as before were realized (means on 5 simulation, etc). Results are exposed tables 10.2a and 10.2b.

In this case, the new agent is clearly better than **B-T0.5**. Means are in its favor and it only loses at the beginning against **Bot17** (they both lose against it). The initial **B-T0.5** loses also against **Bot6** and slightly against 3 smarts agents (orange cells). The only trouble of **B-T0.5-D $\alpha\gamma$** is the dual against **GBR** where its performances decrease (but still better than **B-T0.5**). For the final agent, we can assume the presence of a train and so we can consider **B-T0.5-D $\alpha\gamma$** as slightly better than **B-T0.5**.

10.3 Drawback

With this definition of α and γ , we unfortunately introduced an assumption. Indeed, these parameters take their initial values only when the agent is initialized. It means only when a player touches the "play" button of the application. Consequently, the agent doesn't have a algorithm to guess if the player changed. We assumed the fact that, when the player changes, he reloads the application (or at least return to the main page). It can be a future improvement.

Also an important observation to create an adaptive agent is to recall that it stores the behavior of his opponent thanks to an N value (see algorithm 4). The value is only updated by an increment of one when the opponent chooses this action. So, it doesn't take into consideration the fact a strategy/the player can change over time and also some number can become very huge and create a disbalance.

	B-T0.5	B-T0.5-D $\alpha\gamma$
Bot1	98/99/99	99/99/99
Bot2	66/76/91	71/78/93
Bot3	99/99/99	99/99/99
Bot4	64/82/97	92/96/99
Bot5	77/85/97	82/90/98
Bot6	34/38/41	51/48/58
Bot7	59/63/77	55/77/88
Bot8	69/72/81	70/76/85
Bot9	87/90/93	88/90/93
Bot10	65/66/75	70/71/76
Bot11	59/72/87	70/75/89
Bot12	86/93/99	95/98/99
Bot13	92/96/99	96/98/99
Bot14	58/74/92	51/76/96
Bot15	82/84/94	82/86/93
Bot16	62/78/94	54/71/94
Bot17	26/31/47	21/30/51
Bot18	98/99/99	97/99/99
Bot19	61/78/94	54/72/95
<u>Mean</u>	70/78/87	74/81/90

(a) B-T0.5 trained and B-T0.5-D $\alpha\gamma$ trained vs bots

	B-T0.5	B-T0.5-D $\alpha\gamma$
s s-36	52/56/55	59/56/57
GBR	45/42/43	53/47/46
GSR	45/49/49	50/48/52
B-T0.1	49/50/49	54/49/52
B-T1	54/48/52	52/52/52
B-T5	58/61/67	64/65/71
<u>Mean</u>	50/52/55	55/53/55

(b) B-T0.5 trained and B-T0.5-D $\alpha\gamma$ trained vs smarter agents

Table 10.2: Comparison of B-T0.5 and B-T0.5-D $\alpha\gamma$ both previously trained

Chapter 11

Evolved Opponent Modeling with smoothing

As described section 8.1, " $N(s,a)$ " returns the number of times the opponent selects the action a at state s . In algorithm 4, it appears that the only update available is an increment of one when the opponent plays the action. Consequently, a very old game has the same leverage as a recent one. So, old N values may introduce noise because it isn't always the same player and he can change his strategy. To avoid it and consequently have a more adaptive agent, a well-known technique is to smooth these values. To implement it, we have only to set a maximum number of N for each state and so the algorithm smooths if necessary. The result is to give more importance to the last observed actions. The remaining question is : how to set this maximum number ? This question is answered during the unique section 11.1. Notice that, this chapter is the last one that will tweak our agent. Therefore the complete analysis of this model which will be the final one will occur in the next chapter.

11.1 Find a suitable smoothing

Prior to the experiment, the smoothing technique we used after each update of a $N(s,a)$ value is presented on Algorithm 5. $MaxN$ is the chosen maximum number of N for each state and s is the current state.

Algorithm 5: Pseudo-code for smoothing

Data: $MaxN, s$

```
1  $sum \leftarrow 0$ 
2 foreach  $a \in \text{set of actions}$  do
3    $sum += N(s,a)$ 
4 if  $sum > MaxN$  then
5   foreach  $a \in \text{set of actions}$  do
6      $N(s,a) \leftarrow \frac{N(s,a)*maxN}{sum}$ 
```

The methodology for this experiment is a bit different than previously. In fact, the agents aren't compare against bots anymore. The reason is pretty simple : most of the bots (if not all) always play the same action at a given state and consequently, we assumed smoothing won't influence the performances. In return, the mean of the performances against smarter agents is now on 15 simulations instead of 5 to be sure results are reliable. Firstly, 3 different " $MaxN$ "

values were tested on $B-T0.5-D\alpha\gamma$: 10, 20 and 30 ($BD\alpha\gamma S10$, $BD\alpha\gamma S20$ and $BD\alpha\gamma S30$). These values were chosen because less than 10 means that N would become useless and at more than 30 we expect the performances to become similar to $B-T0.5-D\alpha\gamma$ (because it means less smoothing).

For each value, table 11.1 brings together results obtained after 50/100/500 games with a training (as defined in section 10.2.3) and without any training.

	$BD\alpha\gamma S10$	$BD\alpha\gamma S10$ train	$BD\alpha\gamma S20$	$BD\alpha\gamma S20$ train	$BD\alpha\gamma S30$	$BD\alpha\gamma S30$ train
s s-36	58/60/69	58/61/66	57/60/64	55/60/66	58/61/67	50/55/63
GBR	48/51/56	52/50/55	48/49/54	50/46/51	47/49/54	47/45/50
GSR	51/55/65	50/55/63	53/55/65	52/56/60	48/52/61	53/52/58
B-T0.1	49/57/58	49/54/58	54/58/59	45/51/57	52/55/58	51/50/58
B-T1	54/56/55	53/51/56	55/55/56	53/50/55	56/55/56	52/52/55
B-T5	66/66/68	70/69/71	63/65/68	67/68/70	65/65/68	64/64/69
<u>Mean</u>	54/58/62	55/57/62	55/57/61	54/55/60	54/56/61	53/53/59

Table 11.1: Battle of agents with different maximum values for the smoothing against smart agents

The main observation is the improvement of the mean performances compared to the same agent without smoothing ($B-T0.5-D\alpha\gamma$ has 51/52/56 without train and 55/53/55 with a train see Table 10.1b and 10.2b).

By observing the values obtained by the trained agents, it seems that the smallest $MaxN$ is, the better are the performances. In fact, it is logical that it is easier to recover if the $MaxN$ value is low. We can't make any conclusion concerning the values without train since they basically are identical. For the final agent, we consequently choose $maxN = 10$ and so $BD\alpha\gamma S10$.

This choice concludes all the experiments and the adaptations made. The last chapter analyses precisely the performances of the final agent, the one used in the application.

Chapter 12

Final agent

This last chapter is among the crucial ones. Indeed, it is the complete presentation of the agent used in the application and the result of our work. The section 12.1 relates the code of the final algorithm used. By far, the most important part is section 12.2 which challenge the final agent to prove its performance.

12.1 The algorithm

Before all, a little improvement which doesn't need to be tested from the previous agent and which is dictated by game theory (cfr. Section 4.4.1), is to force to reload as first action. Indeed, the agent can't shoot because the initial power is 0 and so doesn't need to shield because the null opponent's power. Without this enhancement, the agent still finds that he must reload at the first step but it takes some times and in case of many loses in short game, it might consider that this first action is not optimal anymore and therefore try to change it.

In summary, the final representation of the algorithm used for our final agent comes from the opponent modeling algorithm and uses the adaptation of the previous chapters. The algorithm 6 is the pseudo-code of this agent.

As a reminder, $O(S_i, a) = \sum_{c \in C_i} \frac{N(S_i, c)}{N(S_i)} Q(S_i, \langle a, c \rangle)$ and R is the function defined at section 7.1.

The exploration discussed chapter 9 is introduced line 21 with the Boltzmann distribution. Then, α and γ (chapter 10) are linearly modified with lines 9 and 10. Smoothing is done according to chapter 11 from line 29 to 33. The last little improvement described above is applied line 19.

12.2 Final performances

The final performances are analyzed in two parts. First, it is useful as usual to show precisely the results against all **Bots** and almost all smart agents previously made (it doesn't consider **BD α γ S*** because they are roughly equivalent). This is the purpose of the subsection 12.2.1.

The final opponent as also aims to be compared to a human. The goal of the second subsection 12.2.2 is to evaluate this match.

For the 2 sections, the assessment are done with and without training.

Algorithm 6: The final model

Data: α, γ

- 1 $S_0 \leftarrow$ initial state of the game
- 2 $A \leftarrow$ set of $\langle action, action \rangle$ tuple
- 3 $Q \leftarrow$ Q learned from the previous games
- 4 $N \leftarrow$ N learned from the previous games.
- 5 $\alpha \leftarrow 0.6$
- 6 $\gamma \leftarrow 0.1$
- 7 **foreach** *game* **do**
- 8 Initialize $S=S_0$
- 9 $\alpha \leftarrow \max(0.2, \alpha - \frac{0.6-0.2}{100})$
- 10 $\gamma \leftarrow \min(0, 9, \gamma + \frac{0.9-0.1}{100})$
- 11 **foreach** *round i* **do**
- 12 **if** *State S_i not already explored* **then**
- 13 Initialize $Q(S_i, A) = 0$
- 14 Initialize $N(S_i, A) = 0$
- 15 $B_i \leftarrow$ subset of playable actions in A at round i
- 16 $C_i \leftarrow$ subset of playable actions in A at round i for the opponent
- 17
- 18 **if** $i==0$ **then**
- 19 Take decision Reload
- 20 **else**
- 21 Take playable decision $a \in B$ according to Boltzmann distribution of $O(S_i, B)$
- 22 with $T=0.5$
- 23 Observe reward R, next state S_{i+1} and decision takes by the opponent opp_i
- 24
- 25 $V \leftarrow \max_{b \in B_{i+1}} \sum_{c \in C_{i+1}} \frac{N(S_{i+1}, c)}{N(S_{i+1})} Q(S_{i+1}, \langle b, c \rangle)$
- 26
- 27 $Q(S_i, \langle a, opp_i \rangle) \leftarrow (1-\alpha)Q(S_i, \langle a, opp_i \rangle) + \alpha(R + \gamma V)$
- 28
- 29 $sum \leftarrow \sum_{a \in C_i} N(S_i, a)$
- 30 $N(S_i, opp_i) ++$
- 31 **if** $sum > 10$ **then**
- 32 **foreach** $a \in C_i$ **do**
- 33 $N(S_i, a) \leftarrow \frac{N(S_i, a) * 10}{sum}$

12.2.1 Performances against agents and bots

Without train

For this experiment, win rates and tie rates are reported Table 12.1 and 12.2 after 50/100/500 games for each battle based on the mean of 20 simulations. As said before, 500 games is more than sufficient and we choose this number of simulation to have a highly reliable result. Tie rates are in brackets. Moreover, all the associated graphs are in appendices (section 14.5 and 14.6). Some of them later are commented in this section.

First of all, let's look at the first 2 columns of Table 12.1. It appears that this agent only loses against **Bot6** and **Bot17**. The defeat against **Bot6** has a minor impact but it means that our agent remains a bit predictable. Indeed, **Bot6** is an agent who tries to find the pattern used by its opponent to counteract the next action. The other defeat is caused by the exploration. Only agent **s|s-36** can win (84/91/97 percent) against this bot because it has a greedy policy and no random initialization. Indeed there merely exist only one strategy to win against this bot and any wrongdoing can be fatal. The positive point is that the agent eventually wins (54% after 500 games).

In general, for each **Bot***, mean win rates increase with the number of games and so it appears clearly to dominate his opponents. Winning more than three-quarters of the time is good. Also the tie rates decrease with the number of game which means that in addition to have more wins, it also statistically win on more games (since ties are not computed in the win score).

From the second table (12.2), the only defeat in the first 2 columns is against **s|s-6** after 100 games but the agent has still 49.77% of win rate so it is more a draw. After 500 games, our agent clearly wins against all agents but **B-T0.5-D α γ** (again draw).

This global win against all previous agents shows that all the improvements realized thorough this entire Part were effective. The final win rate for 500 games reaches 62 % whereas the tie rate remains quite steady around 40 %.

	50 games	100 games	500 games
Bot1	98.52 (0%)	99.25 (0%)	99.85 (0%)
Bot2	68.03 (29%)	76.38 (23%)	92.51 (14%)
Bot3	99.21 (0%)	99.6 (0%)	99.92 (0%)
Bot4	78.91 (20%)	88.37 (14%)	97.87 (5%)
Bot5	90.28 (13%)	94.78 (10%)	98.97 (5%)
Bot6	48.38 (27%)	49.43 (27%)	54.11 (28%)
Bot7	64.84 (36%)	72.15 (31%)	84 (19%)
Bot8	77.45 (26.5%)	80.6 (26%)	86.17 (28%)
Bot9	91.7 (19%)	92.36 (19%)	94.1 (18%)
Bot10	66.95 (24%)	69.75 (23%)	74.67 (24%)
Bot11	73.52 (33%)	77.41 (33%)	88.62 (34%)
Bot12	96.42 (6%)	98.26 (3%)	99.66 (1%)
Bot13	98.32 (31%)	99.14 (32%)	99.82 (33%)
Bot14	52.19 (22%)	77.1 (12%)	95.92 (2%)
Bot15	79.92 (15%)	84.79 (13%)	93.18 (7%)
Bot16	74.04 (12%)	87.17 (6%)	97.54 (13%)
Bot17	31.44 (22%)	40.36 (20%)	54.26 (13%)
Bot18	95.78 (5%)	97.87 (3%)	99.58 (1%)
Bot19	76.49 (10%)	88.07 (6%)	97.71 (1%)
<u>Mean</u>	76.97 (18.45%)	82.78 (15.84%)	89.92 (12.9%)

Table 12.1: Final model performances against bots with static strategies without any train (graphs section 14.5)

	50 games	100 games	500 games
s s-36	58.36 (40%)	61.39 (39.5%)	69 (38%)
s s-6	51.62 (43%)	49.77 (43%)	58.31 (42%)
p p-6-p15	64.04 (48%)	61.25 (40%)	72.2 (33%)
p p-6-p199	61.77 (48%)	62.61 (40%)	71.27 (34%)
p p-36-p15	59.69 (38%)	63.19 (36%)	67.55 (40%)
p p-36-p199	59.19 (46%)	64 (38%)	68.1 (40%)
GBR	53.15 (41%)	53.33 (41%)	56.69 (39%)
GSR	56.74 (40%)	59.29 (38%)	67.1 (35%)
B-T0.1	53.12 (43%)	57.03 (40%)	58.52 (41%)
B-T0.5	52.7 (40%)	54 (40%)	54.39 (44%)
B-T1	53.9 (40%)	52.78 (42%)	56.79 (43%)
B-T5	68.64 (40%)	66.1 (41%)	69.71 (41%)
BR-DT-5	52.2 (41%)	54.68 (41%)	55.95 (42%)
BR-DT-10	51.94 (40.5%)	51.5 (41%)	55.01 (42%)
DT-5	51.96 (40%)	55.75 (39%)	60.87 (40%)
DT-10	53.82 (41%)	55.35 (40%)	61.19 (40%)
B-T0.5-D $\alpha\gamma$	50.35 (42%)	51.97 (42%)	50.29 (43%)
<u>Mean</u>	56.06 (41.85%)	57.29 (40.08%)	61.94 (39.83%)

Table 12.2: Final model performances against smart agents with dynamic strategies without any train

With train

It is now important to compare these performances with those of the same agent trained in the same conditions (same number of simulations,...). As before, the train is made by observing 100 games of a human. Performances are reported tables 12.3 and 12.4 and associated graphs are section 14.6.

From table 12.3, it appears that the trained agent improves the performances in means against **Bots** by 5/2/1 % for 50/100/500 games. There is still one defeat remaining and it's still against **Bot17**.

About table 12.4, the first conclusion is that there aren't any bad results. The lowest is $\pm 49\%$ and be considered as a draw. The important thing is that the values always increase when the number of game grows excepts against **BR-DT-5** and **BR-DT-10** (but results are still in favor of our agent). Unfortunately, the mean win rates are a bit lower than without train. It can be explained by our train not being optimal and with the fact that all these agents really have varying strategies and learning even when they comes from the same original root, **s|s-36**. Still with this "disadvantage" our agent manages to recover nearly to his original score.

	50 games	100 games	500 games
Bot1	99.21 (0%)	99.6 (0%)	99.92 (0%)
Bot2	76.85 (29%)	81.08 (25%)	93.27 (16%)
Bot3	99.8 (0%)	99.9 (0%)	99.98 (0%)
Bot4	87.17 (20%)	94.34 (11%)	98.97 (2%)
Bot5	88.03 (12.5%)	94.21 (6%)	98.9 (1%)
Bot6	61.13 (20%)	60.33 (20%)	66.56 (18%)
Bot7	72.03 (29%)	77.46 (25.5%)	84.96 (16%)
Bot8	76.97 (29%)	79.8 (29%)	85.31 (30%)
Bot9	91.46 (18%)	91.86 (18%)	93.77 (18%)
Bot10	66.72 (26%)	69.83 (25%)	74.5 (24%)
Bot11	76.69 (35%)	80.17 (36%)	89.48 (36%)
Bot12	99.71 (1%)	99.85 (0%)	99.97 (0%)
Bot13	99.29 (33%)	99.6 (33%)	99.92 (32%)
Bot14	70.81 (20%)	84.96 (10%)	95.97 (2%)
Bot15	85.7 (11%)	88.53 (10%)	94.05 (6%)
Bot16	73.42 (11%)	86.64 (7%)	97.42 (2%)
Bot17	29.02 (25.5%)	37.23 (24%)	53.55 (15%)
Bot18	96.62 (4.5%)	98.28 (2%)	99.66 (0%)
Bot19	72.62 (9%)	85.94 (5%)	97.27 (1%)
<u>Mean</u>	80.17 (18%)	84.72 (15%)	90.71 (12%)

Table 12.3: Final model performances against bots with static strategies with train

	50 games	100 games	500 games
s s-36	59.54 (39%)	61.06 (38%)	66.46 (38%)
s s-6	53.45 (37%)	53.62 (41%)	62.4 (37%)
p p-6-p15	59.42 (43%)	60.92 (37%)	71.11 (36%)
p p-6-p199	61.06 (39.5%)	61.25 (37%)	70.45 (34%)
p p-36-p15	58.15 (39%)	60.85 (36%)	67.7 (39%)
p p-36-p199	56.03 (39%)	59.39 (38%)	65.66 (39%)
GBR	49.22 (36%)	50.53 (40%)	56.35 (39%)
GSR	49.85 (37%)	53.79 (37%)	65.1 (37%)
B-T0.1	51.72 (37%)	55.22 (37%)	59.41 (40%)
B-T0.5	49.92 (37%)	50.91 (39%)	53.64 (43%)
B-T1	51.38 (40%)	52.15 (40%)	55.69 (42%)
B-T5	69.64 (35%)	69.64 (38%)	70.87 (41%)
BR-DT-5	51.12 (37%)	48.66 (38%)	53.24 (41%)
BR-DT-10	53.86 (38%)	50.65 (38%)	54.64 (40%)
DT-5	51.3 (38.5%)	52.36 (38%)	60.31 (40%)
DT-10	50.33 (39%)	53.99 (40%)	59.72 (41%)
B-T0.5-D $\alpha\gamma$	48.87 (38%)	50.37 (40%)	52.08 (42%)
<u>Mean</u>	54.4 (38,18%)	55.61 (38,36%)	61.46 (39,35%)

Table 12.4: Final model performances against smart agents with dynamic strategies with train

Comparison

Now, it would be interesting to analyze some showdowns. To do it precisely, graphs of the evolution of these fights are below. For each selected versus, the dual is presented without and with train one next to the other. Figure 12.1 contains some graphs against **Bots** while Figure 12.2 is all about interesting dual against smarter agents. These analysis use the performance metrics criteria supplied in Section 4.3.

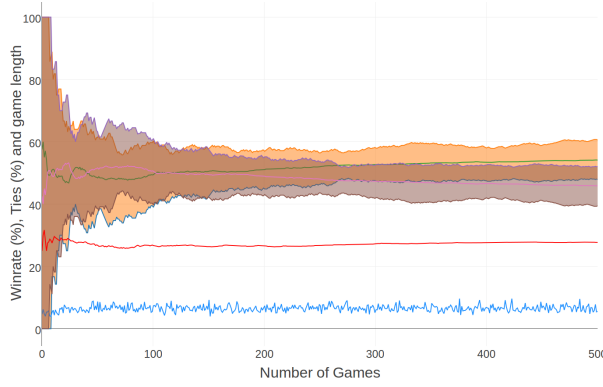
Firstly, Figures 12.1a and 12.1b show clearly that the **Bot6** which rely on pattern struggle much more against the trained agent. Indeed it allows the agent to gain more variation in his opening moves.

From Figures 12.1c and 12.1d it appears according to the game length and the tie rate that the agent quickly finds the best strategy (shield until the opponent shoot then shoot). Indeed, tie rate tends to 0% and win rate to 100%. Game length is less than 10 and so we can assume it doesn't wait the second opponent's shoot and thus the second pattern loop of the Bot. By observing the minimum value at 500 games, it seems that the trained agent has sometimes some difficulties to find the optimal strategy value.

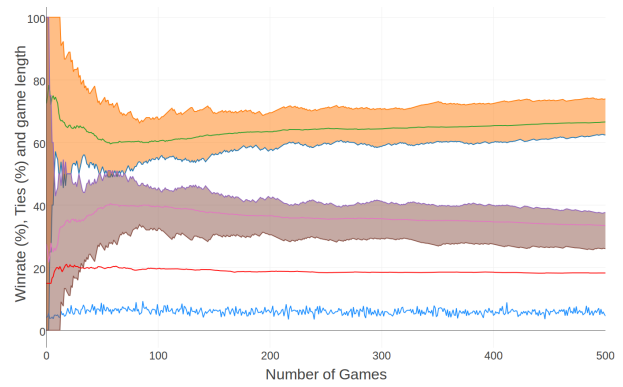
The last thing to observe on graph 12.1d is the huge difference between the minimum and the maximum value of the win rate at the beginning (till 50 games) which shows an high variability due to the training guidance.

The results of **Bot17** are on figures 12.1e and 12.1f. This is the only bot our agent doesn't widely dominate after 500 games. Between 200 and 300 games are needed to recover the very bad start. Here, the game length oscillates a lot at the beginning (0 to 100 games) but gradually increase over time.

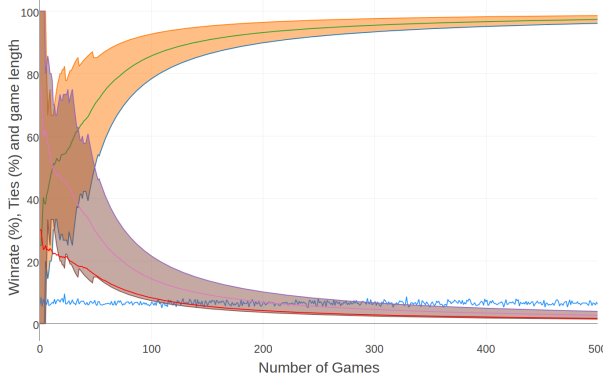
Finally, figures 12.1g and 12.1h are only here to prove in practice the importance of the analysis of the length. At the beginning of the simulations, say before 100 games, the game lengths oscillate a lot and decreases until it become constant. It means that the agent try to find the best solution and when it finds it and is adapted to the opponent, it always play the same thing. It doesn't need to change again because the opponent has a static strategy. This behavior can be observed against a lot of bots. It clearly proves this agent learns the strategy of the opponent to counter it. As expected, note that it quickly finds the good solution.



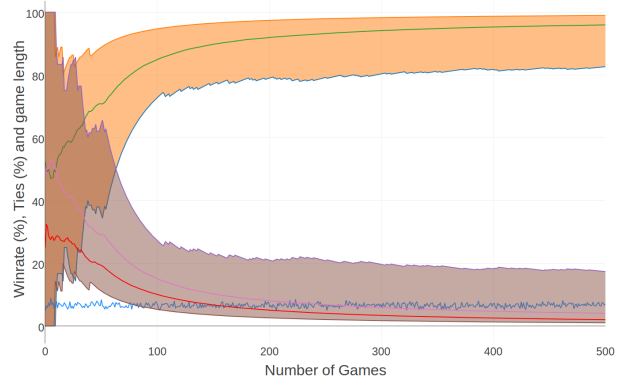
(a) Final agent vs Bot6



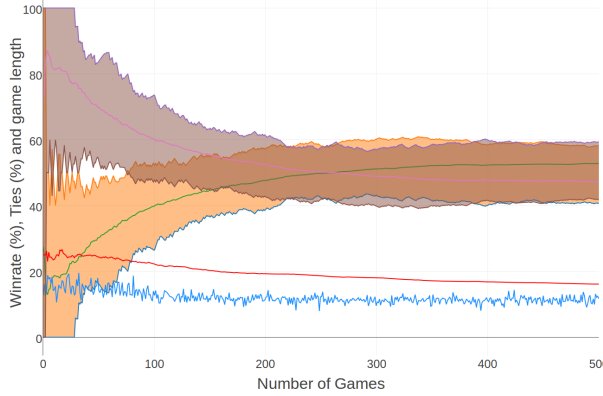
(b) Final agent trained vs Bot6



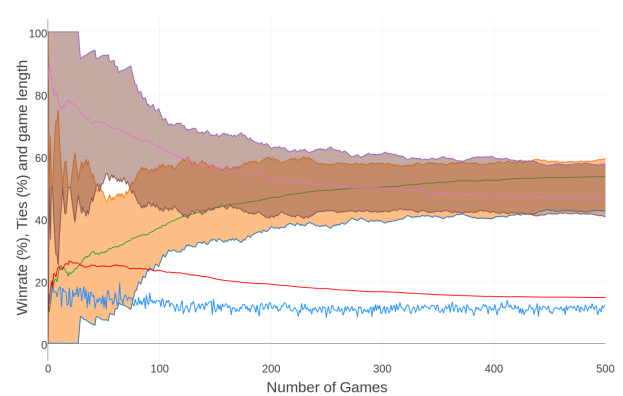
(c) Final agent vs Bot14



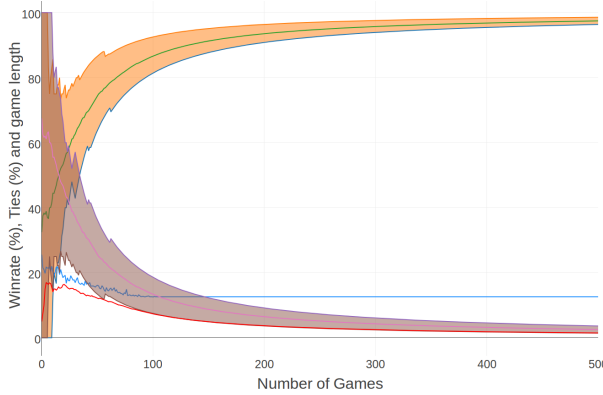
(d) Final agent trained vs Bot14



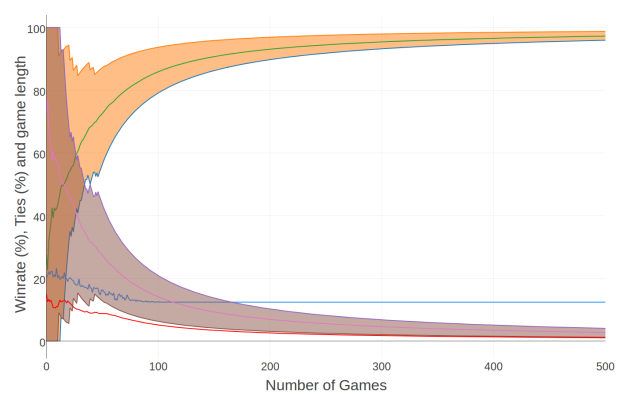
(e) Final agent vs Bot17



(f) Final agent trained vs Bot17



(g) Final agent vs Bot19



(h) Final agent trained vs Bot19

Figure 12.1: Interesting graphs of the final agent against bots

A good indication comes from figures 12.2a and 12.2b. It shows that the final agent clearly outperforms the basic opponent modeling model and proves that the adaptations we made aren't meaningless. There is also a high ties rate, near 40% but this is the case for every non-bot agent and this value is also achieved versus human beings.

The results against `s|s-6` (figures 12.2c and 12.2d) are a bit weird. Indeed, the game length is increasing with the number of games. Even if it remains between 0 and 20, this isn't a preferable behavior. The positive observation is that it seems by observing the mean rate that the agent eventually finds a good strategy (at 400 games without train and 300 with train).

The last analysis concerns figures 12.2e and 12.2f. Because of the difference between maximum and minimum win rate, conclusion can't be done before 100 games. This graph proves again that `GBR` is too much dependent of the initial random. Generally, we can say the final agent is better than `GBR` because of the slightly positive win rate's slope and the win rates are almost always greater than 50% (the only time where it isn't is before 100 games). After that, it seems our agent sometimes finds a strategy during a finite number of games (big slope on the extremums).

As a sum up we can affirm that our agent is really competitive against bots and other agents in general. The agent also recover pretty well even when wrongly induced by a training. The agent as a satisfactory win rate of more than 50% as a "normal" tie rate.

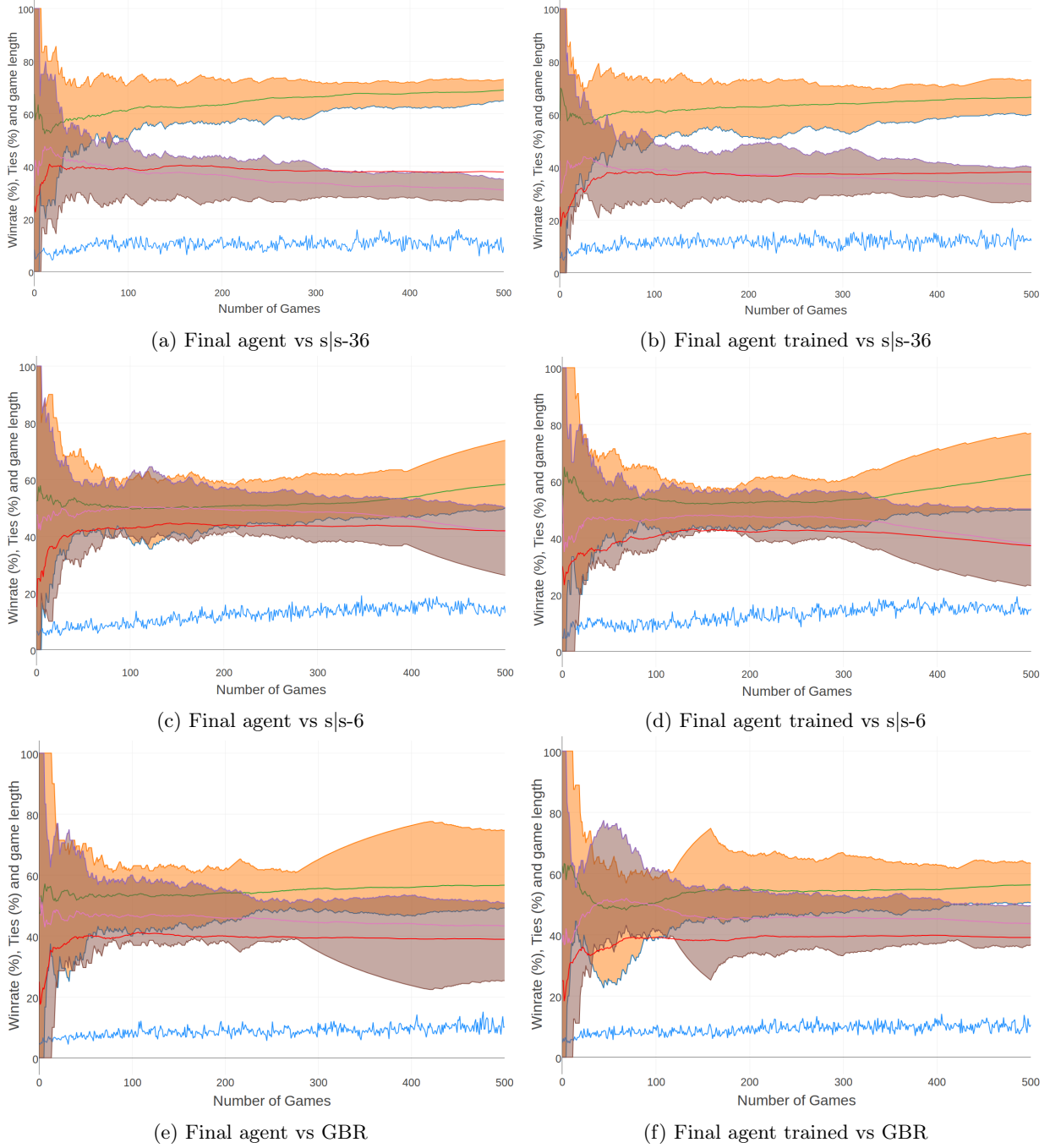
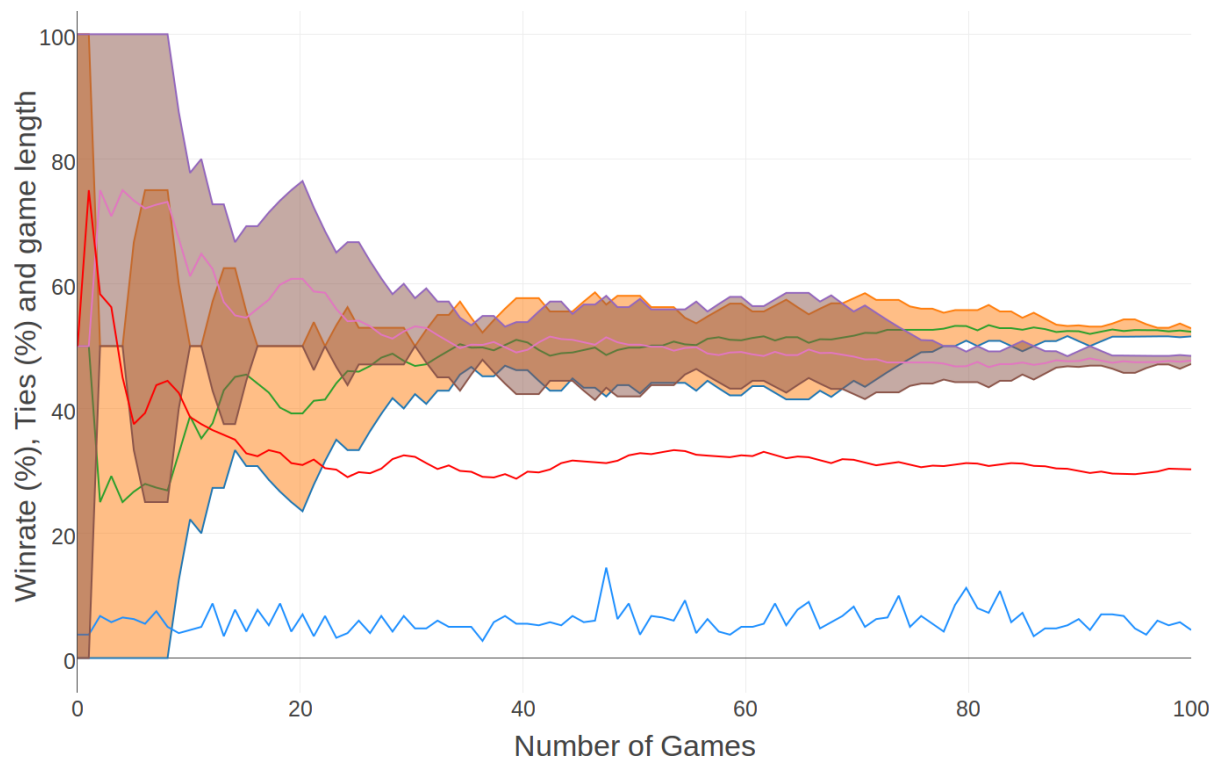


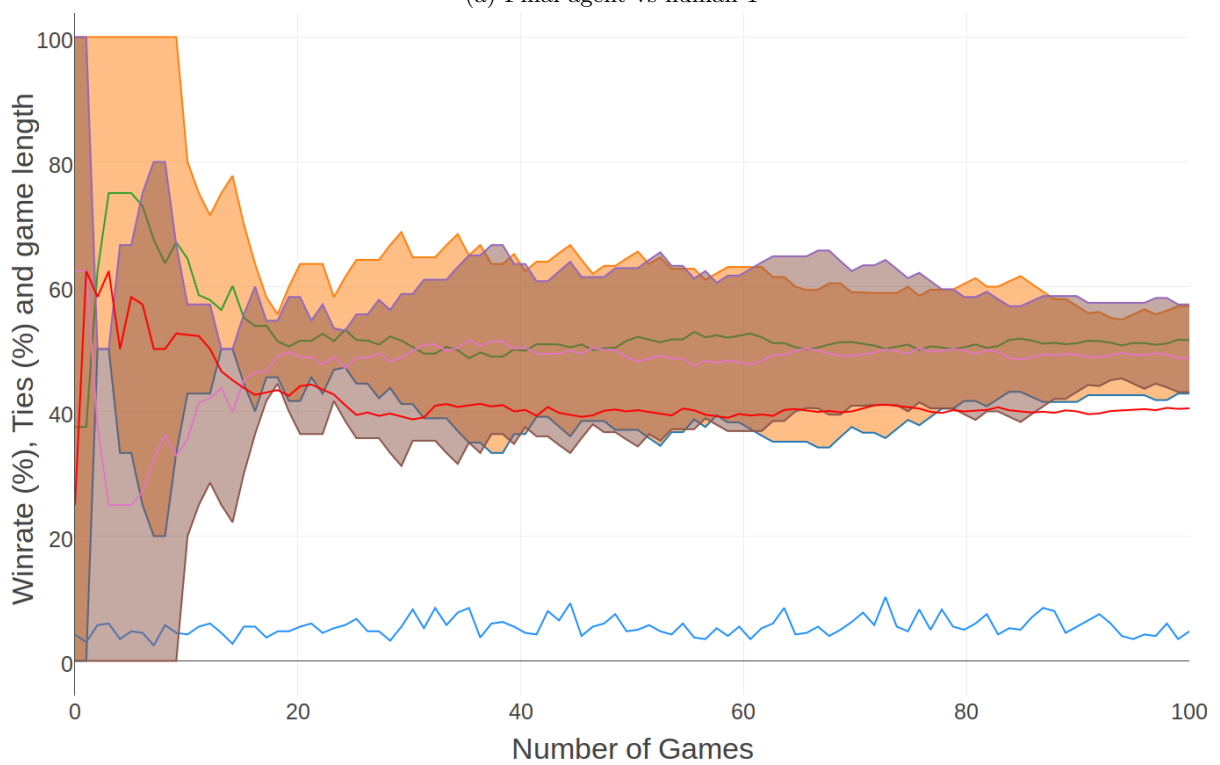
Figure 12.2: Interesting graphs of the final agent against smart agents

12.2.2 Performances against humans

This subsection opposes 2 humans and the agent. Graphs of the showdown are figures 12.3 and 12.4. There are 4 graphs because, since the agent was tested trained and untrained (as before). Each agent was tested 4 times for each figure.

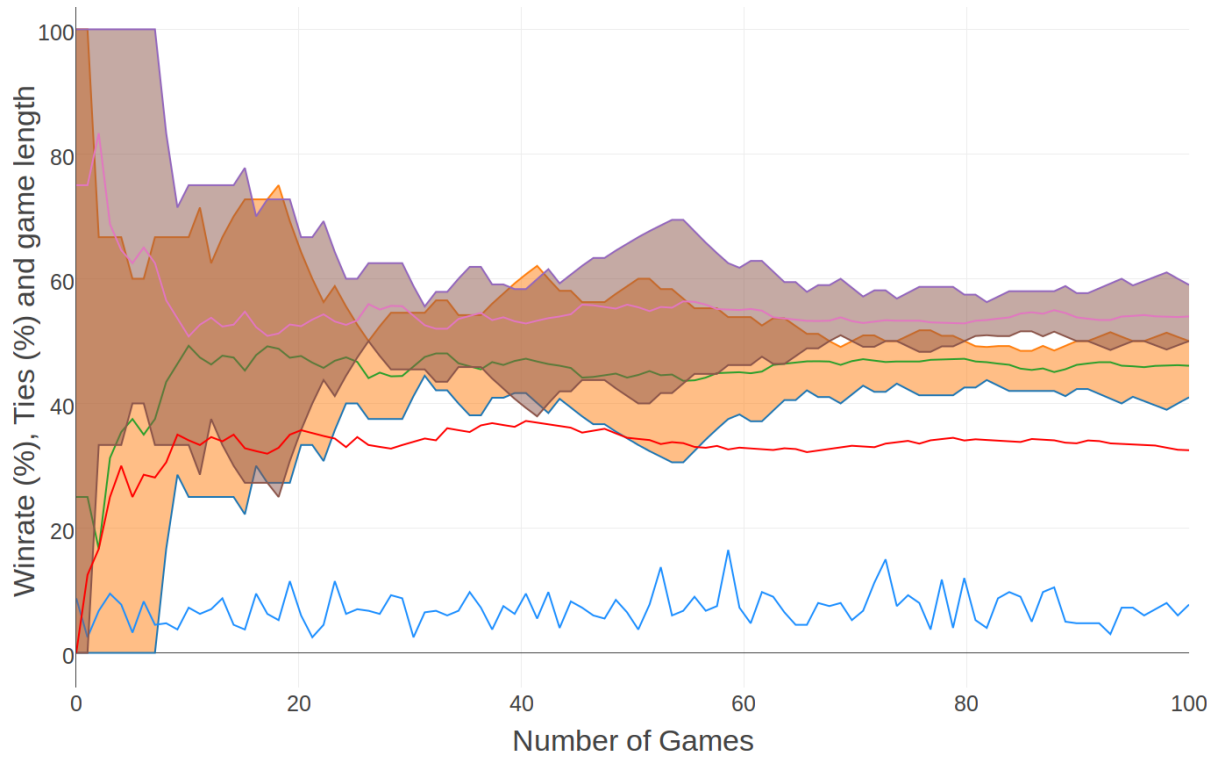


(a) Final agent vs human 1

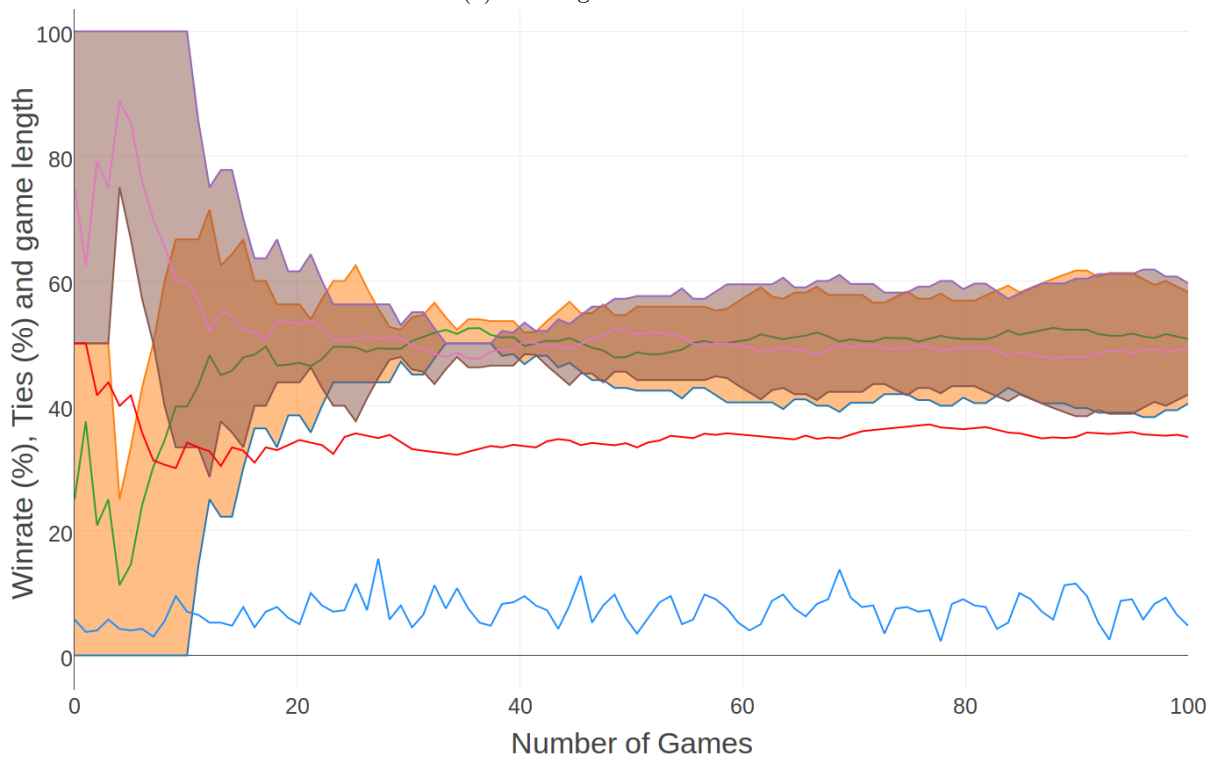


(b) Final agent trained vs Human 1

Figure 12.3: Final agent vs human 1



(a) Final agent vs human 2



(b) Final agent trained vs human 2

Figure 12.4: Final agent vs human 2

It is firstly important to compare the behavior of the agent with and without train against each human.

For the first human, the results before 40 games are clearly better with a train. It seems logic because the trained agent knows the rules since the beginning. Between 40 and 100 games, the results are similarly in means. However, after 80 games, the minimum win rate is better without train but it is only on 4 simulation which is mostly similar. Nothing can be concluded about the game length. The tie rate is a bit lower without train.

For the second set of graphs, although being quite similar, the trained agent seems a better in general. Indeed, the beginning is the same but the version without train has always a win rate mean lower than 50% (green curve) while the second version exceeds 50% after more or less 50 games (in means). Tie rates and game lengths are equivalent.

Another interesting comparison is between the behavior against human 1 and human 2. Figures 12.3a and 12.4a have the same general shape at the beginning for the win rates. The human clearly outperforms the agent until 20 games. Nevertheless, from this point until the end, the second human remains constant but it isn't the case for human 1. Its win rate continues to decrease. Tie rates are more or less the same and game length is smaller for human 1. The comparison between figures 12.3b and 12.4b is also interesting. After 100 games, the results are the same and in favor of our agent. In the 2 cases, the agent begins to win after between 40 and 50 games. The tie rates is lower for the second human. A general observation on the four graphs also shows that at the beginning for the 5-10 first games, continued win streaks (100% win) are possible either for the agent or the human since they don't know each other and the first result are simply based on "luck".

The only meaningful ending associated to these result is that after 100 games the human cannot dominate the agent anymore. The win rate indeed reach approximately 50% in mean when trained.

Chapter 13

Conclusion

This second part concludes the development of an agent capable of competing against bots, other agents and finally human beings. The sequence of improvements stumbled across different problems that possessed the original technique and its successors, from the lack of adaptability to the need of exploration.

These difficulties are typical during the creation of an agent and can't be totally optimized and rendered perfect. As our final bot may not be impeccable, it still shows some interesting results. A good win rate against bots, reasonable score against other agents and a fair tie against humans. Although we restricted our self to an unique technique that we tried to perfect, there exists dozens of other ways to design a good agent. But this wasn't our goal. We just needed an efficient agent that could be most real time as possible, our game being very fast, and that consumes as few as memory as possible since this is the principal problem of smartphones. This was achieved thanks to this model since thousand of game history can be summarized in a file smaller than 2mo and decisions are taken in a single look up and update loop. An interesting aspect for a future work could be to find models that are more on the edge of actual smartphones capabilities and therefore try to flirt with the limits.

Nevertheless this work is more a premise that shows the possibilities of smartphones and the feasibility/complexity of such a project. Machine learning on phones has only reached its first steps since the domain is new and still unknown for many. As presented in the first part, more and more people interest themselves on the subject and mostly about neural networks. The field will therefore grow, such as the hardware to support more and more of the possibilities brought by software and particularly ML. Or at least that is all that we hope for this fabulous domain.

Bibliography

- [1] Graham Templeton. What is moore's law, 2015.
<http://www.extremetech.com/extreme/210872-extremetech-explains-what-is-moores-law>.
- [2] Newell Allen, Shaw J.C., and Simon H.A. Chess - playing programs and the problem of complexity. 1958.
- [3] PEW Research Center. Global attitude survey : Q70, q71, q72. 2015.
- [4] KDnuggets. Computing resources for analytics, data mining, data science work or research poll. 2015.
- [5] Joshua Ho. The samsung galaxy s7 & s7 edge review, part 1. 2016.
- [6] Tom's Hardware. 2014 mobile hdd charts. 2014.
- [7] Tom's Hardware. 2014 ssd charts. 2014.
- [8] Chen Yiqi, Lin Zhiyuan, Pienta Robert, Kahng Minsuk, and Chau Duen Horng. Towards scalable graph computation on mobile devices. 2014.
- [9] Manuela Veloso. Reinforcement learning and plan recognition, 2011.
<http://www.cs.cmu.edu/reids/planning/handouts/RL-HMM.pdf>.
- [10] He Devin and Tom William. Evaluating model evaluation on mobile phones. 2016.
- [11] Kurenkov Andrey, 2016. A 'Brief' History of Game AI Up To AlphaGo.
- [12] Shannon Claude E. Programming a computer for playing chess. *Philosophical Magazine, Ser.7, Vol. 41, No. 314, XXII*, 1950.
- [13] AAAI. Chess topic on aai website. 2016.
- [14] Grimsdale Chris. The history of alan turing and the universal turing machine. <https://chrisgrimsdale.wordpress.com/the-history-of-alan-turing-and-the-universal-turing-machine/>.
- [15] Jonathan Schaeffer. *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer, 1997.
- [16] Arthur Samuel. Some studies in machine learning using the game of checkers. page 210–229, 1959.
- [17] Schaeffer Jonathan, Lake Robert, Lu Paul, and Bryant Martin. Chinook - the world man-machine checkers champion. *AI Magazine Volume 17 Number 1*, 1996.

- [18] Tesauro Gerald and Sejnowski Terrence. A parallel network that learns to play backgammon. 1989.
- [19] Rieser Verena and Lemon Oliver. Reinforcement learning for adaptive dialogue systems (theory and applications of natural language processing). 2011.
- [20] Tom M. Mitchell. The discipline of machine learning. July 2006.
- [21] English dictionary. <http://www.collinsdictionary.com/dictionary/english/>.
- [22] Jean-Marc Vincent and Arnaud Legrand. Performance measurements of computer systems: tools and analysis - master 2r sl module md. http://mescal.imag.fr/membres/arnaud.legrand/teaching/2006/M2R_EP_Measurements.pdf, 2007.
- [23] Nick T. A modern smartphone or a vintage supercomputer: which is more powerful ? 2014.
- [24] Shengkui Gao. A survey of latest performance, development and measurement issues of smart phones design. 2011.
- [25] Dan Rowinski. How qualcomm is truly taking machine learning to the edge. 2016.
- [26] Ignacio Contreras. Qualcomm research brings server-class machine learning to everyday devices—making them smarter. 2015.
- [27] Experts Exchange. Processing power compared : Visualization a trillion-fold increase in computer performance. 2015.
- [28] Takac L. and Zabovsky M. Data analysis in public social networks. 2012.
- [29] Backstrom L., Huttenlocher D., Kleinberg J., and Lan X. Group formation in large social networks: membership, growth, and evolution. 2006.
- [30] Mislove A., Marcon M., Gummadi K. P., Druschel P., and Bhattacharjee B. Measurement and analysis of online social networks. 2007.
- [31] Gong N. Z., Xu W., Huang L., Mittal P., Stefanov E., Sekar V., and Song D. Evolution of social-attribute networks: measurements, modeling, and implications using google+. 2007.
- [32] Apple inc. Memory usage performance guidelines. 2013.
- [33] Google. How google translate squeezes deep learning onto a phone. 2015.
- [34] Roger B. Myerson. Game theory: Analysis of conflict. *Harvard University Press*, page 1, 1991.
- [35] Bosansky Branislav. Iterative algorithms for solving finite sequential zero-sum games. page 7, 2014.
- [36] Don Ross. Game theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2016 edition, 2016.
- [37] Kuhn Harold William. Classics in game theory. page 65, 1953.
- [38] Manea Mihai. Non-cooperative games, department of economics, mit. 2010.

- [39] William Poundstone. Prisoner's dilemma: John von neumann, game theory, and the puzzle of the bomb. 1992.
- [40] Cantu Harper. The game theory handbook - everything you need to know about game theory. 2016.
- [41] Sergiu Hart. Games in extensive and strategic forms. In Elsevier, editor, *Handbook of Game Theory with Economic Applications 1*. 1992.
- [42] Cram101 Textbook Reviews. e-study guide for game theory - an introduction - statistics. 2014.
- [43] Chatain Olivier. Cooperative and non-cooperative game theory. 2014.
- [44] Cram101 Textbook Reviews. Study resource for anderson's - an introduction to management science. 2016.
- [45] Hogarth Dale. Program for solving two-person zero-sum games. 2006.
- [46] R. Apt Krzysztof. Chapter 6 : Strictly competitive games. 2011.
- [47] Von Neumann John and Morgenstern Oskar. Theory of games and economic behavior. 1956.
- [48] Shapley Lloyd S. Stochastic games. 1950.
- [49] Mac Dermed Liam, Isbell Charles L., and Weiss Lora. Markov games of incomplete information for multi-agent reinforcement learning. 2011.
- [50] R. Apt Krzysztof. Chapter 3 : Strict dominance. 2011.
- [51] Osborne, J. Martin, and Rubinstein Ariel. A course in game theory (mit). 1994.
- [52] Jemal H. Abawajy. Network and traffic engineering in emerging distributed computing applications. 2013.
- [53] R. Apt Krzysztof. Chapter 1 : Nash equilibrium. 2011.
- [54] R. Apt Krzysztof. Chapter 2 : Nash equilibria and pareto efficient outcomes. 2011.
- [55] Cook Richard and Ryan Clare. Rock paper scissors players are natural copycats. 2011.
- [56] Wang Zhijian, Xu Bin, and Zhou Hai-Jun. Social cycling and conditional responses in the rock-paper-scissors game. 2014.
- [57] Dance Gabriel and Tom Jackson. Rock-paper-scissors: You vs. the computer. 2011.
- [58] Salen Tekinbaş Katie and Zimmerman Eric. *Rules of Play : Game Design Fundamentals*. MIT Press, 2003.
- [59] Wolfgang Kramer. What is a game ?, December 2000. <http://www.thegamesjournal.com/articles/WhatIsaGame.shtml>.
- [60] How to play the shotgun game, 2015. <http://www.wikihow.com/Play-the-Shotgun-Game>.
- [61] Pierre Dupont and Cédrick Fairon. Ucl - ingi2263: Hidden markov models.

- [62] Chain rule. https://www.eecs.qmul.ac.uk/norman/BBNs/Chain_rule.htm.
- [63] Marco Saerens. Ucl - sinf2275: Decision making.
- [64] Florentin Woergoetter and Bernd Porr. Reinforcement learning, 2008. http://www.scholarpedia.org/article/Reinforcement_learning.
- [65] Gonçalo Neto. Single-agent to multi-agent reinforcement learning: Foundational concepts and methods. Learning theory course.
- [66] Bakkes Sander C.J., Spronck Pieter H.M., and van den Herik H. Jaap. Opponent modelling for case-based adaptive game ai. 2009.
- [67] Yogeswaran Mohan and Ponnambalam S. G. Reinforcement learning: exploration–exploitation dilemma in multi-agent foraging task. September 2012.

Chapter 14

Appendices

14.1 Game's action table

<u>Your choice</u>	<u>Opponent's choice</u>	<u>Immediate effect</u>
Reload	Reload	Increase your power and the power of the opponent by 1
Reload	Shield	Increase your power by 1
Reload	Fire	You loose
Shield	Reload	Increase the power of the opponent by 1
Shield	Shield	Nothing happens
Shield	Fire	If the opponent's power of the shoot is strictly greatest than the solidity of your shield, you loose. Else, decrease the solidity of your shield by the opponent's power of the shoot
Fire	Reload	You win
Fire	Shield	If the power of your shoot is strictly greatest than the solidity of the opponent's shield, you win. Else, decrease the solidity of the opponent's shield by the power of your shoot
Fire	Fire	It's a tie

Table 14.1: All possible game situations.

14.2 The simulator

To really estimate the skills of an agent/IA, we need to challenge him during a lot of games and have an overview of these games. We can't reasonably test individually each agent variant. Therefore, its opponent can't only be a human and we need a simulator. A good simulator is a simulator with the possibility to easily implement a new agent. It must also have a lot of options to tune each meta-parameters and to have different outputs so that we can, as suggest before, estimate the performance of each agent and find the best ones. To really have a notion of how our agent behave, a GUI and graphs are available.

Our simulator was written in python3. To execute it, you first need to install 2 libraries (found on pip3), pytz and numpy for graphs and calculations respectively.

The next sections are used to talk about the usage of our simulator section 14.2.1 and the creation of an agent if a user wants to develop its own IA (section 14.2.2).

14.2.1 Input and output

The command line to launch the simulator (if you are in the right directory) is shown on listing 14.1 with each positional/optional argument defined at table 14.2. In addition to this table, we have a verbosity level optional argument ("v"). The effect of this argument depends of the number of "v" as you can see on table 14.3 and influenced the output of the simulation. Notice that if you want to play against an agent, you need to use "-v" or "-vvvvv" as verbosity and use Agent.human_agent as p1 or p2 argument. For each round, you must select between the keys "r" (reload), "s" (shield), "1" (fire power 1), "3" (fire power 3), "5" (fire power 5) and "0" (fire power 10).

```
python3 matchmaker.py [-h] [-a1 ALPHAP1] [-g1 GAMMAP1] [-dsh1] [-a2 ALPHAP2] [-g2 GAMMAP2] [-dsh2] [-v] [-d DELAY] [-r REPEAT] [-sd SAVEDIRECTORY] ngames p1 p2
```

Listing 14.1: Command line to launch the simulator

Argument	Type	Purpose
ngames	int	Number of games to simulate
p*	string	Agent name for player *(1/2) (you must use <code>directory.name</code>)
-h, --help	/	Show the help message
-a*, --alphaP*	float	Define, if needed, the initial α (learning rate) value of the player *
-g*, --gammaP*	float	Define, if needed, the initial γ (discount factor) value of the player *
-dsh*, --dontSaveHistoryP*	/	Used if you don't want to save the history file created during this simulation by the player *
-d, --delay	float	Delay between each action (useful with -v)
-r, --repeat	int	Number of time the simulation is launched
-sd, --saveDirectory	string	the sub-directory where the graphs are stored in Graph

Table 14.2: Definition of each positional/optional argument of the simulator

As discussed before, the output depends of the verbosity specified in the input. We don't need to be more precise about the standard output because it is really intuitive. When asked,

Level	Consequence on the output
0*v	No output
1*v (-v)	The games are shown on the GUI (on standard output)
2*v (-vv)	The games are shown as a text (on standard output)
3*v (-vvv)	Nothing but a short summary appears on the standard output and graphs of the simulation are created
4*v (-vvvv)	Makes the action of -vv and -vvv
5*v (-vvvvv)	Makes the action of -v and -vvv

Table 14.3: Action of each verbosity level on the output

different graphs are produced (html files by default in the Graph/default directory) :

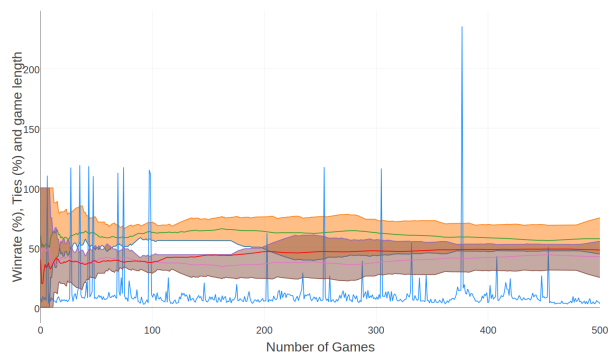
- A graph with the action distribution of each player to see the usage each player made of each action
- A graph with the value of the shield of each player at the end of each game
- A graph with the evolution of the winrate (if "-r" is specified with a value greater than 1, you can see the minimum, maximum and mean of the win rate among the simulations). You can also observe the evolution of the game length and the tie rate.

14.2.2 General implementation of each agent

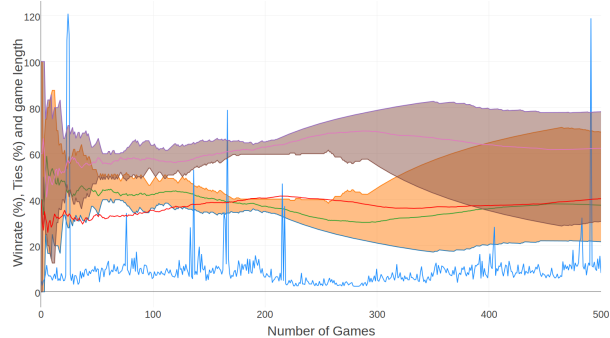
The definition of each agent must inherit the Agent class specified in "skeleton.py" and be in the same directory as this file. If someone wants to create an agent, here are the functions he can re-implement :

- `__init__` : this function is called one time for each simulation and so, the number of times specified by the "-r" option. It initializes the agent and must at least call the constructor of Agent. This constructor takes several parameters as the name of the agent or the file where the developer wants to save an history. Then, a developer can declare the instance variables he wants to use.
- `agent_init` : this function is called between each game when a game begins.
- `play` : called at each round of each game. It is the main function to modify. It must return the action of the agent for the current round. (only mandatory function to implement)
- `postplay` : this method is called at each round when each player has chosen and played his action.
- `endGame` : finally this method is called between each game of a simulation.

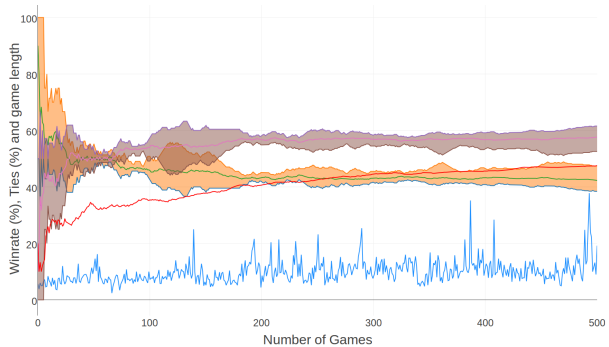
14.3 Graphs associated with table 9.2



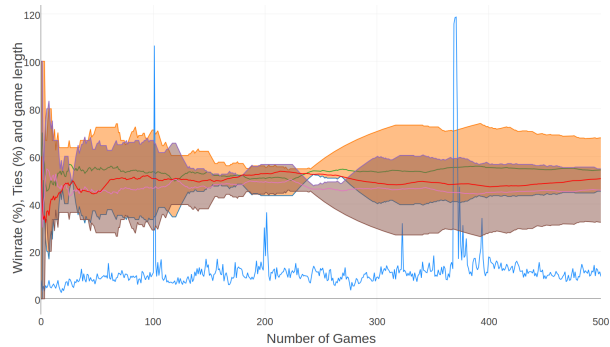
(a) GBR vs s|s-36



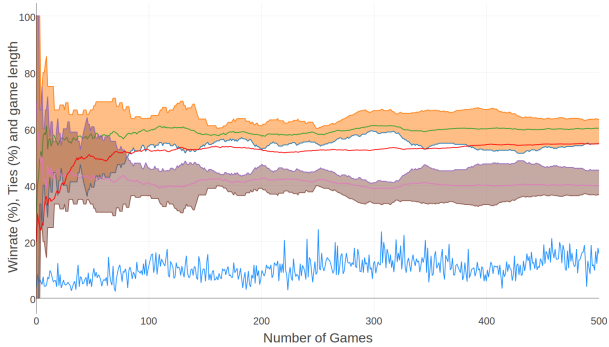
(b) GSR vs GBR



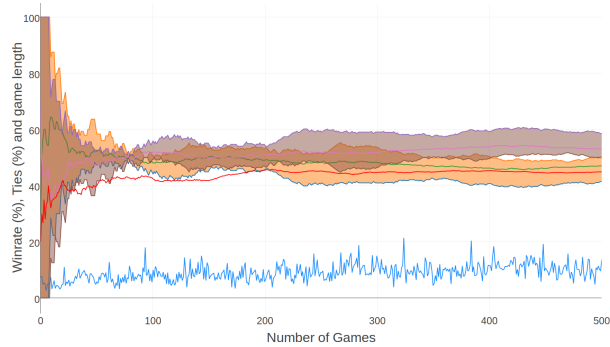
(c) B-T0.1 vs GBR



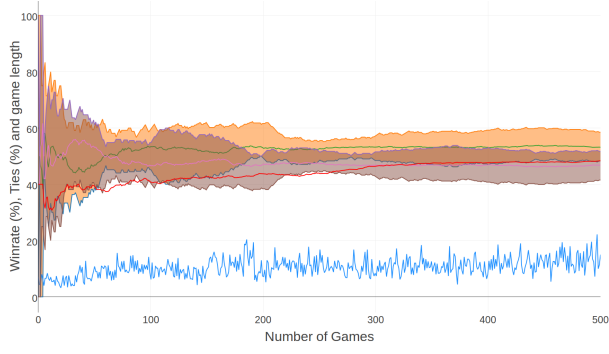
(d) B-T0.1 vs GSR



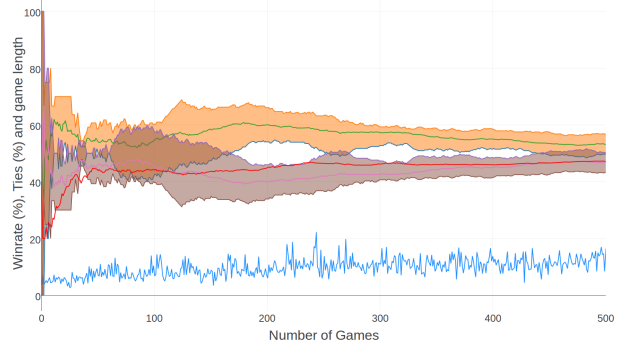
(e) B-T0.5 vs s|s-36



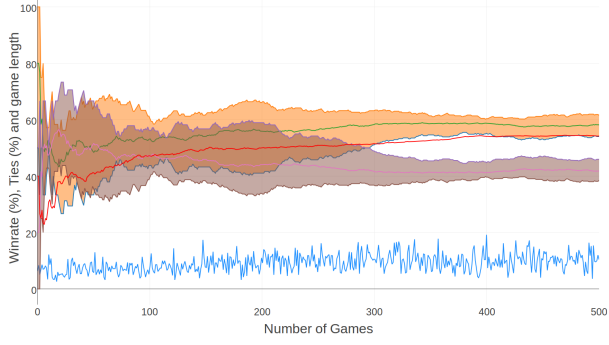
(f) B-T0.5 vs GBR



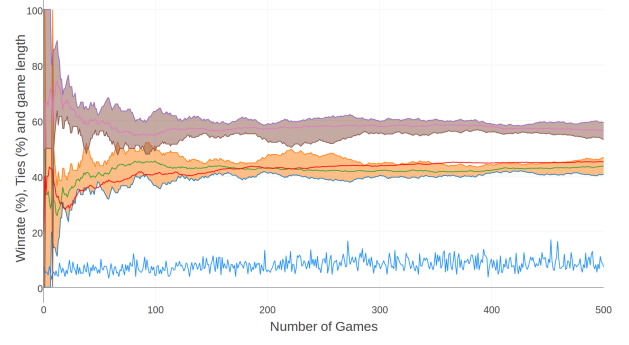
(g) B-T0.5 vs GSR



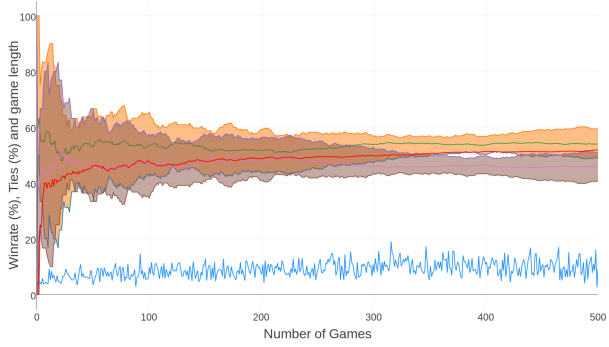
(h) B-T0.5 vs B-T0.1



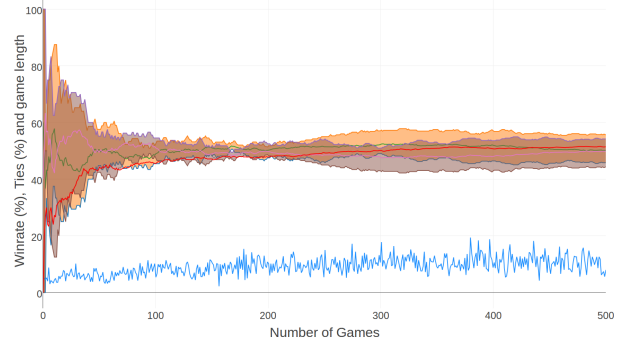
(i) B-T1 vs s|s-36



(j) B-T1 vs GBR



(k) B-T1 vs GSR



(l) B-T1 vs B-T0.1

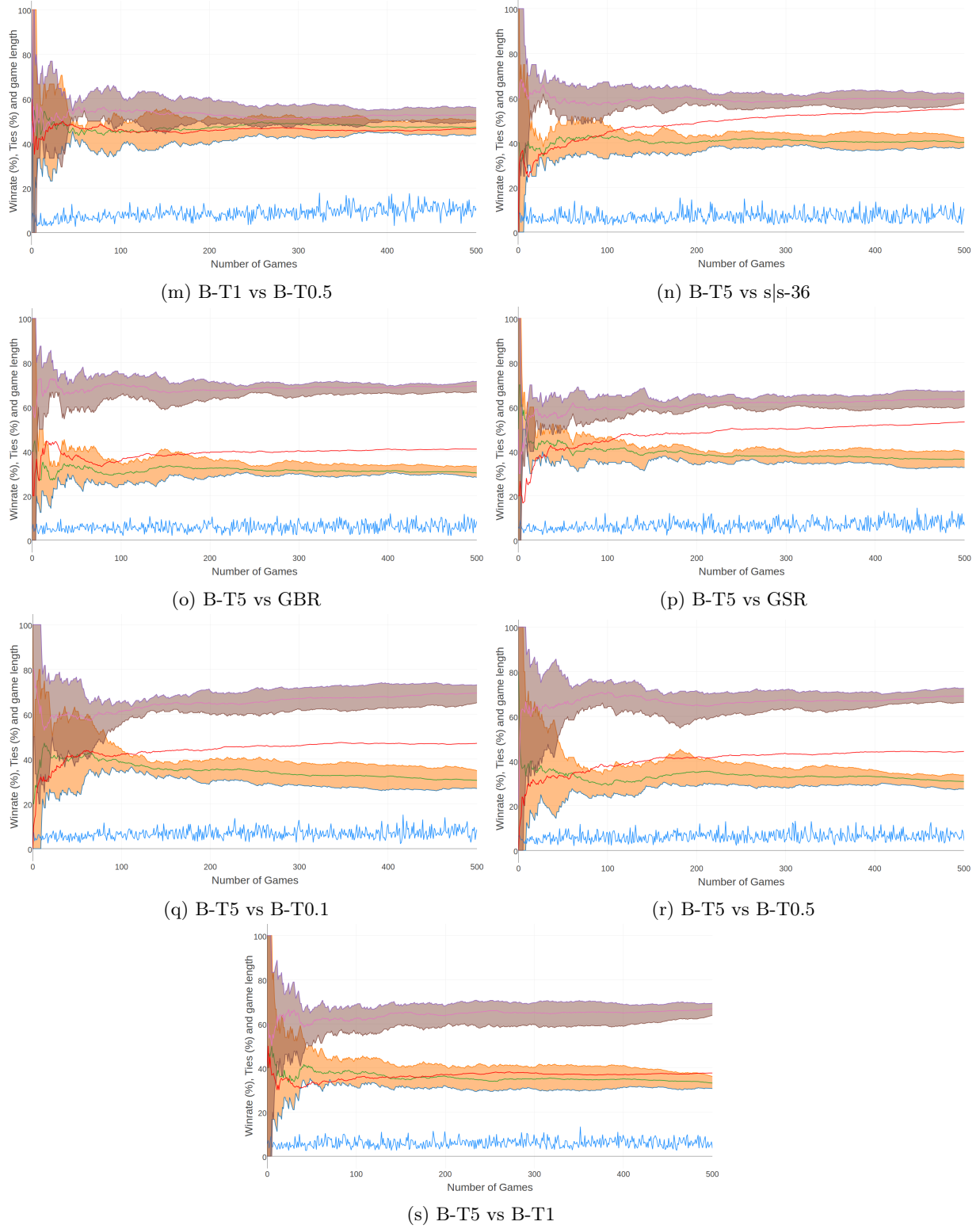
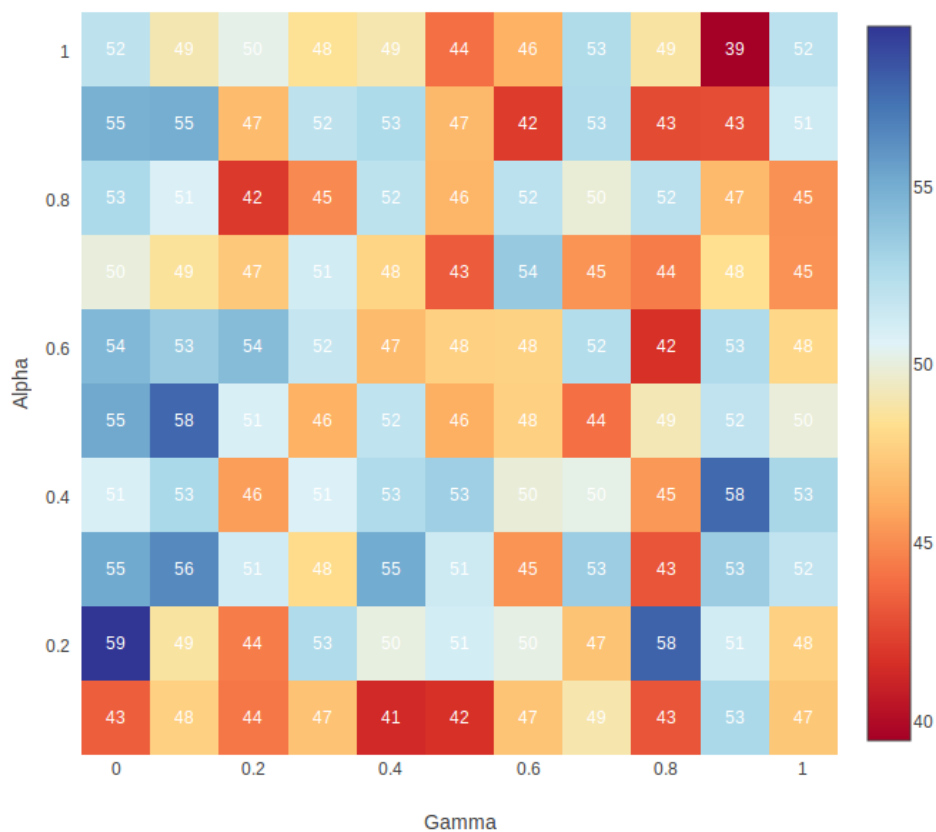
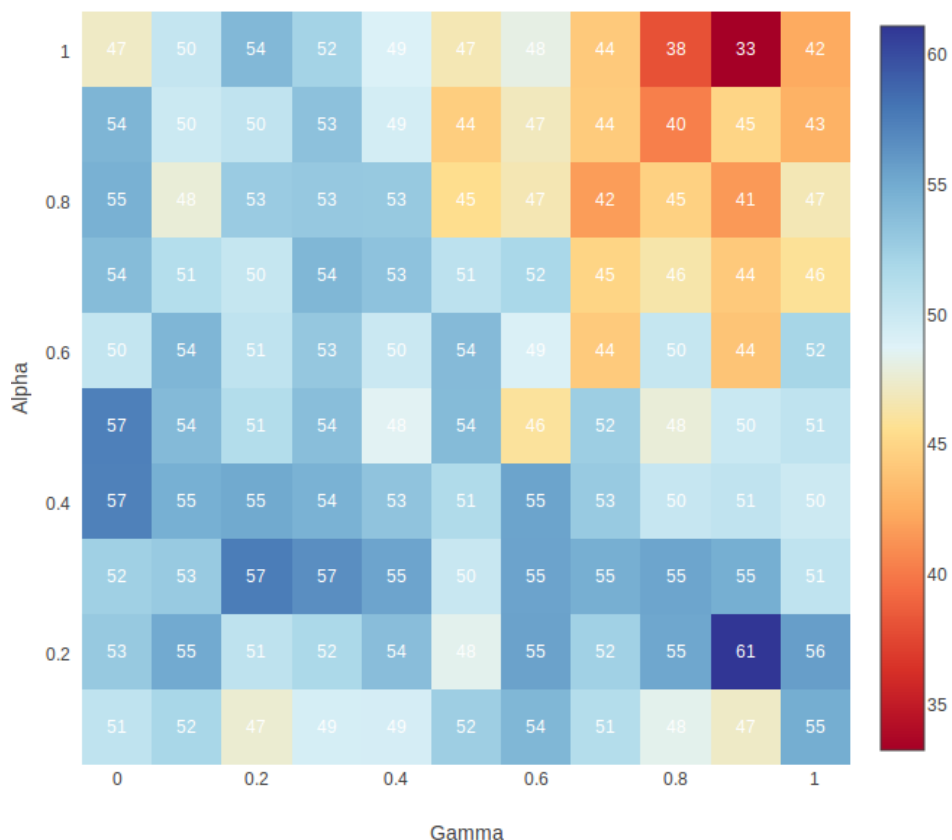


Figure 14.1: Graphs associated with table 9.2

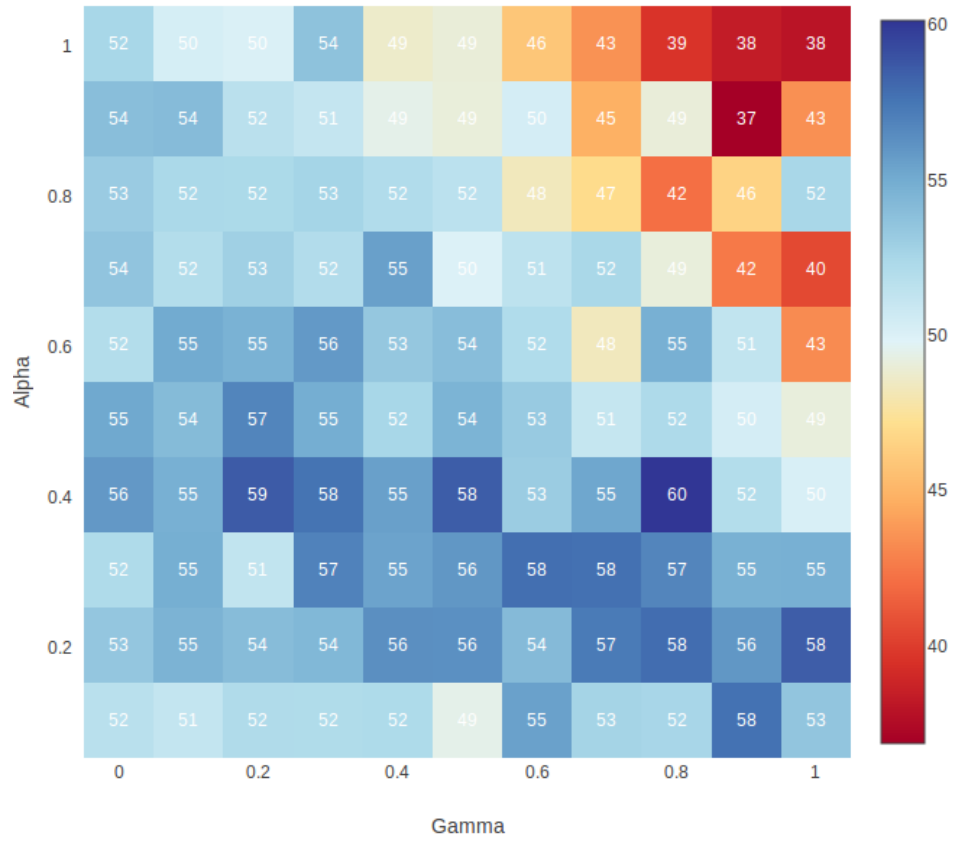
14.4 Heat maps of B-T0.5 against GSR, B-T0.1 and GBR



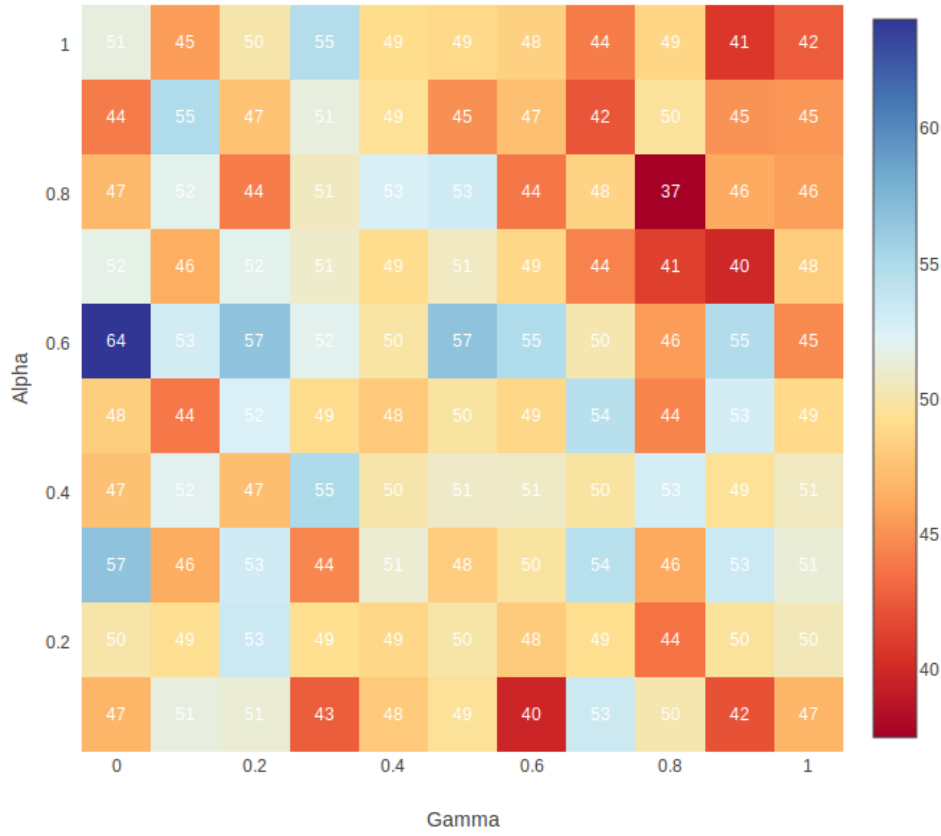
(a) B-T0.5 vs GSR on 50 games



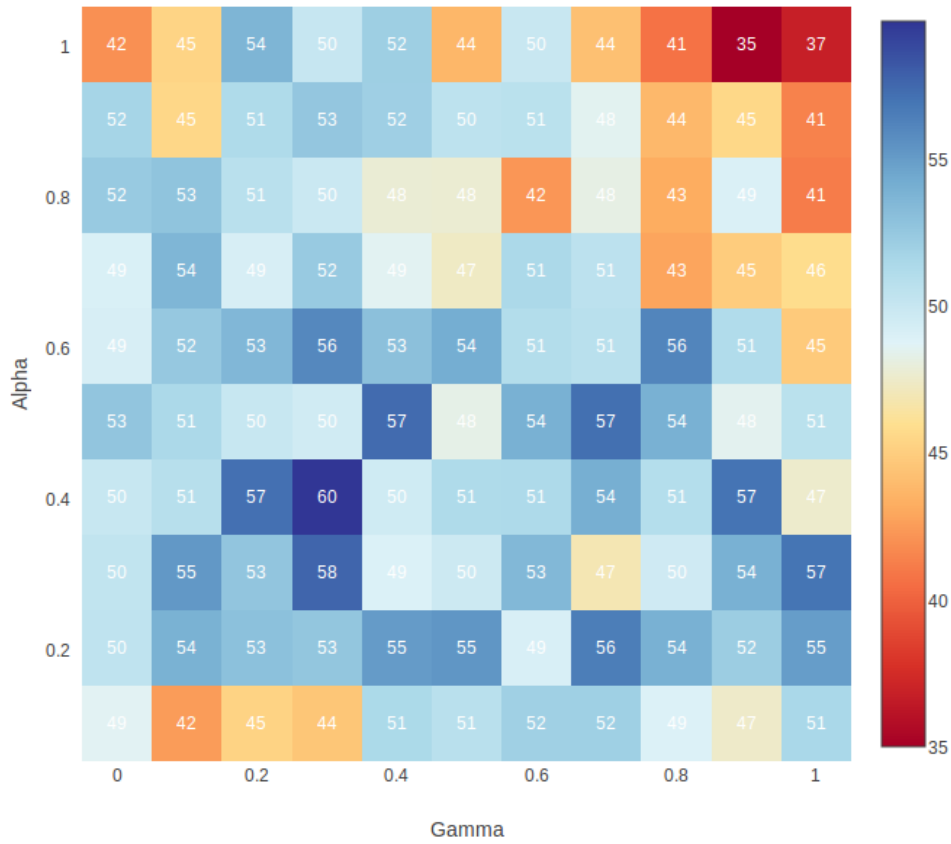
(b) B-T0.5 vs GSR on 100 games



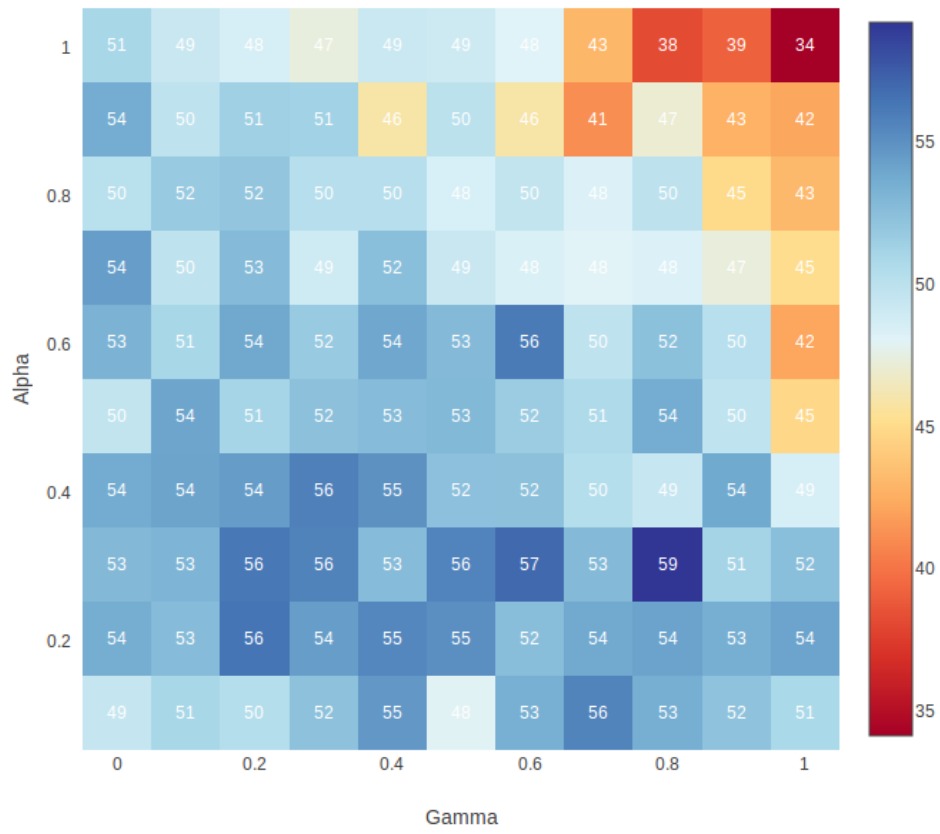
(c) B-T0.5 vs GSR on 500 games



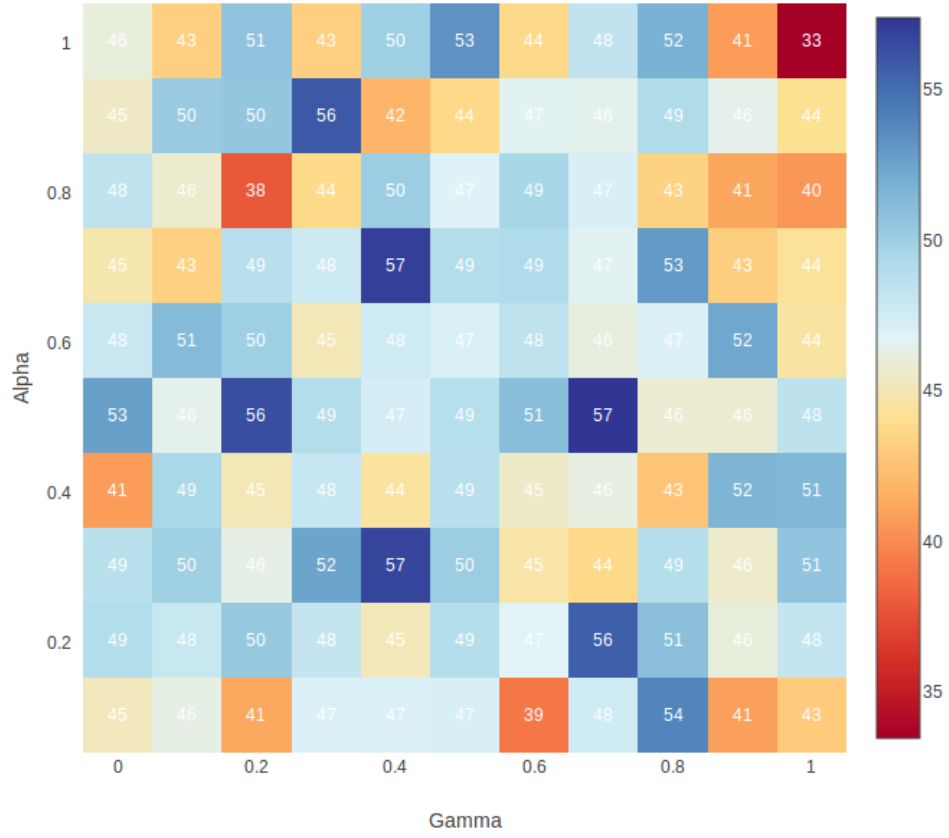
(d) B-T0.5 vs B-T0.1 on 50 games



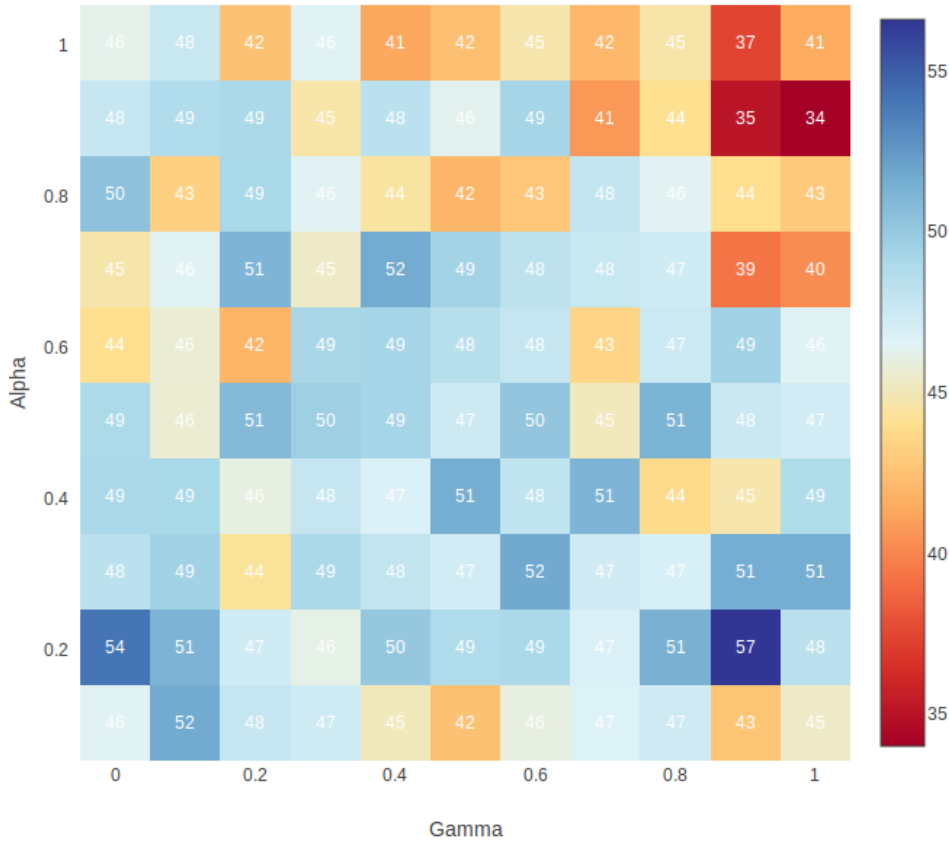
(i) B-T0.5 vs B-T0.1 on 100 games



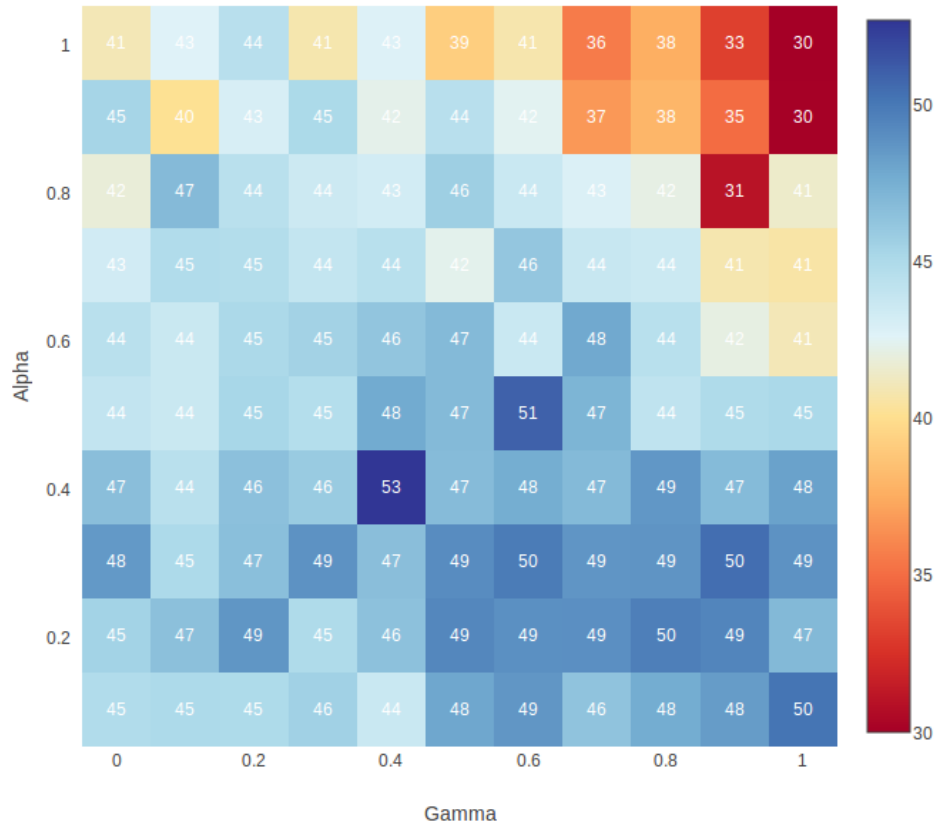
(d) B-T0.5 vs B-T0.1 on 500 games



(g) B-T0.5 vs GBR on 50 games

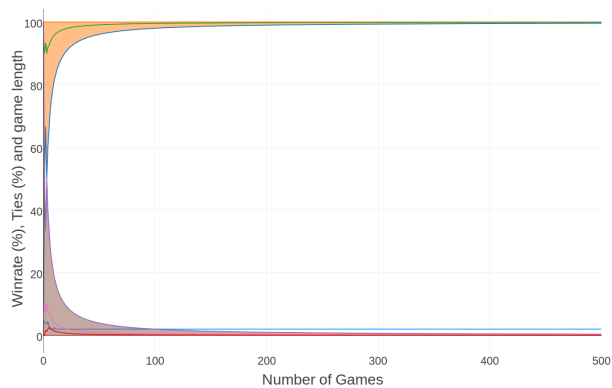


(h) B-T0.5 vs GBR on 100 games

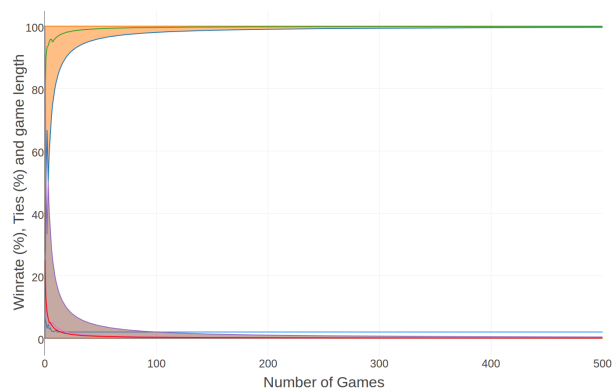


(i) B-T0.5 vs GBR on 500 games

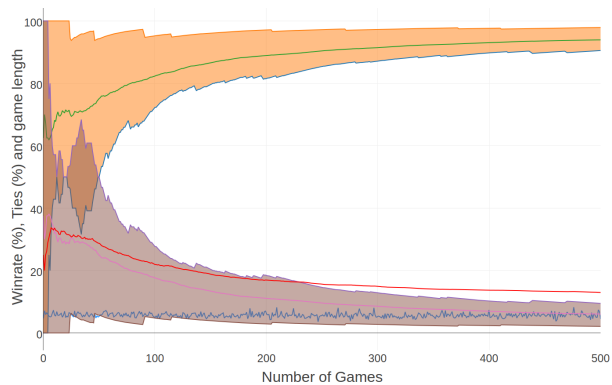
14.5 Graphs associated with table 12.1 and 12.3



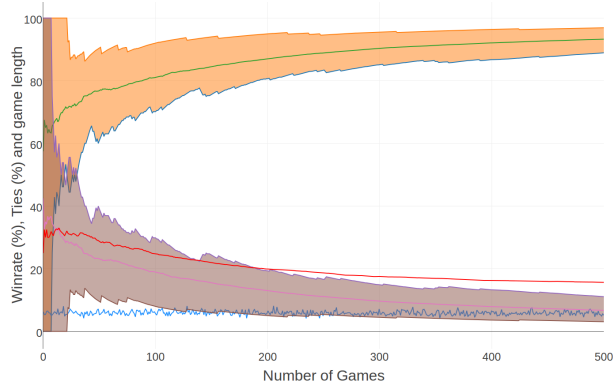
(a) Final agent vs Bot1



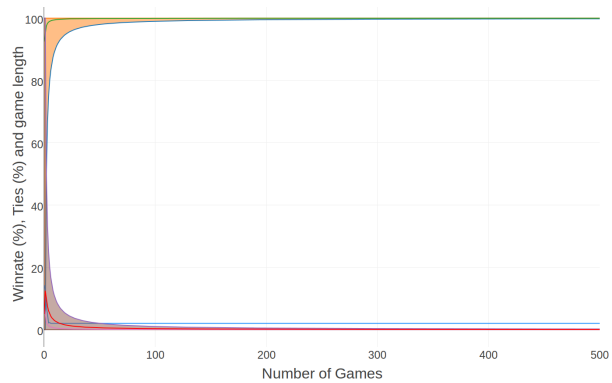
(b) Final agent trained vs Bot1



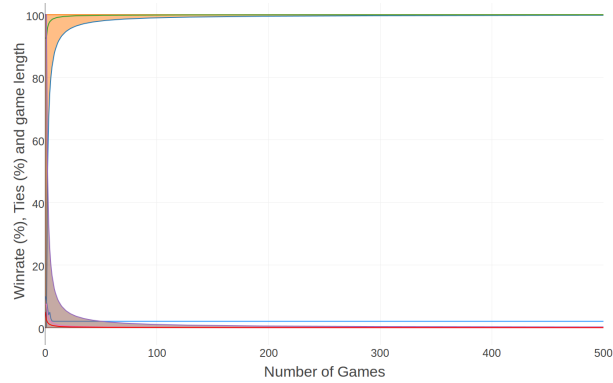
(c) Final agent vs Bot2



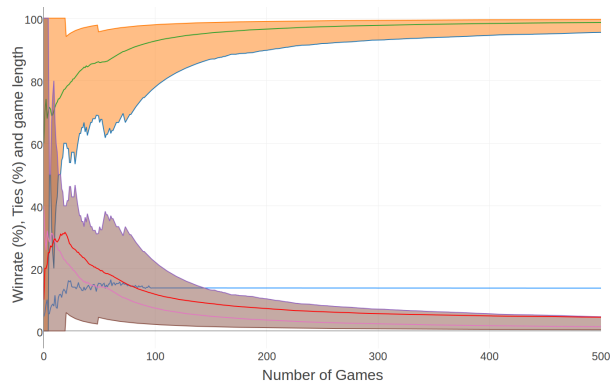
(d) Final agent trained vs Bot2



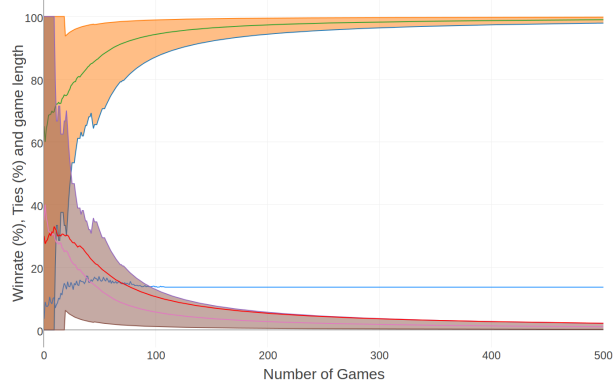
(e) Final agent vs Bot3



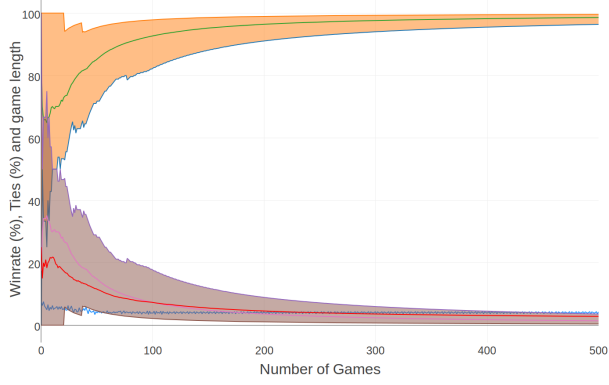
(f) Final agent trained vs Bot3



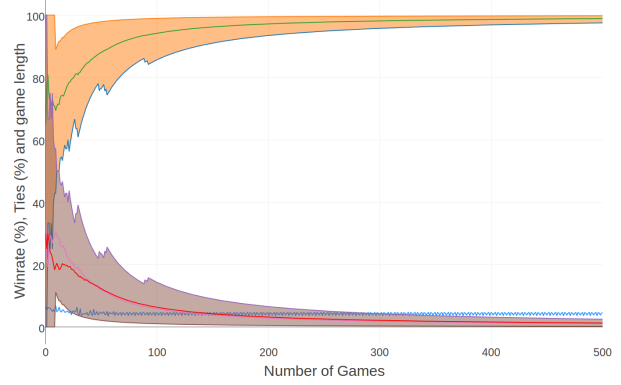
(g) Final agent vs Bot4



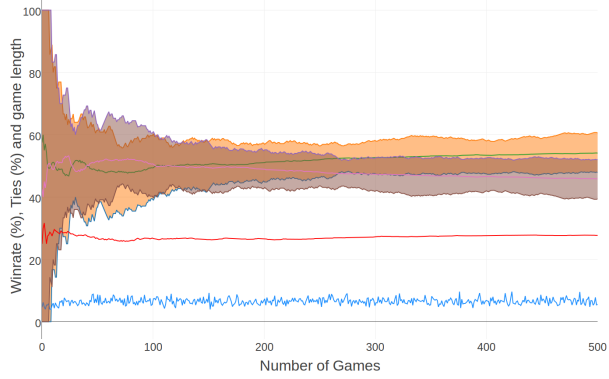
(h) Final agent trained vs Bot4



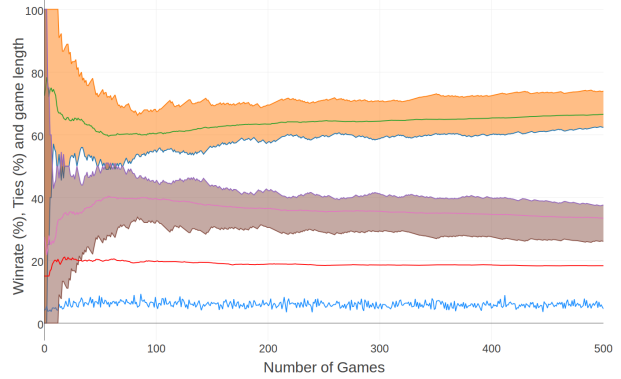
(a) Final agent vs Bot5



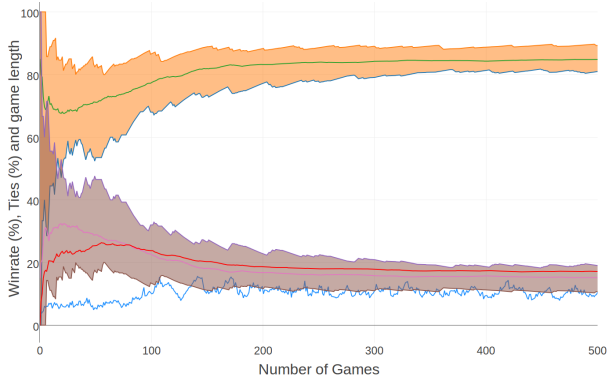
(b) Final agent trained vs Bot5



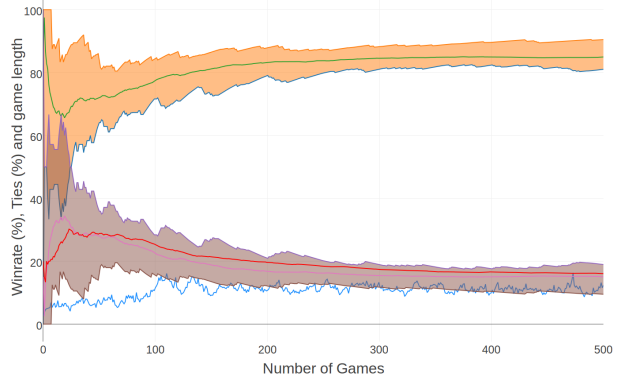
(c) Final agent vs Bot6



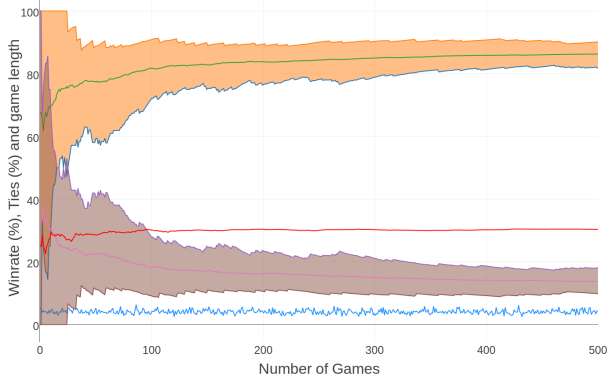
(d) Final agent trained vs Bot6



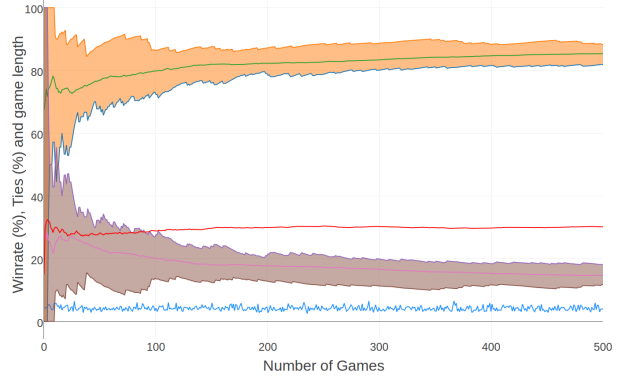
(e) Final agent vs Bot7



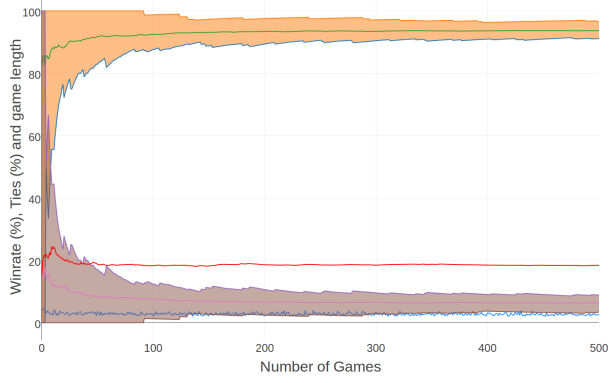
(f) Final agent trained vs Bot7



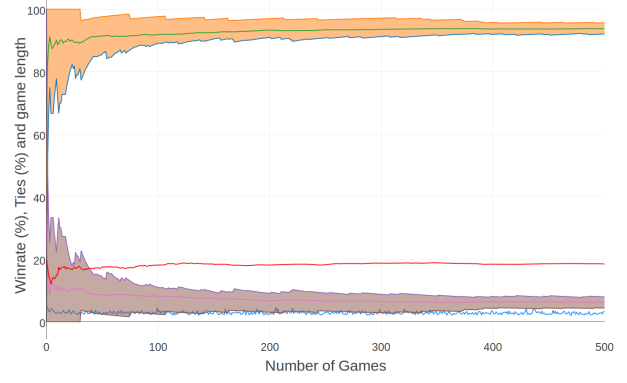
(g) Final agent vs Bot8



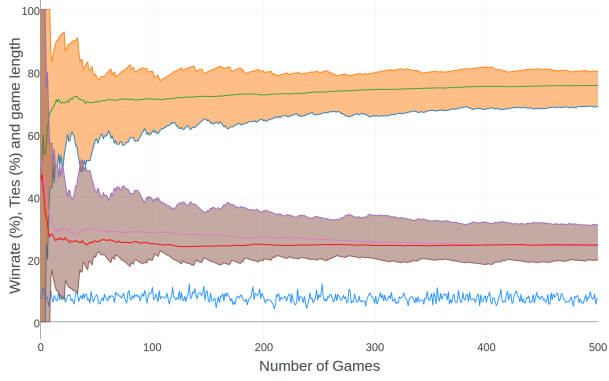
(h) Final agent trained vs Bot8



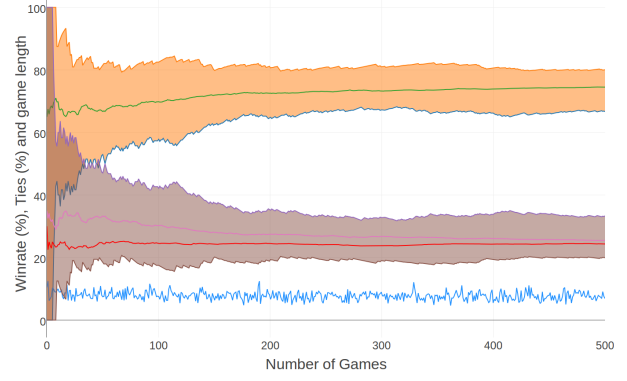
(a) Final agent vs Bot9



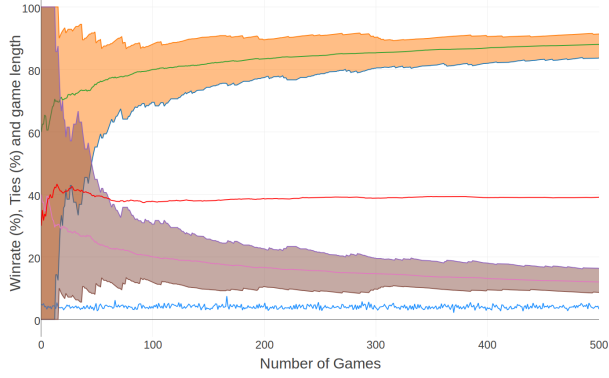
(b) Final agent trained vs Bot9



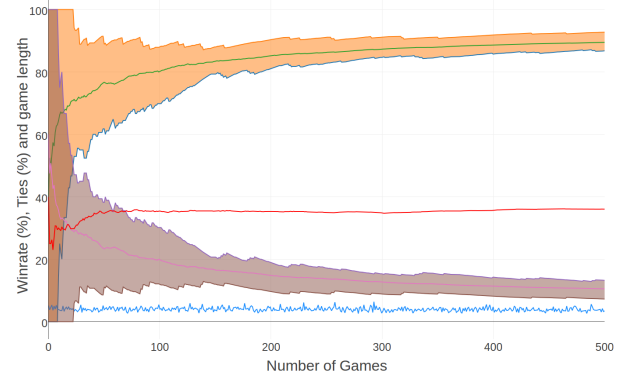
(c) Final agent vs Bot10



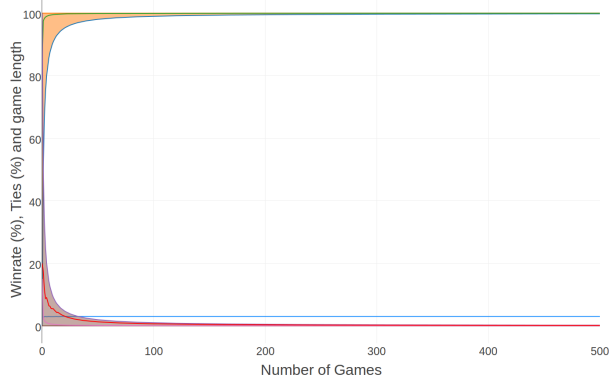
(d) Final agent trained vs Bot10



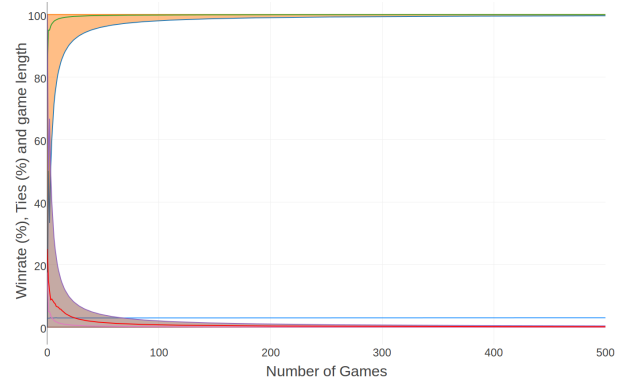
(e) Final agent vs Bot11



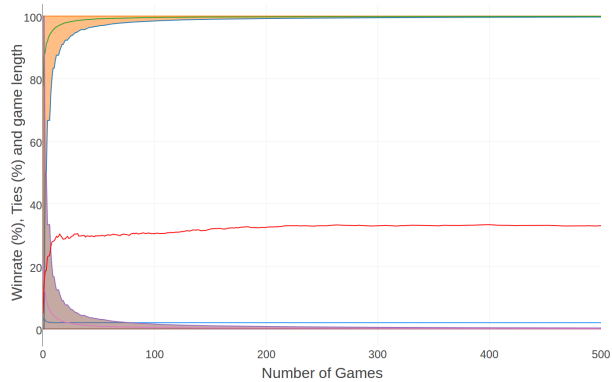
(f) Final agent trained vs Bot11



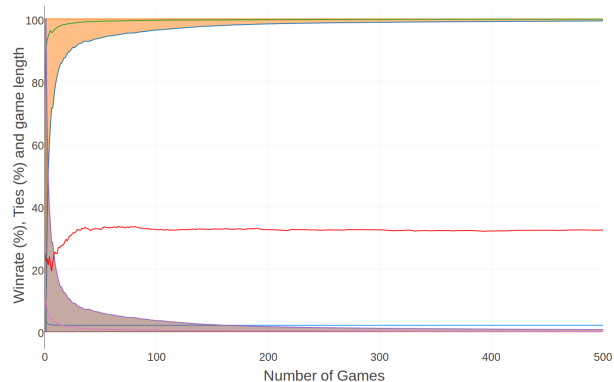
(g) Final agent vs Bot12



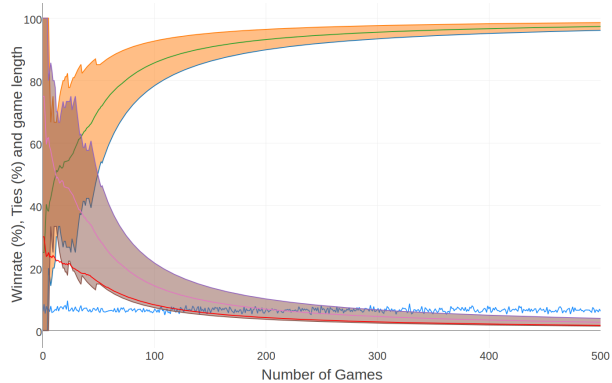
(h) Final agent trained vs Bot12



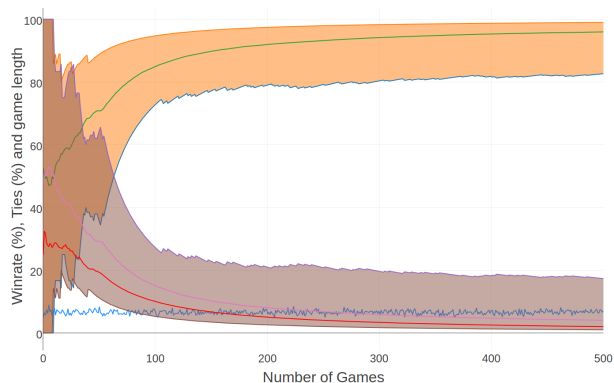
(a) Final agent vs Bot13



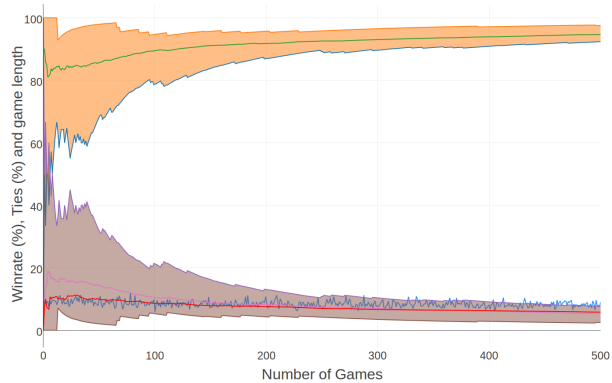
(b) Final agent trained vs Bot13



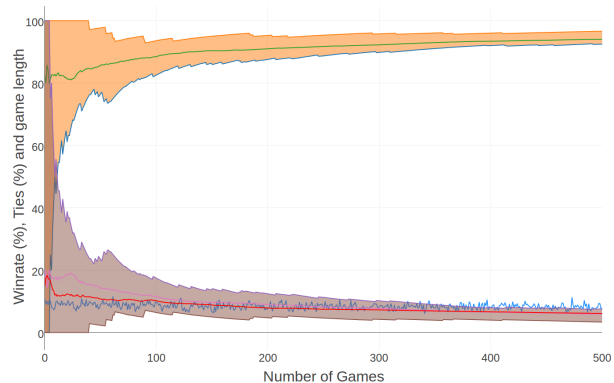
(c) Final agent vs Bot14



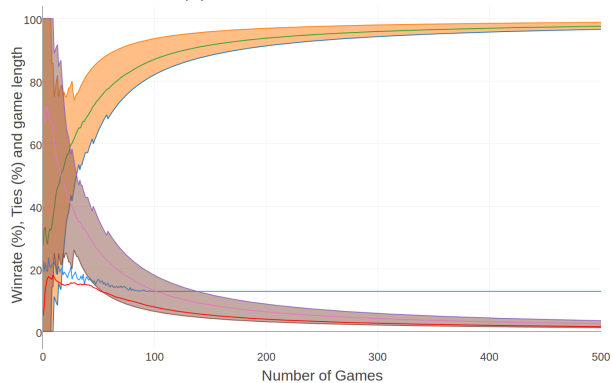
(d) Final agent trained vs Bot14



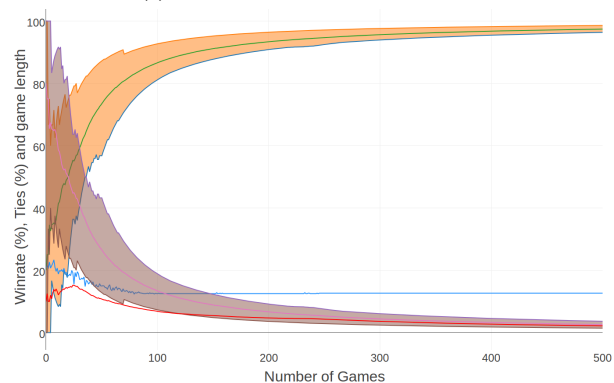
(e) Final agent vs Bot15



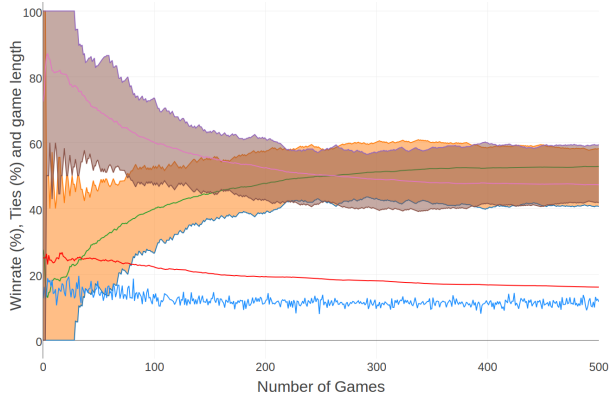
(f) Final agent trained vs Bot15



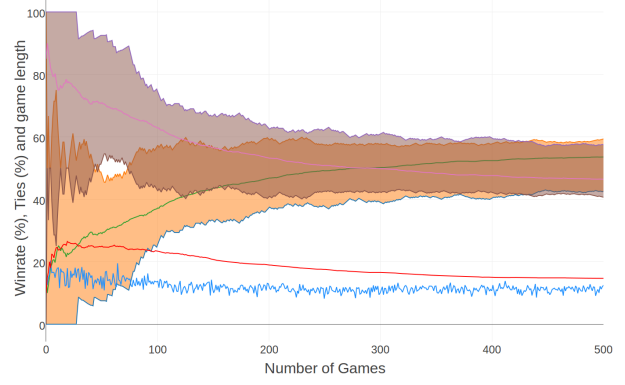
(g) Final agent vs Bot16



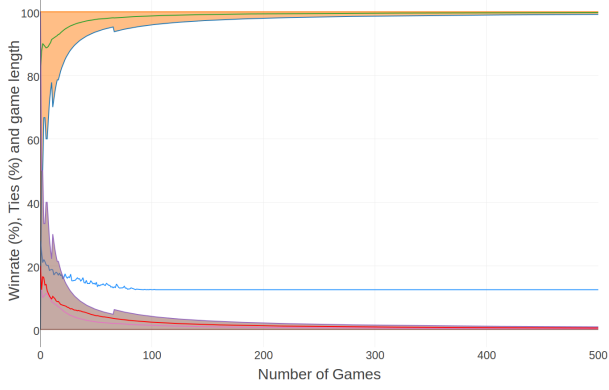
(h) Final agent trained vs Bot16



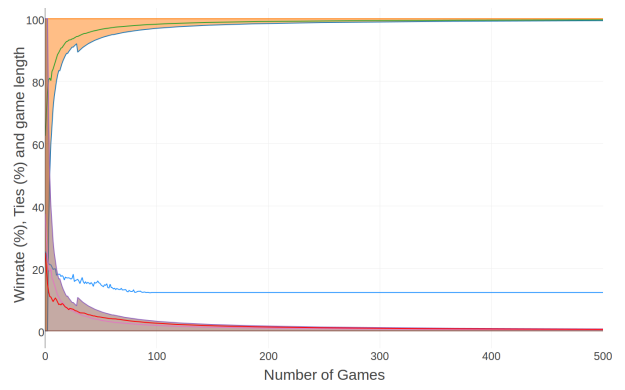
(a) Final agent vs Bot17



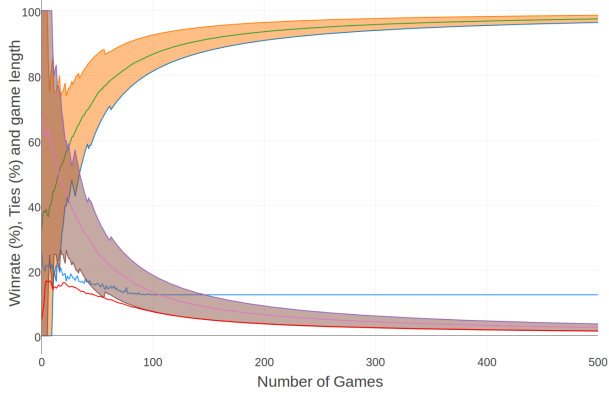
(b) Final agent trained vs Bot17



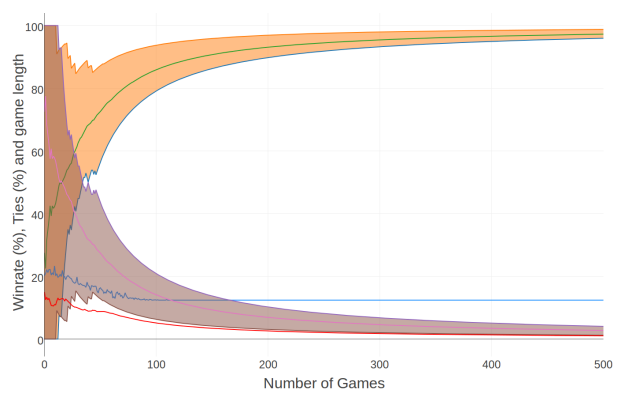
(c) Final agent vs Bot18



(d) Final agent trained vs Bot18

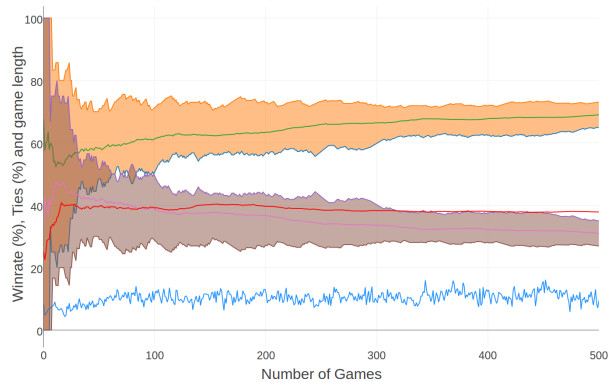


(e) Final agent vs Bot19

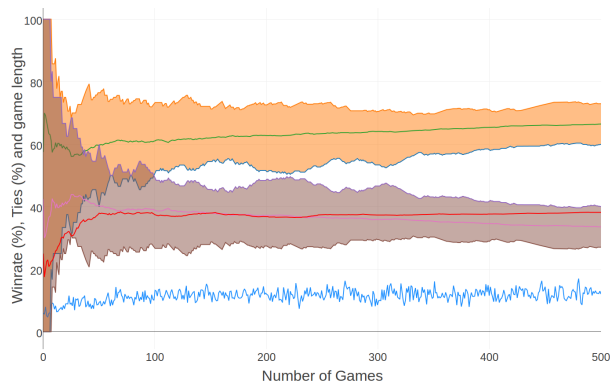


(f) Final agent trained vs Bot19

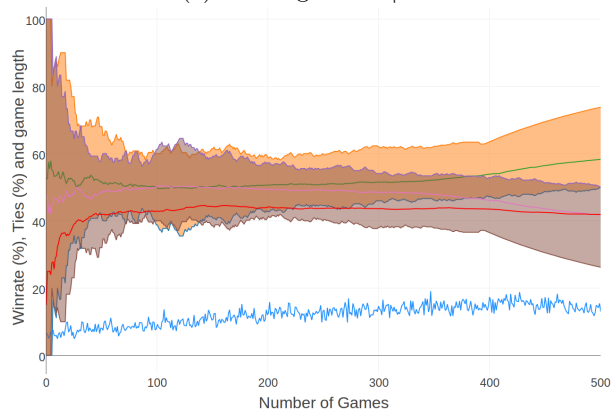
14.6 Graphs associated with tables 12.2 and 12.4



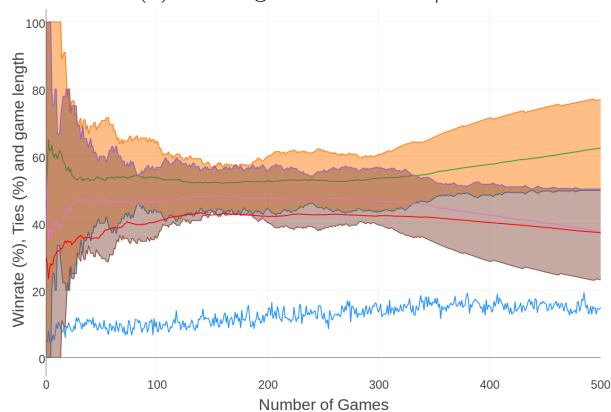
(a) Final agent vs s|s-36



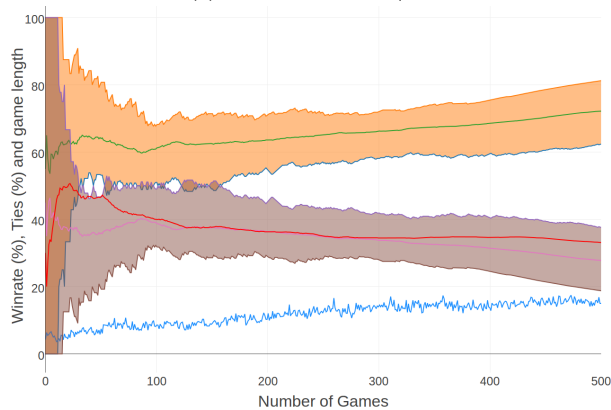
(b) Final agent trained vs s|s-36



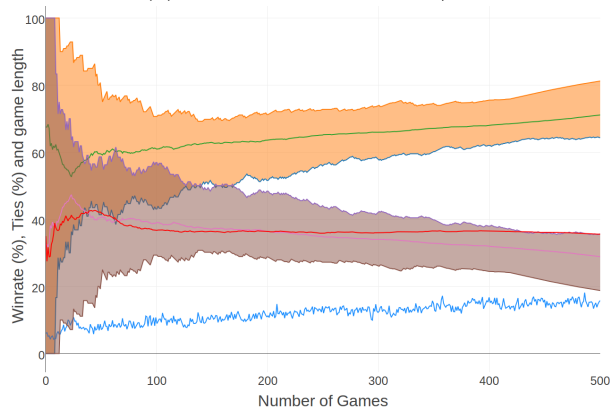
(c) Final agent vs s|s-6



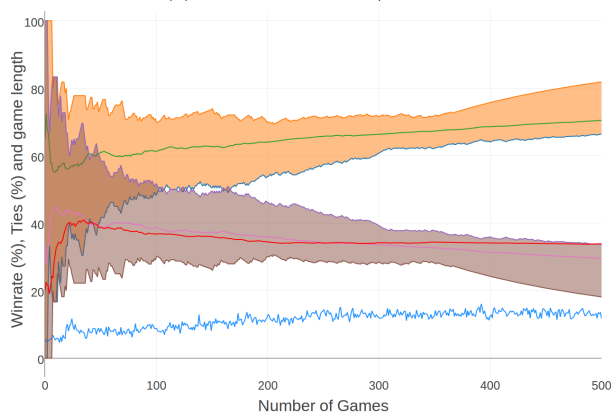
(d) Final agent trained vs s|s-6



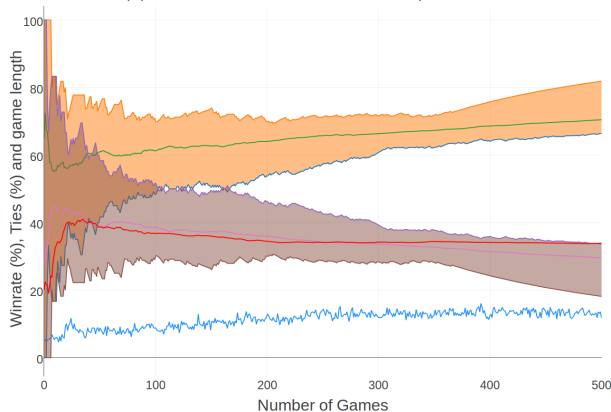
(e) Final agent vs p|p-6-pl5



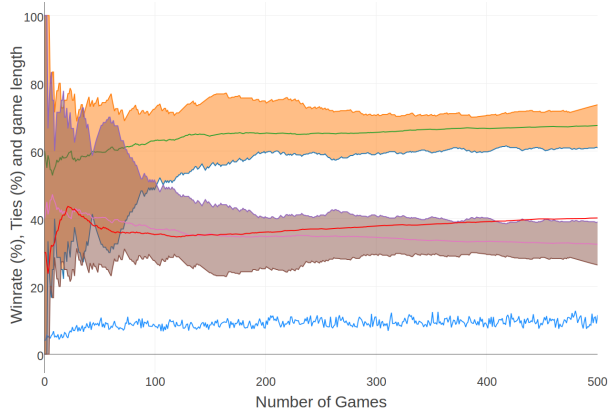
(f) Final agent trained vs p|p-6-pl5



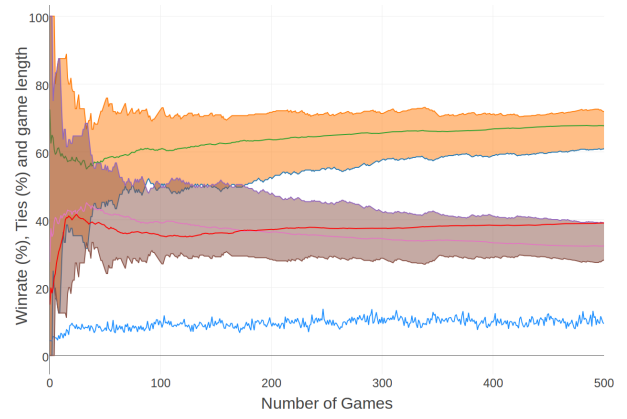
(g) Final agent vs p|p-6-pl99



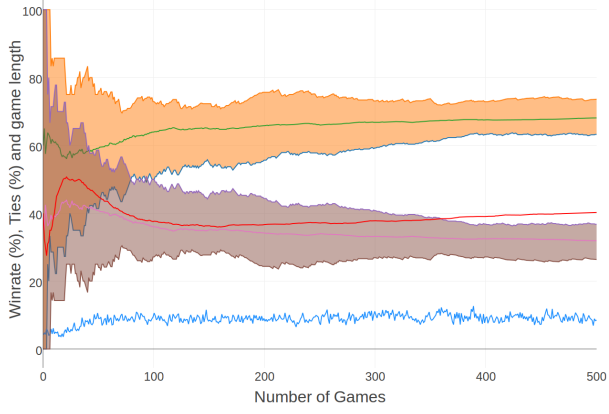
(h) Final agent trained vs p|p-6-pl99



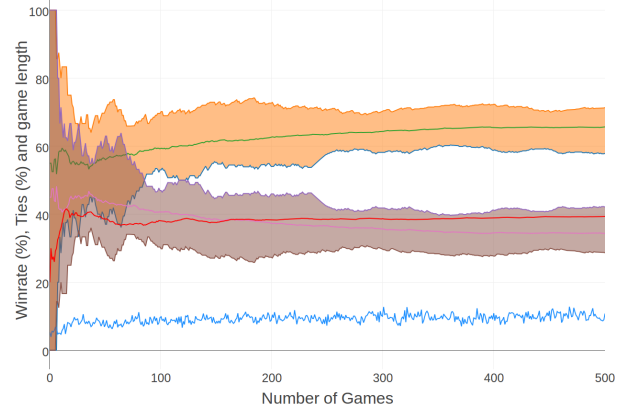
(a) Final agent vs p|p-36-pl5



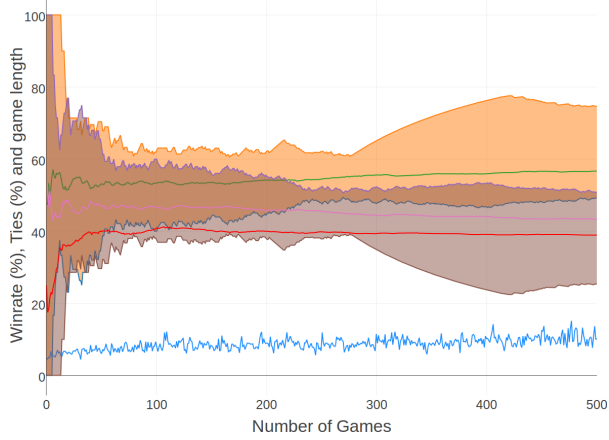
(b) Final agent trained vs p|p-36-pl5



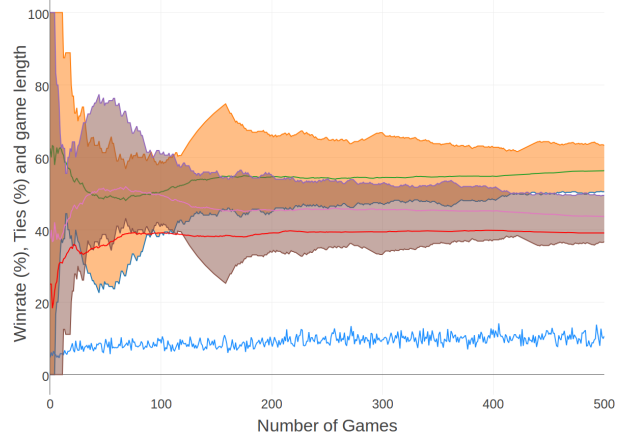
(c) Final agent vs p|p-36-pl99



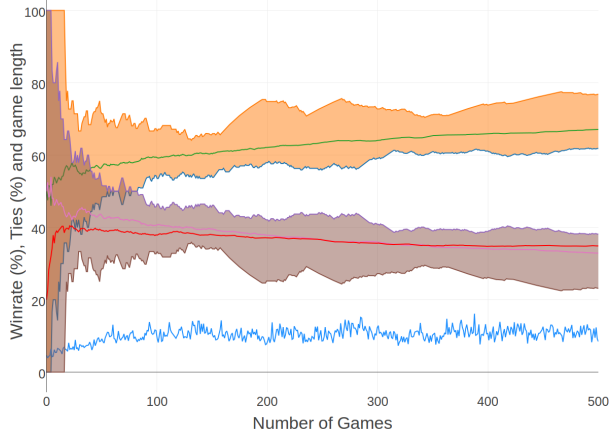
(d) Final agent trained vs p|p-36-pl99



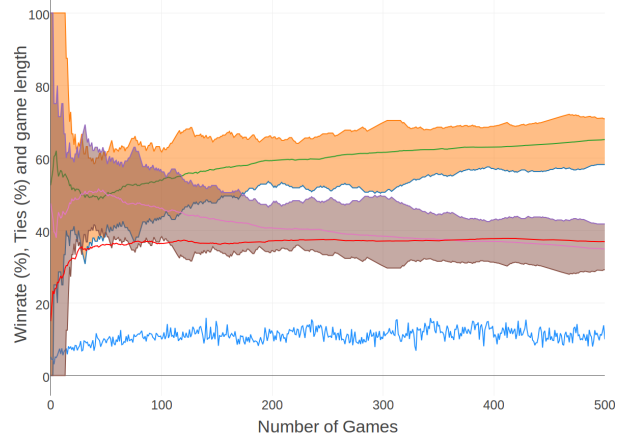
(e) Final agent vs GBR



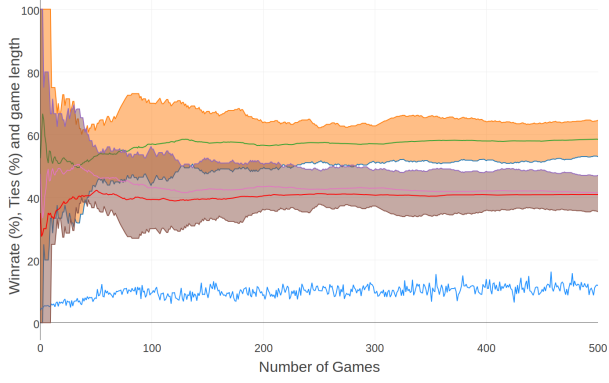
(f) Final agent trained vs GBR



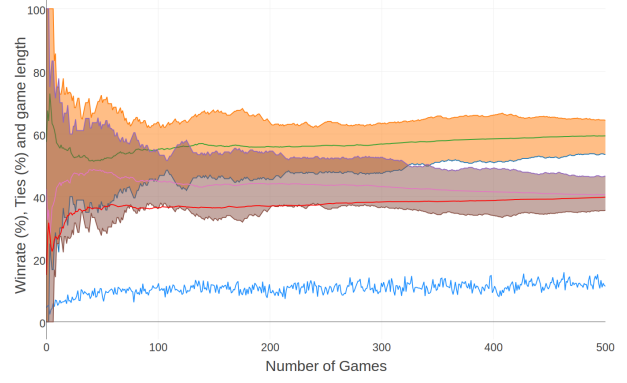
(g) Final agent vs GSR



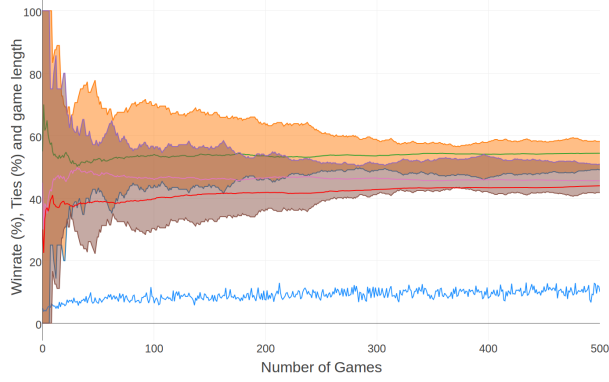
(h) Final agent trained vs GSR



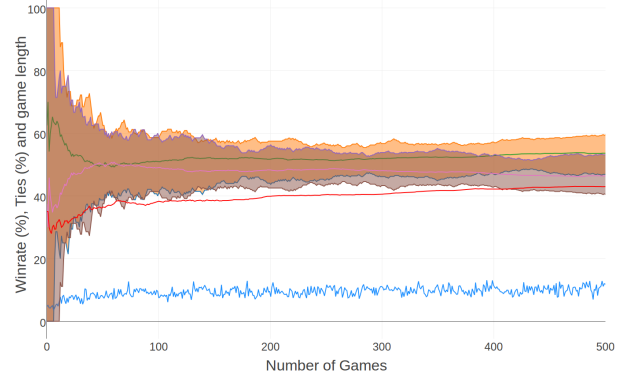
(a) Final agent vs B-T01



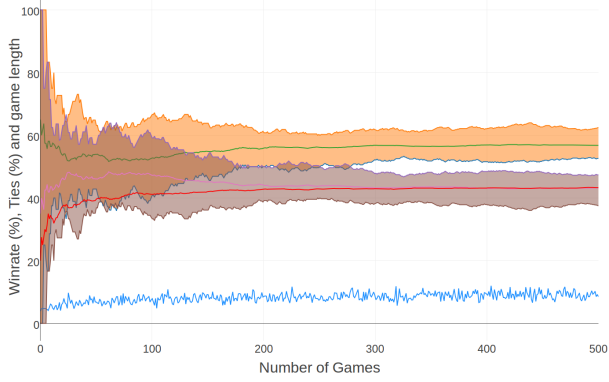
(b) Final agent trained vs B-T01



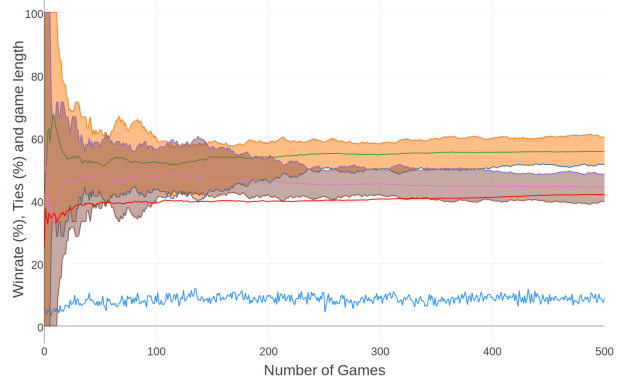
(c) Final agent vs B-T05



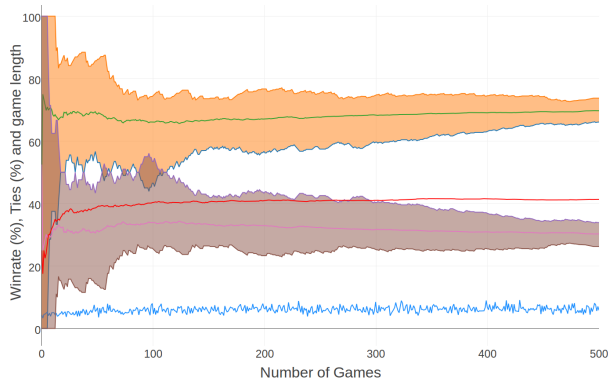
(d) Final agent trained vs B-T05



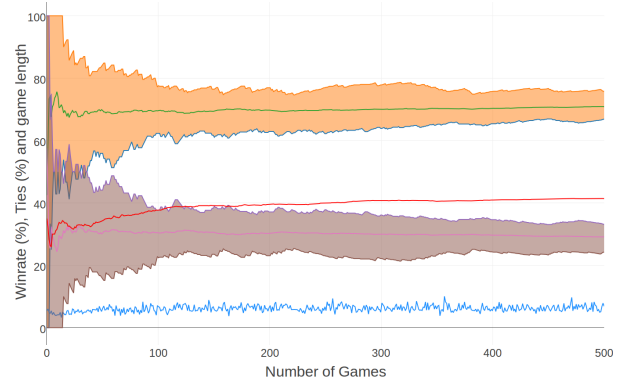
(e) Final agent vs B-T1



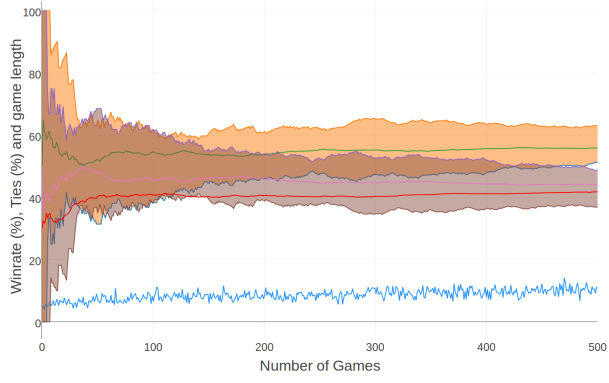
(f) Final agent trained vs B-T1



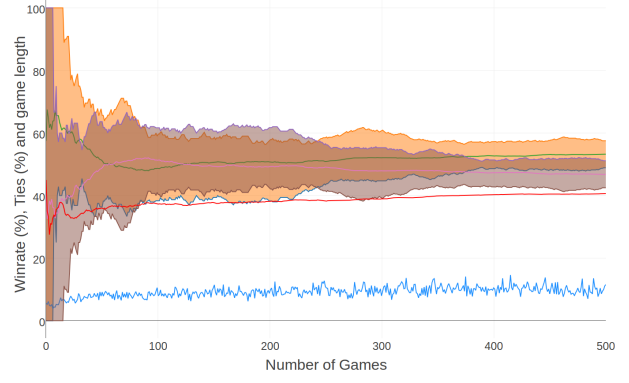
(g) Final agent vs B-T5



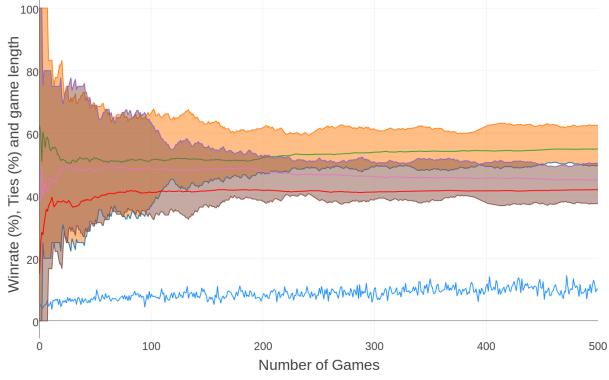
(h) Final agent trained vs B-T5



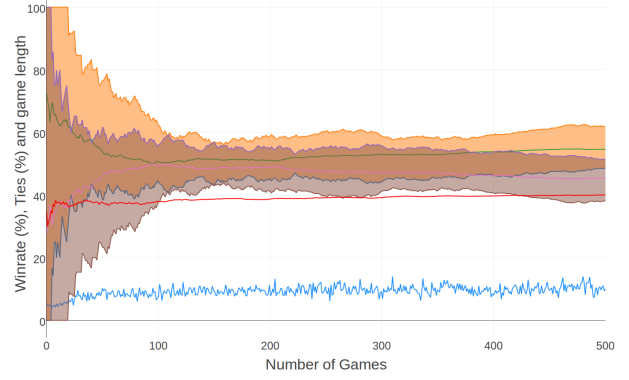
(a) Final agent vs BR-DT-5



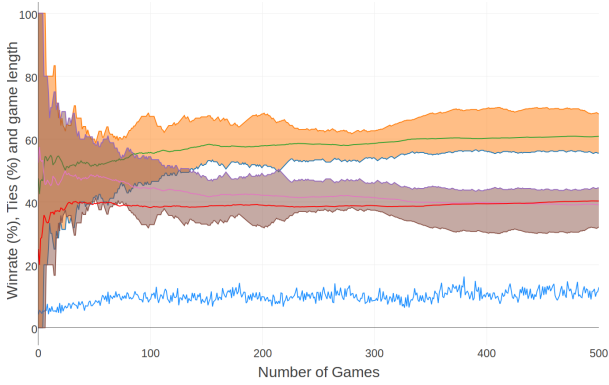
(b) Final agent trained vs BR-DT-5



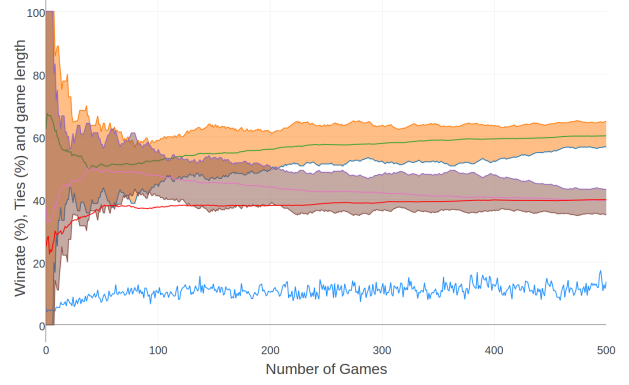
(c) Final agent vs BR-DT-10



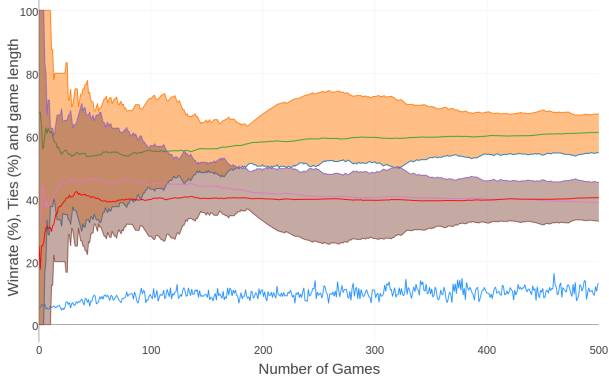
(d) Final agent trained vs BR-DT-10



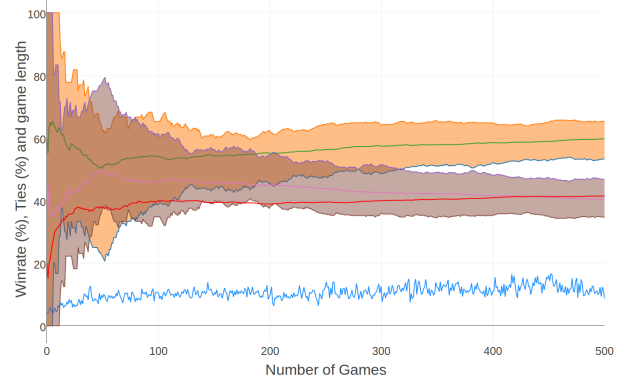
(e) Final agent vs DT-5



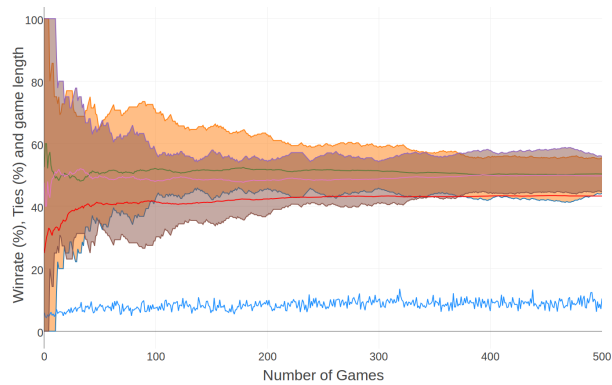
(f) Final agent trained vs DT-5



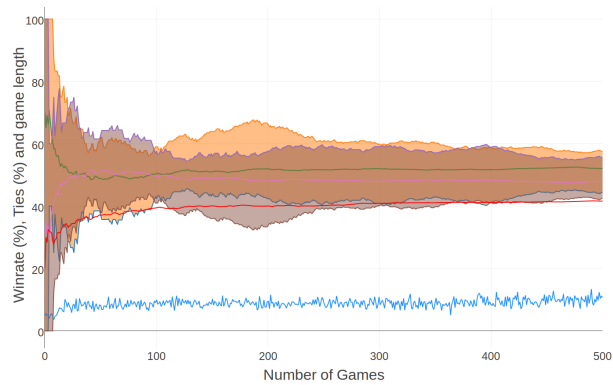
(g) Final agent vs DT-10



(h) Final agent trained vs DT-10



(a) Final agent vs B-T05-Dag



(b) Final agent trained vs B-T05-Dag

