國 立 交 通 大 學

碩 士 論 文

A study of Monte Carlo Methods for Phantom Go

研 究 生：卜賽德

指導教授：吳毅成　教授

資 訊 科 學 與 工 程 研 究 所

中 華 民 國 102 年 9 月

A study of Monte Carlo Methods for Phantom Go

研 究 生：卜賽德

指導教授：吳毅成

Student：Buron Cedric

Advisor：I-Chen Wu

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所
碩 士 論 文

A Dissertation
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master of Science
in

Computer Science

September 2013

Hsinchu, Taiwan, Republic of China

中 華 民 國 102 年 9 月

A study of Monte Carlo Methods for Phantom Go

研究生：卜賽德　　　　　　　　　　　　指導教授：吳毅成 博士

國立交通大學資訊科學與工程研究所博士班

摘要

本論文通過研究 imperfect information games 和改進版 Monte Carlo 方法來建立一個有效的遊戲應用。遊戲是測試人工智慧的重要領域。當今，最有效的方法是 Monte Carlo 法。這些方法是基於概率，並被廣泛用於創建遊戲序，特別是對於圍棋，也 imperfect information games，比如橋牌，撲克牌或幻影圍棋。幻象遊戲根據 Perfect Information 創建，但每個玩家只能看到自己的棋牌。

Imperfect information 博弈是相當難以處理。由於玩家無法知道遊戲的狀態，因此非常困難使用 Minimax 演算法，或找到一個 Nash equilibrium。啟用特定的 Monte Carlo 方法可以把上述問題處理的很好。到現在為止，最好的電子幻影圍棋遊戲是 Flat Monte Carlo，Cazenave 在 2006 年編寫。我們發展另一種方法 two-level Monte Carlo method，並作分析比較。

A study of Monte Carlo Methods for Phantom Go

Student：Buron Cedric Advisor：Dr. I-Chen Wu

Institute of Computer Science and Engineering
National Chiao Tung University

Abstract

This thesis deals with imperfect information games and the application of Monte Carlo methods to build an effective playing program. Games are an important field for testing Artificial Intelligence. Nowadays, the most efficient methods are Monte Carlo ones. These methods are based on probabilities, and have been widely used to create playing programs, particularly for the game of go, but also for imperfect information games, as Bridge, Poker or Phantom Go; phantom games are created according to a Perfect Information game, but in which each player only sees his own moves.

Imperfect information games are quite hard to handle. As the different state of the game is unknown to the players, it is very difficult to use Minimax algorithms, and also to find a Nash equilibrium. Specific Monte Carlo methods enabled to get good playing programs in these games. However, till now, the best playing program for Phantom Go was a Flat Monte Carlo one, written in 2006 by Cazenave. As new methods have been found since then, we also tried a two-level variant of Monte Carlo, which would enable to take in consideration what does or does not know each player during the playout.

致謝

首先，我要感謝我的教授吳毅成先生。他在我編撰這篇論文時為我提供了很大的幫助。沒有吳毅成教授的耐心和專業的指導就不會有這項成果。我在此對吳毅成教授表示敬佩。在我必須回到法國的時候，吳毅成教授為我聯繫了巴黎第九大學 Cazenave 教授。我再次對吳毅成教授表示無限感激。

我感謝在法國接待我並在實驗中指導我幫助了我的巴黎第九大學教授 Tristan Cazenave。Cazenave 教授為我提供了實驗的基礎和環境。同時感謝 Abdallah Saffidine 博士，他給了許多寶貴的建議也在實驗中解答了我的疑惑。也感謝巴黎第九大學 Miguel Couceiro 教授和巴黎第八大學 Nicolas Jouandeau 教授。我很榮幸能有機會和你們合作。

感謝台灣交通大學電子遊戲實驗室和法國 LAMSADE 實驗室的所有同學。和你們度過了愉快的時光。這段時間裡我們在研究上和精神上互相支持。

同時感謝台灣交通大學行政處 CS 和 EECS 的所有工作人員，特別是李倖禎女士和游雅玲女士 。如果沒有你們的幫助，作為國際交流學生的我會遇到很多困難。

最后我要感謝我的愛人侯驊真女士和我的家人，感謝他們在我六個月異國的學習過程和六個月的實驗過程中的精神支持和經濟支持。

Acknowledgements

First of all, I want to thank Professor Wu I-Chen . He provided me great help during the preparation of this thesis. I want to thank him and express my respect to him for his patient and professional guidance, without which nothing would have been possible. When I have had to get back to France, Professor Wu I-Chen contacted professor Cazenave, from the University of Paris Dauphine. For that also, I want to express my immense gratitude to him.

I want to thank Professor Tristan Cazenave, who welcomed me in France and helped and guided me in my research at the University Paris-Dauphine. He provided me with a basis for my work and research. I would also like to thank Dr. Abdallah Saffidine who gave me many valuable suggestions about theory and technique and answered my doubts. I am grateful to Professor Miguel Couceiro, from the University of Paris-Dauphine University and Professor Nicolas Jouandeau from the University Paris VIII. I am honored to have had the opportunity to cooperate with them.

I give thanks to my fellows in the lab in Chiao Tung University in Taiwan and to the Department of LAMSADE in France. If I have had a pleasant time, it is thanks to them. They gave me spiritual support during my studies. I am very grateful to the staff of NCTU Administration Office of CS and EECS, especially Ms Li Xingzhen and Ms Yu Aileen. As an international student, I would have encountered many difficulties without their help.

Finally, I want to thank my companion Miss Hou Huazhen and my relatives. I thank them for their support during these six months abroad and the their moral and financial support during this thesis.

Contents

List of Figures

List of Tables

List of Definitions

Chapter 1 Introduction

There are mainly two ways to deal with games. The first one is to try to solve the game, *i.e.*, either find a strategy which leads to win the game or prove that it is impossible to do so. Some games, as tic-tac-toe, or 8×8 Hex are solved, but other ones, as Chess, are not solved. This does not mean that computers cannot beat humans.

The second way of dealing with games is to build efficient programs, able to beat other programs and humans. When it is not possible to solve games, often because of their length, researchers try to build an artificial intelligence (AI) for them. This way of dealing with games is very interesting because it is more likely to be applied in real life. Indeed, the real situations are often too big, too complicated to be “solved”.

But even beating humans can be too complicated for computers. The game of Go is a very ancient game, mainly played in Asian countries, as Taiwan, China, Japan and Korea. More than that, it is a real challenge for Computer Science researcher, as the best human players can still beat the best programs on 19×19 boards, because there are too many possibilities to play in an efficient way. This game, as Chess or Hex, is a perfect information game, as both players know everything which is on the board. In this thesis, we deal with another problem than the size of the game, imperfect information Games (IIG).

In these games, the player does not have all information about the state of the game. Notice that these games are different from the incomplete information games (in which players miss information on the type of the other players) and the games with stochasticity (intervention of chance). These games are very interesting because they are closer to real situations, in which we don't know which are the actions and reactions of the environment and/or other people.

Methods to build playing programs for these games are quite recent, because it is quite hard to handle the issues of the unknown information. Actually, until some improvements on Monte Carlo methods, it was very hard to build a really good playing program for them. More classical methods have been tried, but they were quite few successful ones. Recently, this kind of games has been studied with more success, in particular card games as Poker, Bridge and Skat. We decided to follow these new methods, even if the card games are quite far from board games, as Go. After the introduction (Chapter 1) we will clarify the framework in Chapter 2: we need in game theory, we will present the game we have worked on, Phantom Go. In the Chapter 3, we will have a look on the different Monte Carlo methods and improvements we have used. The Chapter 4 will be dedicated to a new method we have worked on, the two-level Monte Carlo search. We will show our results in the Chapter 5 and then give a conclusion and some direction for upcoming works.

Chapter 2 Imperfect information games

While the Baroque rules of chess could only have been created by humans, the rules of go are so elegant, organic, and rigorously logical that if intelligent life forms exist elsewhere in the universe, they almost certainly play go.

Edward LARKER

In this chapter, we give some elements on games with imperfect information, also called games of imperfect information, or Imperfect Information Games (IIG). To have a good idea of what is precisely a imperfect information game, we will firstly introduce some elements of game theory in general. We will then give more elements on IIGs, and finally make a quick review of the different methods used to deal with such games.

2.1 Elements of game theory

2.1.1 Definition of a game

The formal definition of a game has been given in [32]. It is defined as follows:

Definition 2.1.1. Game

A **game** is defined by three sets: The set of **players**, $\mathcal{N}$, the set of **strategies**, or policies $\mathcal{S}$ and the set of **payoffs**, $\mathcal{X}$.

In this thesis, we will only consider a particular class of games, the two player zero-sum games, which are also defined in [32]:

Definition 2.1.2. Two player zero-sum game

Let $\mathcal{G} = (\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{X}})$ be a game.

$\mathcal{G}$ is two player zero-sum game if and only if $|\mathcal{N}| = 2$, and

$\forall s = (x_0, x_1) \in \boldsymbol{\mathcal{X}},\ x_0 + x_1 = 0.$

Notes about these definitions:

- In classical game theory, the players are supposed to know the rules and the structure of the game: they only ignore their opponent's strategy. In imperfect information games, this is not the case: at least one player knows some elements his opponent ignores.
- We assume that each player tries to maximize his score, but in two player zero-sum games, this is equivalent to minimize the score of the opponent. In algorithms such as Minimax, one player tries to maximize his score (we call it "Max player"), while the other one tries to minimize it (we call it "Min player").
- The strategy defines what a player is going to do. In sequential games, this can be expressed as "if the previous player plays A, I play B, if he plays C I play D and if he plays E I play F". The players are expected to choose a strategy to maximize their payoff.
- The payoff is a $|\mathcal{N}|$-tuple defining for each terminal position the score of each player.

We also need some other definitions, from [30]:

Definition 2.1.3. Strategy profile and outcome function

A **strategy profile** is a set of strategies for each player. It is defined for one and only one strategy for each player.

The **outcome function** of a game take a strategy profile S and give an **outcome** $y = Y(S) = (y_1, y_2, \ldots, y_{|\mathcal{N}|})$.

To represent games, we use three different forms: normal form, extensive form and characteristic form. For most of the games, the normal and extensive forms are the more convenient ones. The characteristic form is mainly used for cooperative games, which are quite far from our field of interest. We are therefore going to introduce the two other forms (extensive and normal). The definitions can be found in [30]

2.1.1.1. The normal form

Games in this form are described by a matrix. In each block of the matrix are represented the wins of each players according to the strategy they chose. Let take a simple example.

EXAMPLE 2.1.1. The game of rock-paper-scissors

The game of Rock-paper-scissors is a two-player game. Each player secretly chooses rock, paper or scissors. Once both have chosen, they reveal their choice. If both players made the same choice, it is a drawn. If one player chooses rock (resp. paper, scissors) and the other one scissors (resp. rock, paper) then the first player wins. We associate the value 1 to "win", the value -1 to "loss" and 0 to a drawn. By using the normal form to describe this game, we get the following matrix:

	Rock	Paper	Scissors
Rock	$(0,0)$	$(1,-1)$	$(-1,1)$
Paper	$(1,-1)$	$(0,0)$	$(-1,1)$
Scissors	$(-1,1)$	$(1,-1)$	$(0,0)$

In this example, the players are represented in the first column and the first line. The possible strategies are "Rock", "Paper", "Scissors". The payoffs are the tuples in the table.

The matrix is slightly different for a sequential game, and will not be a square one. Let take the example of tic-tac-toe.

EXAMPLE 2.1.2: THE GAME OF TIC-TAC-TOE

Tic-tac-toe is a two player game, red and blue. It is played on a 3×3 grid. The two players alternatively fill one spaces of the grid, and try to get 3 of their symbols aligned in a row, a column or diagonally. We there get interested the endgame. There are only three empty spaces (cf. Figure 2.1)

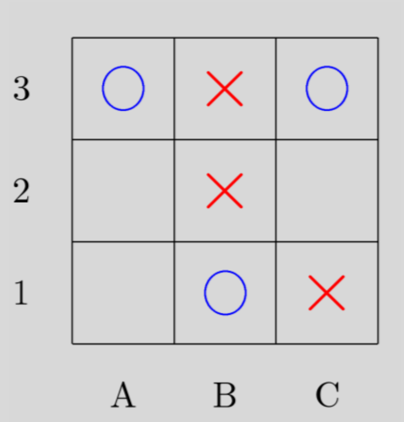


Figure 2.1. Tic-tac-toe endgame

In this configuration, assuming that Blue is Min player and plays before Red (Max player), a possible strategy for Blue is “play $A1$; if Red plays $A2$, then play $C2$, else play $A2$”. A possible strategy for Red is “if Blue plays $A1$ play $A2$, if Blue plays $A2$, play $A1$, if Blue plays $C2$, play $A1$”. We will write this Blue strategy $(A1\, C2; A2)$ and the Red one$(A2, A1, A1)$. Then, the normal form of this game is as represented on the next page.

As we can see, this representation is kind of redundant. Let consider for instance the first four lines of the first column. They all correspond to the same move for both players. As they are in the same column, they correspond to the same strategy for Blue which will choose A1, but also for Red: as Blue chooses A1, it means that Red will choose in any of these situations A2, and then Blue will play on$C2$. This representation is therefore very useful to compare the different payoffs and strategies, but it is often more convenient to represent the games under the form of a tree, the Extensive form.

EXAMPLE 2.1.2. The game of tic-tac-toe (continuation)

	$(A1\ C2, A2)$	$(A2\ C2, A1)$	$(C2\ A2, A1)$
$(A2, A1, A1)$	$(0,0)$	$(-1,1)$	$(-1,1)$
$(A2, A1, A2)$	$(0,0)$	$(-1,1)$	$(0,0)$
$(A2, C2, A1)$	$(0,0)$	$(0,0)$	$(-1,1)$
$(A2, C2, A2)$	$(0,0)$	$(0,0)$	$(0,0)$
$(C2, A1, A1)$	$(0,0)$	$(-1,1)$	$(-1,1)$
$(C2, A1, A2)$	$(0,0)$	$(-1,1)$	$(0,0)$
$(C2, A2, A1)$	$(0,0)$	$(0,0)$	$(-1,1)$
$(C2, A2, A2)$	$(0,0)$	$(0,0)$	$(0,0)$

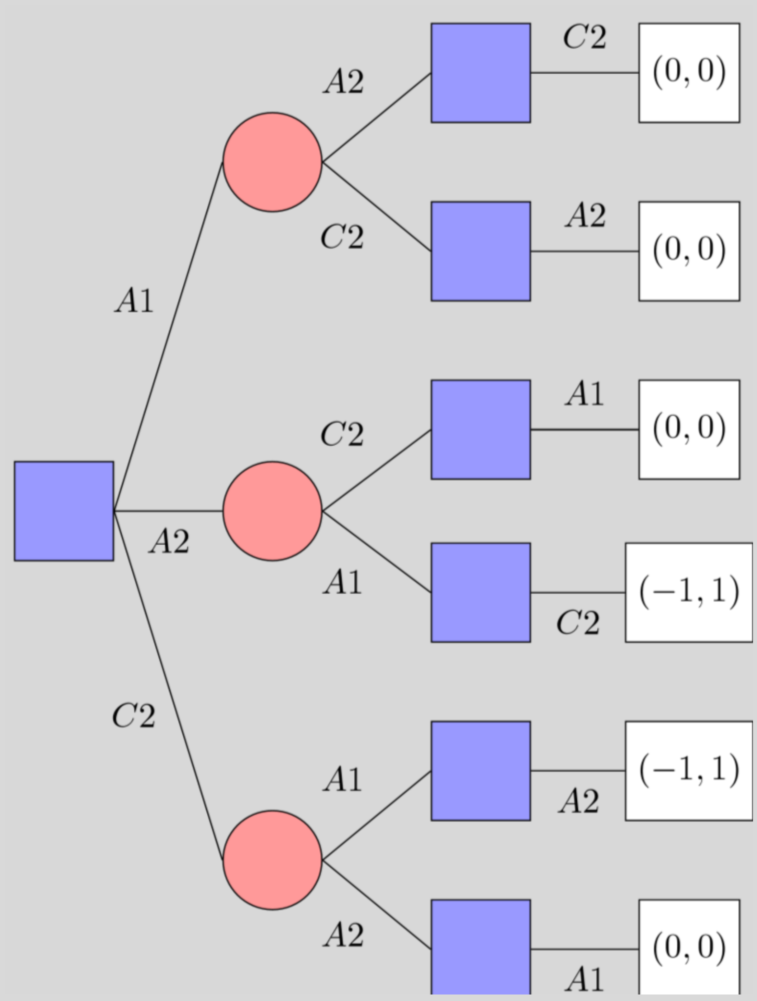
2.1.1.2. The extensive form

In this form, games are represented as trees. Each node corresponds to a state of the game, in which it is either the turn of Max player, either the one of Min player. The edges correspond to a given move. A terminal state of the game is represented by a leaf. We write the payoff behind the leaves. A total game corresponds to a path from the root of the game till a leaf.

Extensive form games are much better to represent sequential games than simultaneous games. Let take the same example as before, the tic-tac-toe endgame, on Figure 2.1.

Then, the extensive form of this game is:

EXAMPLE 2.1.3: THE GAME OF TIC-TAC-TOE (EXTENSIVE FORM)



We can now give a definition for a (sequential) subgame:

Definition 2.1.4. Subgame

A subgame of a game $(\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{X}})$ is a game $(\boldsymbol{\mathcal{N}}' = \boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}}', \boldsymbol{\mathcal{X}}' \subsetneq \boldsymbol{\mathcal{X}})$, for which:

1. It contains all the nodes that are successors of the initial node.

2. It contains all the nodes that are successors of any node it contains.

Now, let consider a simultaneous game. It is not possible to represent the game strictly as we represented tic-tac-toe, as the moves are simultaneous. To avoid the problem we represent the game another way: an information set is represented by a dashed line linking the different possible states of the set. Notice that we can also represent imperfect information game the same way. Now, let take the same example as we took for normal form: Rock-paper-scissors. On the **EXAMPLE** 2.1.4, we clearly see the information set for the Min player (blue): as he does not know what Red chose, the three nodes are possible.

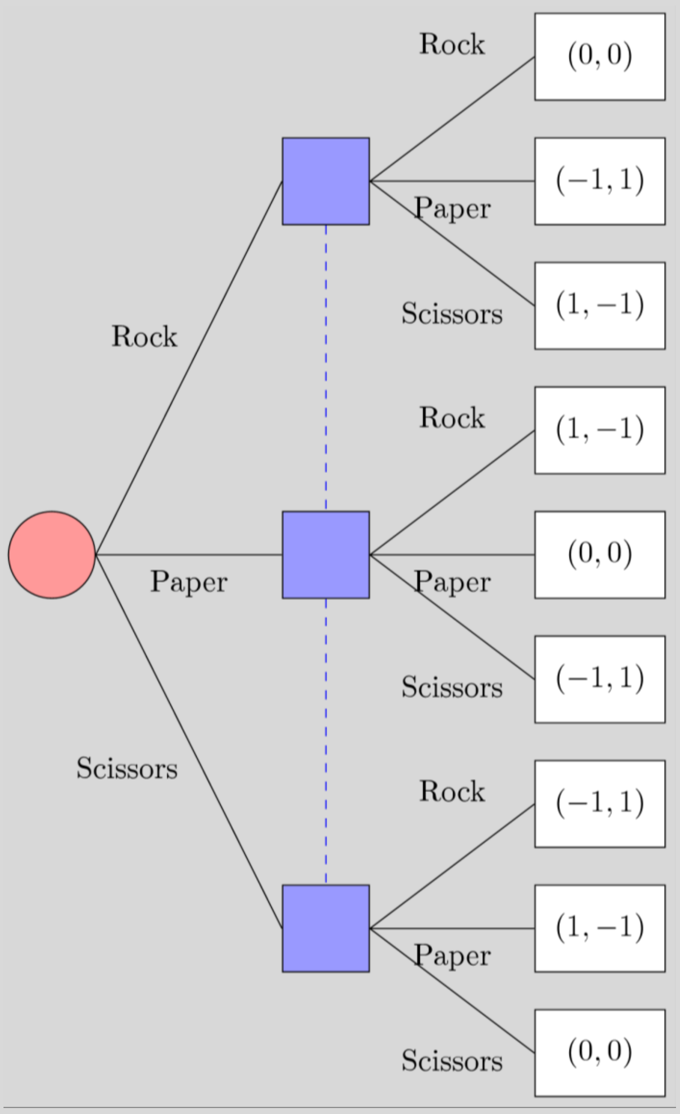
We can now give a new definition of what a subgame, from [30]:

Definition 2.1.5. Subgame; extensive form

A subgame of a game $(\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{X}})$ is a game $(\boldsymbol{\mathcal{N}}' = \boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}}', \boldsymbol{\mathcal{X}}' \subsetneq \boldsymbol{\mathcal{X}})$, for which:

1. The initial node is in a singleton information set.
2. It contains all the nodes that are successors of the initial node.
3. It contains all the nodes that are successors of any node it contains.
4. If a node is in the subgame then all members of its information set belong to the subgame.

EXAMPLE 2.1.4. ROCK-PAPER-SCISSORS: EXTENSIVE FORM



2.1.2 Nash equilibrium

2.1.2.1. Nash equilibrium for sequential games

Instinctively, a Nash equilibrium (which has been described in [29]) is a situation for which any player cannot change his strategy without decreasing his payoff.

Definition 2.1.6. Nash equilibrium

Let $(\boldsymbol{\mathcal{N}}, \boldsymbol{S}, \boldsymbol{\mathcal{X}})$ be a game, and $\boldsymbol{S} = (\boldsymbol{s_1}, \boldsymbol{s_2}, \dots, \boldsymbol{s}_{|\boldsymbol{\mathcal{N}}|}$ a strategy profile; let $f = (f_1, f_2, \dots, f_{|\mathcal{N}|}$ a function such that for a strategy profile S, $f_i(S) = X(Y(S))$. $S^* = (s_1^*, s_2^*, \dots, s_{|\mathcal{N}|}^*)$ is a **Nash equilibrium** if and only if: $\forall i: 0 \leq i \leq |\mathcal{N}|, \forall S$, strategy profile of the *i-th* player, $f(s_1^*, s_2^*, \dots, s_{i-1}^*, s_i^*, s_{i+1}^*, \dots, s_{|\mathcal{N}|}^*) \geq f(s_1^*, s_2^*, \dots, s_{i-1}^*, s_i, s_{i+1}^*, \dots, s_{|\mathcal{N}|}^*)$.

There is at least one Nash equilibrium for any two-player zero-sum sequential game. But in imperfect information games and simultaneous games, the Nash equilibrium cannot be a “pure” strategy profile.

2.1.2.2. Nash equilibrium for simultaneous games, pure and mixed strategies

To define what is a Nash equilibrium for simultaneous games, we have to define the notion of mixed strategy.

A **pure strategy** is a strategy describing deterministically what to do while a **mixed** strategy is a set of strategies associated to probabilities.

We now get the following result, from [29]:

There is at least one (possibly mixed) Nash equilibrium for any game of perfect information.

For instance, a Nash equilibrium of the Tic-tac-toe subgame in EXAMPLE 2.1.1 is $(\text{A1 C2}, \text{A2}), (\text{A2}, \text{A1}, \text{A1})$. The only Nash equilibrium for Rock-Paper-Scissor EXAMPLE 2.1.1 is ($\frac{1}{3}$ rock, $\frac{1}{3}$ paper, $\frac{1}{3}$ scissors) for each player (it is a symmetric game), because playing differently would lead the adversary to get a counter-strategy more efficient.

2.2 Imperfect information games

Now, let consider a new concept: games of incomplete information. What is a game of incomplete information? Instinctively, it is a game for which each player has not the same information. There are many examples of imperfect information games, from card games to the point of this thesis: phantom games.

In this section, we firstly define what a game with incomplete information is. Then, we give an example of such a game, which is also the point of this thesis: Phantom Go.

Incomplete information are different from imperfect information games.

Definition 2.2.1. Imperfect information game

A **game of imperfect information** is a game for which one player at least does not know the complete history of the game at least at one moment of the game.

Let give an example of such a game. A simultaneous game can be seen as a sequential game with imperfect information. We can take the game of rock-paper-scissors (EXAMPLE 2.1.4) and transform it into this equivalent form: player 1 secretly chooses rock, paper or scissors, and player 2 chooses then his item. This game is exactly equivalent to the original game. Its equilibrium is strictly the same as the one of the original game ($\frac{1}{3}$ rock, $\frac{1}{3}$ paper, $\frac{1}{3}$ scissors). In this game, player 2 does not know what move player 1 chose before playing himself. Notice that he knows what are the strategies available to player 1, his payoffs, and what information player 1 knows about him, which is not the case in incomplete information games.

Instinctively, incomplete information games correspond to games for which the information is not the same for both players about the rules or the structure of the game. Harsanyi in [21] gives a more precise definition of what an incomplete information game is:

Definition 2.2.2. Incomplete information game

A game is said with **incomplete information** if and only if at some moment of the game:

1. the strategies available to at least one player (his strategy space) are not known to every player,
2. the physical outcome of the game is not known to every player,

or

3. the utility function of at least one player are not known to every player

An example of such a game would Mafia. In this game, created in 1986 (see for instance [5]). In this game, each player plays a role, either gangster (a small number of players) or residents (notice that we do not take the third class of characters, detectives in consideration because we don’t need it for our explanation). The residents do not know who the gangsters are. The game has two phases. During the first one, the gangsters secretly decide to kill one resident. During the second one, the citizens (gangsters and residents) conjointly decide to kill one citizen. The residents will try to find a gangster, while the gangsters (without revealing themselves) will try to convince the other citizens to kill another player. As the residents do not know which players are gangsters, they can be duped. In this game, some players, the residents, do not know the type of the other ones. This game is an incomplete information game.

In [21, 19, 20], Harsanyi showed that imperfect and incomplete information games are equivalent, and that it is practically possible to transform incomplete information games into imperfect information ones.

From the 50's many imperfect information games have been studied, and in particular phantom games, for which rules are the same as a game of perfect information, but for which each player only knows his own pieces. An example of such a game is Kriegspiel (phantom Chess). It has been studied by LI in [26]

The topic of this thesis is Phantom Go. This game is taken from Go, but the fact that it is an imperfect information game changes quite a lot the way of creating a player for it. We are now going to explain the game.

The Phantom Go is very close to Go. The main rules are the same: Atari, capture, counting the points and territories are exactly the same as in the game of Go.

2.2.1 The rules of the Go game

We first describe the game of Go (for more information, please refer to [31]). The Go game is a board game. The board (or "Goban" in Japanese) is a 9×9, 13×13, or 19×19 intersections board, you can see an example on Figure 2.2. At his turn, each of the two players puts a stone on an intersection. The goal of the game is to create territories and capture the stones of the opponent by surrounding them.

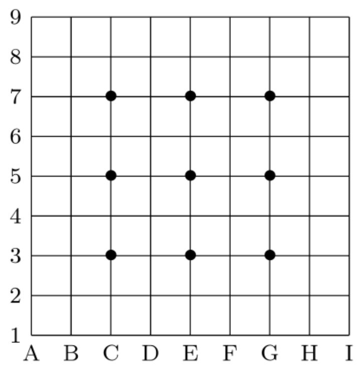


Figure 2.2. A 9×9 goban

2.2.1.1. Capture

A stone is linked to other ones by the lines of the board. If a stone or a group of stones are surrounded by opponent's stones, they are captured (as on Figure 2.3 and Figure 2.4)

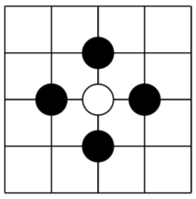


Figure 2.3. The white stone is captured

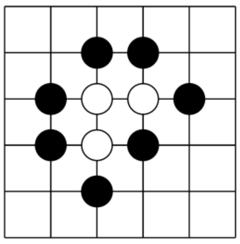


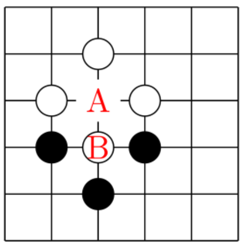
Figure 2.4. The white group is captured

2.2.1.2. Suicide

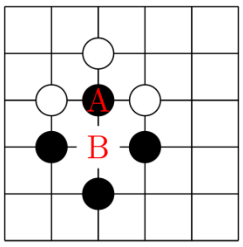
Putting a stone so that a group of your own stones is captured (commit a suicide) is forbidden, except if putting this stone allows you to capture adversary's stones, and thereby avoid the suicide.

2.2.1.3. Ko

The “ko” rule is made to avoid repetitions in the game: The rule forbids coming back immediately to a previous position of the game. Concretely, this means you cannot capture a stone that has just captured you, as shown on Figure 2.5. Nonetheless, you perfectly can do it one move later, if the move is still allowed.



a: Initial situation, black plays on A and captures B



b: Then,white cannot immediately capture A by playing on B, which would lead to the previous situation

Figure 2.5. Illustration of the ko rule

2.2.1.4. Liberty and Atari

The number of empty intersections linked to a group is called «liberty». When a stone or a group of stones has only one liberty *i.e.*, the stones can be captured in the next stroke) the situation is called Atari. An example of this situation is shown on Figure 2.6.

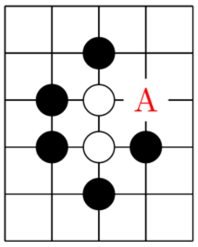


Figure 2.6. The white group is in atari: if Black plays in A, the white stones are captured

2.2.1.5. Living and dead groups and eyes

To be sure that a group of stones is not going to be captured, we constitute what we call "eyes". An eye is an empty intersection surrounded by stones of the same player. If there is only one eye, the rules allow surrounding the group and finishing by putting the last stone on the eye. But a group having two eyes cannot be captured, as the player cannot put a stone simultaneously on the two eyes. A group of stones with two eyes is called alive, a group in a situation which does not allow to create these two eyes is said dead. The Figure 2.7 summarizes these situations.

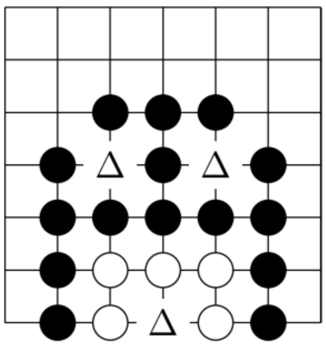


Figure 2.7. The black group is alive, the white one is dead; the eyes are identified by Δ

2.2.1.6. Territories and score

Another side of Go is the constitution of territories. When stones of the same color circle an area, this area is called "territory". The game ends when both players consider there is not anything interesting to do, and pass. At the end of the game, the score is calculated this way: each captured stone gives one point, each empty intersection of the territory gives one point. Each opponent's stone on his territory gives one point.

An example is given on Figure 2.8. This example comes from the British go association. In this game, the stone a is considered as dead, as the player admits there is no way to escape. Moreover, during the game, Black has captured 6 stones, and White 1 stone. The territory of Black is 18 intersections, the territory of White is 19 intersections. Thus, Black has 18+6+1=25 and White 19+1=20. Notice also that, as black begins, there may be some handicaps points for White ("komi"), depending on the size of the board.

There is also another way to count the points (simpler). With the "Chinese rules", each empty intersection in a player's territory gets him one point. Each one of his stone on the board (which is not on the opponent territory) also gets him one point. For instance, the score of the Figure 2.8 would be 43 for Black and 38 for White (plus the komi).

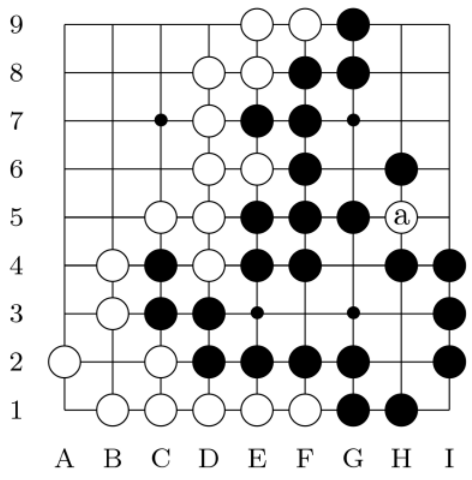


Figure 2.8. End of a game

2.2.2 The Phantom Go

In the Phantom go, there are three different boards: one for each player and one for a third part, the referee. Each player plays on his own board, and the referee plays the strokes of both players.

Each player ignores the stones of his opponent, except when the referee gives him information. The referee indicates that the move is illegal (but he does not indicate if it is because it is an occupied position or a suicide). The judge also indicates the captures. The game is usually played on 9×9 boards.

We give an example of a game and the notations for the game on Figure 2.9. The information `it is impossible to play here' is represented as a square, even if we treat it as an actual stone.

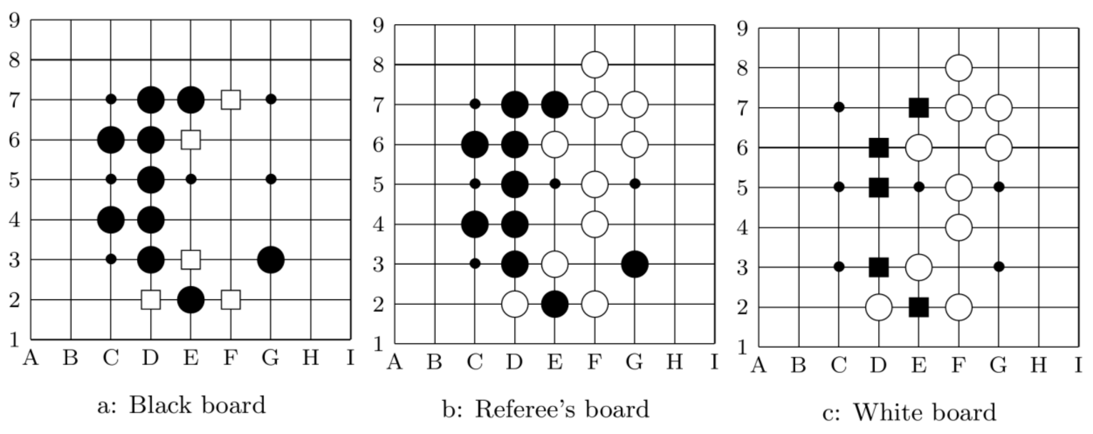


Figure 2.9. A Phantom Go game

Chapter 3 The Monte Carlo Search and its variants

In mathematics, as in physics, so much depends on chance

Stanislaw ULAM

In this chapter, we present the different Monte Carlo methods which have been used to create playing programs for Phantom Go, and those we used to create our own method, the two-level Monte Carlo, which is presented in the next chapter. We will also present classical improvements for Monte Carlo, and which we have used in our programs.

3.1 Flat Monte Carlo search

As we explained, we can deal with games by solving them (for instance explore the whole tree of an extensive form game) and getting the Nash equilibrium (see for instance [23] for that kind of methods), but this is sometimes too long, and we prefer dealing with it by exploring only parts of the tree. This is the idea of the Monte Carlo Search. The Monte Carlo method has been discovered in 1949 in the article [28]. It has been applied to games, particularly Go, because the domain was too wide to use classical minimax methods. In spite of exploring the whole tree of the game, for each child of the current position, we only explore certain path to a final position, and from the results obtained, we approximate the value of this child. The paths are called “playouts”. Monte Carlo method has got many improvements since its creation, as [6] shows. It has then been used on imperfect information games (see for instance [15]). The program of Cazenave, [8] and [10], uses this principle, as shown on Figure 3.1.

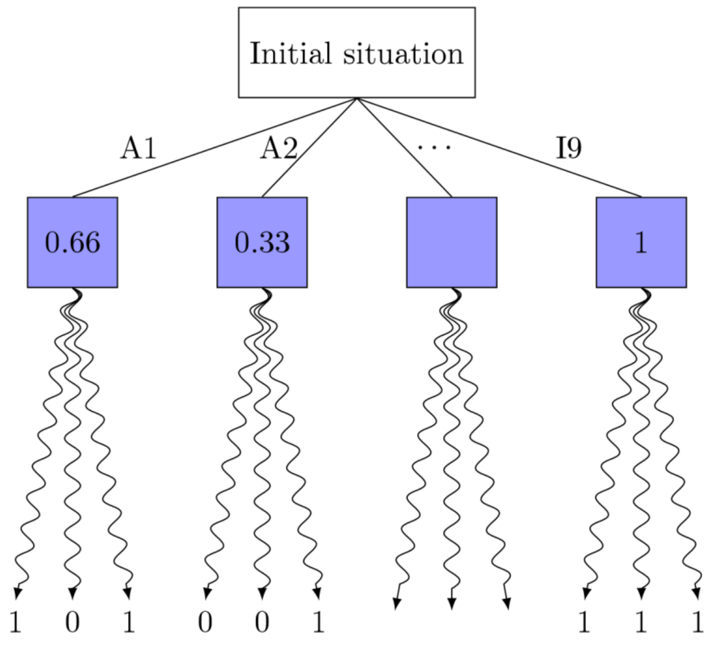


Figure 3.1. Monte Carlo Search for the first move

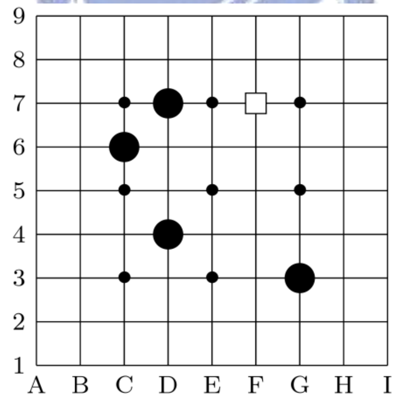
The algorithm is quite simple:

Algorithm 3.1.1. Flat Monte Carlo

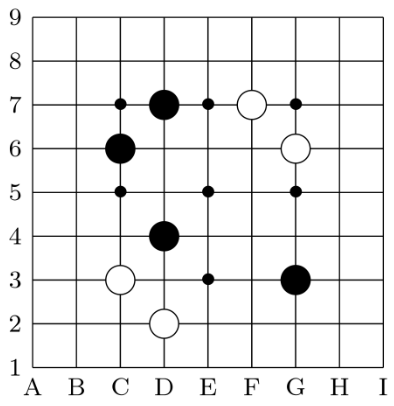
```
Function Flat (InfoSet I, Player P):
    int score[]
    For each move m:
        I' = copy(I)
        play(m, I')
        sample(I', P)
        score[m] = Playout(I',m)
    move = argmax_m(score[m])
    return move
```

Let have a look at how to create a playout. To create a playout, we have to choose a strategy (we usually choose a simple strategy), and use this strategy for every players until getting to a final state of the game. We then look at our payoff, and add it to the average value of the node we are evaluating, as shown on Figure 3.1.

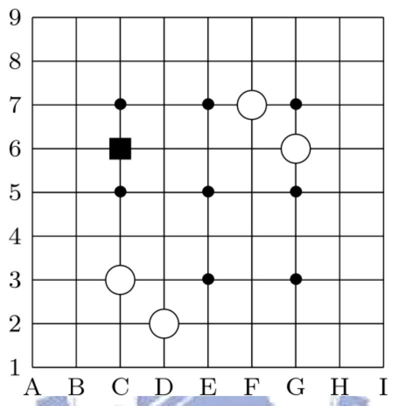
This method (which has been created in 2001 by Ginsberg in [18] for Bridge, under the name of Perfect Information Monte Carlo search, PIMC) gives good results in Phantom Go. It is impossible to make a playout on a situation which is not complete. For instance, we can imagine a trick-based card game, in which it is impossible to know which card belongs to which player. This is why we need to sample the board before making the playouts. Now, to create a playout, we sample a coherent state and then create the playout on this state. To have an example of how this method works, see Figure 3.2 on which we can see a Phantom Go board after 4 moves.



a. Black board



b. Referee's board



c. White board

Figure 3.2. A Phantom Go game

It is the turn of Black, he firstly chooses a move to evaluate, for instance $B2$ and samples the board (Figure 3.3) and then creates some playout according to a policy (for instance playing random legal move, without filling the eyes).We then get a final position, for instance the Figure 3.4. From this position, we get the score of Black, which is, if we use the Chinese rule to count the points 41, and if we consider the game as a zero-sum game, the score of Black will be 1 (and the one of White, 0, if we count 0 for loose and 1 for win). If we repeat this technique several times for each intersection, we get a score. We then choose the best score and play on this intersection.

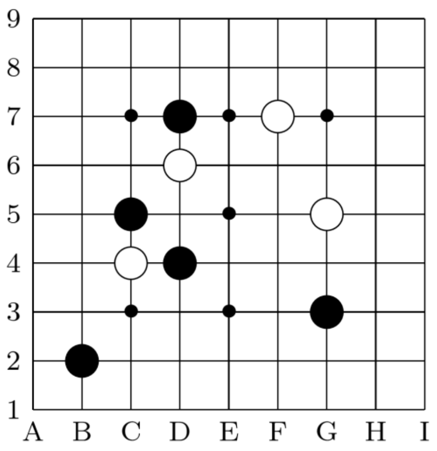


Figure 3.3. After the sampling

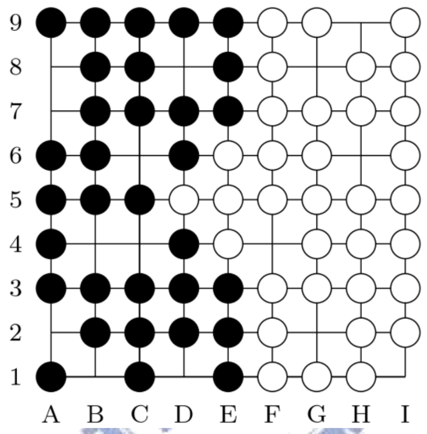


Figure 3.4. Final position

3.2 Variants

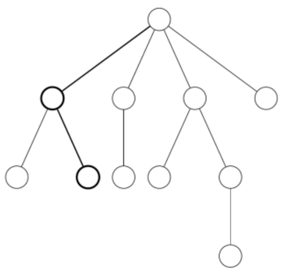
3.2.1 Monte Carlo Tree Search

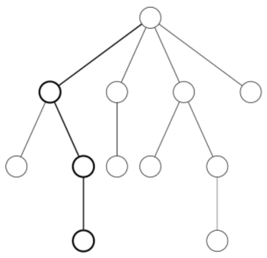
This method is now widely used and has been tried on Phantom Go, and we consider that it is necessary to evocate it.

The basic idea of MCTS is to create a partial game tree, but using at the same time the advantages of Monte Carlo Search. We will give the description of the algorithm, which has firstly been described in [24]. The MCTS algorithm is composed of 4 steps, which will be repeated until we have no more time (see

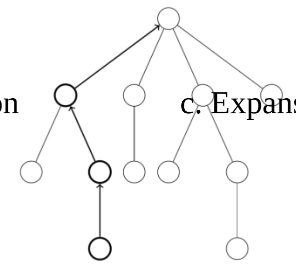
Figure 3.5, inspired from [11]):

1. Selection (as shown in Figure 3.5 (b)): According to an algorithm, for instance UCB, we choose, from the root, a leaf to expand in the partial tree
2. Expansion (as shown in Figure 3.5 (c)): From the selected leaf (a non-final state of the game) we choose one legal move, and add this node as a child of the selected node
3. Simulation (as shown in Figure 3.5 (d)): From the selected child, we run a playout till a final state, and get the score of the current player.
4. Backpropagation (as shown in Figure 3.5 (e)): We update information of the partial tree. For instance, if we use UCB, we increment the number of times each node of the selected sequence has been played, and also the number of "wins" of these nodes if the outcome of the playout is "win".



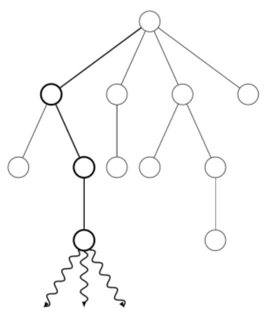


b. Selection



c. Expansion

d. Simulation



e. Backpropagation

Figure 3.5. MCTS algorithm, inspired from [11]

In [3] and [4], Borsboom shows that using MCTS is not better than a flat Monte Carlo approach. This result can be understood because MCTS, as flat Monte Carlo search do not take in consideration the imperfect information of the game. This is why we use another variant of the Monte Carlo method, inspired by Nested Monte Carlo and Recursive Perfect Information Monte Carlo.

3.2.2 Nested Monte Carlo

This method has firstly been published in 2009 by Cazenave, in [9]. A similar method has been used, as “Recursive Monte Carlo Search” by Furtak and Buro in [7] (cf. next subsection).

In this section, we will describe the nested Monte Carlo algorithm, and in the next one the imperfect information Monte Carlo Search, which is the adaptation of the nested Monte Carlo search to imperfect information games.

In the nested Monte Carlo algorithm, the idea is to use the general flat Monte Carlo algorithm, but replacing the traditional playout by a Monte Carlo one. This Monte Carlo policy can be nested itself. We give a more formal definition in Algorithm 3.2.1, from [9].

Algorithm 3.2.1. Nested Monte Carlo

```
function Nested(Position position, int level)
   best score ← - 1
# initialization of best score
   while not end of game do
      if level = 1 then
# the last level, we do not nest anymore.
         move = argmaxm(playout(play(position;m)))
      else
# we nest the algorithm.
         move = argmaxm(nested(play(position;m); level
      1))
      end if
      if scoreofmove > bestscore then
         best score ← scoreofmove
         best Move ← moveofbestsequence
      end if
   end while
   position = play(position; bestMove)
# we effectively play the best move
   return score
end function
```

The nested Monte Carlo algorithm has been used to get better results than flat Monte Carlo in real time. But Nested Monte Carlo can be used in another way. By nesting the Monte Carlo method once, we can use this method to deal with imperfect information, as we can see in the next section.

Figure 3.6 shows how this algorithm works. To help comprehension, we only use one playout per position, but you can notice that more often, we make several playouts for the same move, to get a better value.

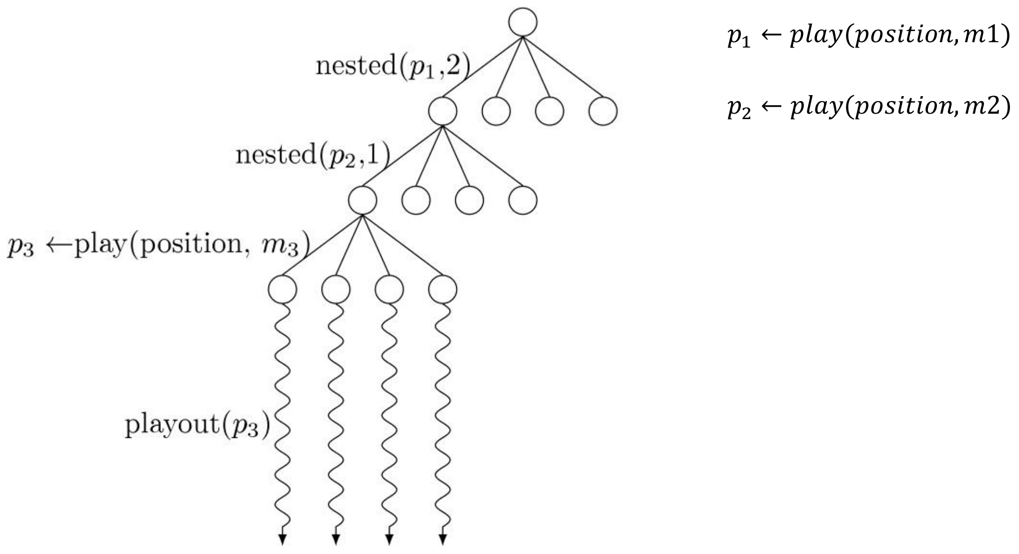

nested(p_1,2)
nested(p_2,1)
p_3 ←play(position, m_3)
playout(p_3)
$p_1 \leftarrow play(position, m1)$
$p_2 \leftarrow play(position, m2)$

Figure 3.6. Nested Monte Carlo, 3 levels

3.2.3 Imperfect Information Monte Carlo Search

There is a good reason not to deal with imperfect information games as we deal with games of perfect information with Monte Carlo methods. As we explained before, the flat Monte Carlo method proceeds as follows: After sampling the board, we play as the game were a game of perfect information, and do not take in consideration the fact that the game is an imperfect information one (see for instance [14]).

In [7], Michael Buro and Timothy Furtak show how to use the concept of a recursive Monte Carlo Search to deal with imperfect information. To do so, the main idea is to nest the sampling as we nested the Monte Carlo algorithm. Here is the algorithm:

Algorithm 3.2.2. Imperfect Information Monte Carlo Search

```
function Imperfect(Info Set IS, Player P, Time t)
   for m ∈ Moves(IS ) do
      val[m] ← 0
   end for
   while t 6 = 0 do
      X ← Sample(IS )
      for m ∈ Moves(X ) do
         v ← FinishedGameV alue(X;m; P )
         val[m] ← val[m] + v
      end for
      t ← Remaining time
   end while
   return argmax_m(val[m])
end function
function FinishedGameValue(State X, Move m, Player P)
   Y ← Play(P;X;m)
   while Y is not a terminal Position do
      P ← Next Player(P )
      Y ← Play(P; Y; ChooseMove(P; GetIS(Y; P )))
# GetIS (Y; P ) gets the Information Set of player Pin the
State Y
   end while
   return Score(P; Y )
end function
```

What is to notice is that *each time we call imperfect* we nest the Monte Carlo algorithm, we sample the board. We transmit an information set, nothing more. Which means that each player has to complete this information set, to sample the board, for each new call of FinishedGameValue. Nonetheless, once we get to the last level, we do not sample anymore and make a playout, which means that from this point, we make the same mistake as before.

The Figure 3.7 illustrates the algorithm

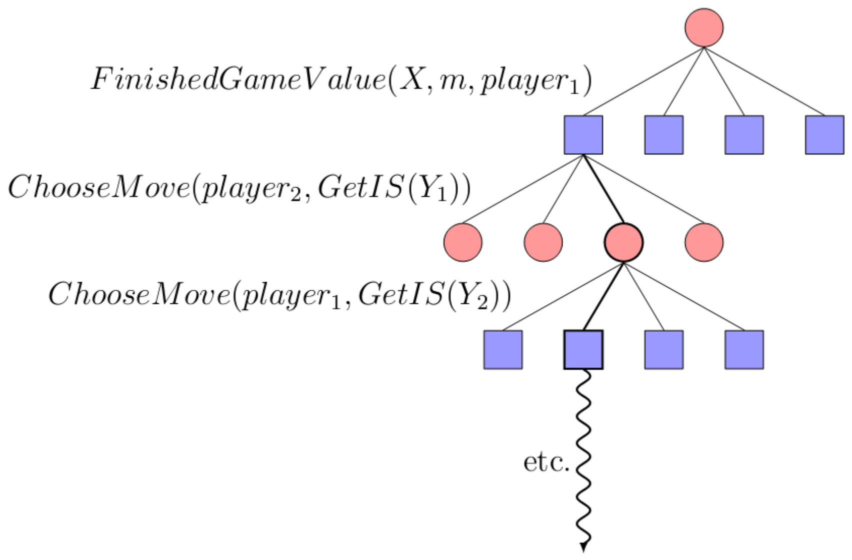

$FinishedGameValue(X, m, player_1)$
$ChooseMove(player_2, GetIS(Y_1))$
$ChooseMove(player_1, GetIS(Y_2))$
etc.

Figure 3.7. Imperfect Information Monte Carlo Search

3.3 Improvements

3.3.1 AMAF

All Moves As First is an improvement for Monte Carlo Search, and has been developed by Silver and Gelly (cf. [16]) and used in Go and MCTS [17] with success. The main idea is to consider during the playouts the moves as if they were children of the current node. In our case, we use this improvement for flat Monte Carlo Search, which means that in spite of updating only the score of the chosen move after a playout. While AMAF gives the same pound to each score (whether it is the child of the current node or not) we update the scores of all the moves which have been played, we give a different pound to the real child of the current position and to the other ones. This method is called α-AMAF, and is explained by Helmbold, and Parker in [22]. The formula to determine the score of a legal move i is:

$$\alpha V_i + (1 - \alpha) v_i$$

where V_i is the score of i as the child of the current move, and v_i is the score of the move not as the child of the current move.

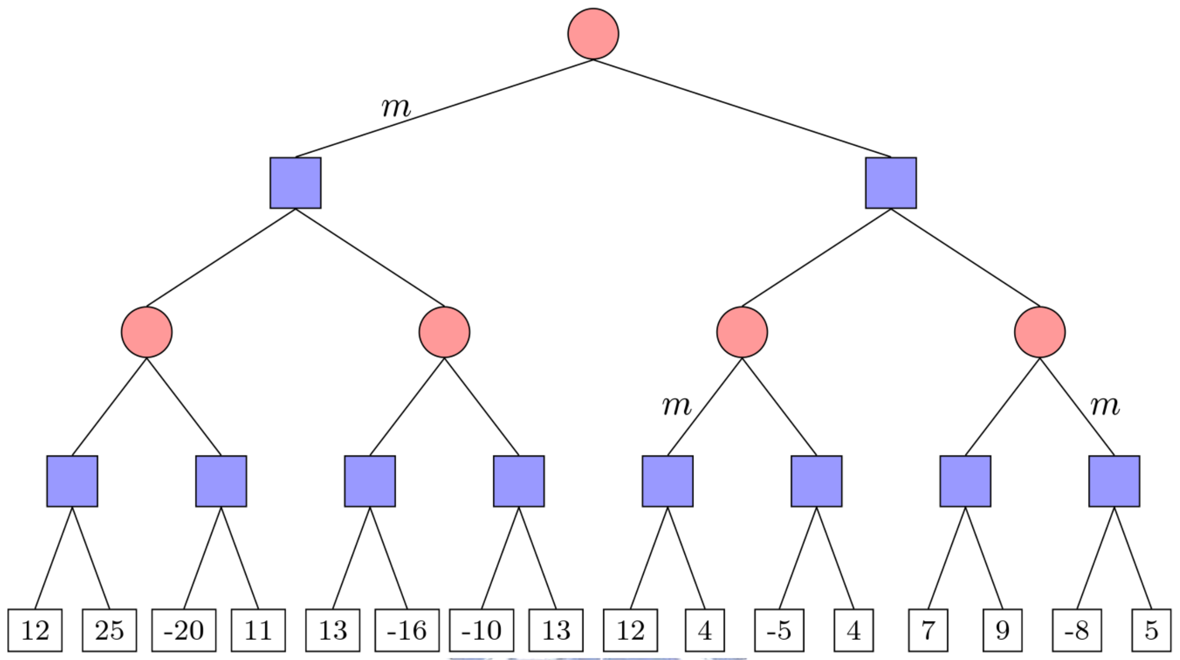


Figure 3.8. Illustration of AMAF

To illustrate AMAF, we can have a look at the tree on Figure 3.8. If we use AMAF, with $\alpha = 0.7$ the score of the move m is:

$$0.7 \times (12 + 25 - 20 + 11 + 13 - 16 - 10 + 13) + 0.3 \times \left(0.5 \times \frac{12+4}{2} + 0.5 \times \frac{5-8}{2}\right)$$

$$= 3.425$$

3.3.2 RAVE

Rapid Action Value Estimation is an improvement of AMAF, from [16] and [22]. We notice that AMAF is quite efficient at the beginning of the game of Go, but far less at the end, when the situation highly depends on the order of the moves. Thus, for each move i we replace α by a varying value:

$$\alpha_i = \max(0, \frac{V - n_i}{V})$$

where V is a determined value and n_i is the number of times the move i has been played.

3.3.3 UCB

Upper Confidence Bound has been developed by Auer, Fisher and Cesa-Bianchi in [2]. It allows to choose the most interesting moves. Indeed, it is not necessary to search moves which are notably bad. To do so, we choose the child to sample according this formula:

$$move = argmax(\overline{V_i} + \sqrt{\frac{2\ln n}{n_i}})$$

where $\overline{V_i}$ is the average score (number of "wins") of the move i, n_i the number of times we played move i, and n the total number of simulations. UCB takes two elements in consideration. The average number of wins of a move represents the exploitation, which means we try to focus more on the best moves. On the other hand, it is possible that a move is actually good even if we do not have a good score for it for the moment. We have to explore at least a little each move, and this is the role of the second term. UCB plays with these two aspects. Let give an example of UCB on Figure 3.9 (we assume that the chosen move leads to a lost playout). This policy allows to get a lower regret. UCB has mainly been used in Monte Carlo Tree Search. Nonetheless, as shown in [12], it is also possible to use it in Flat Monte Carlo programs.

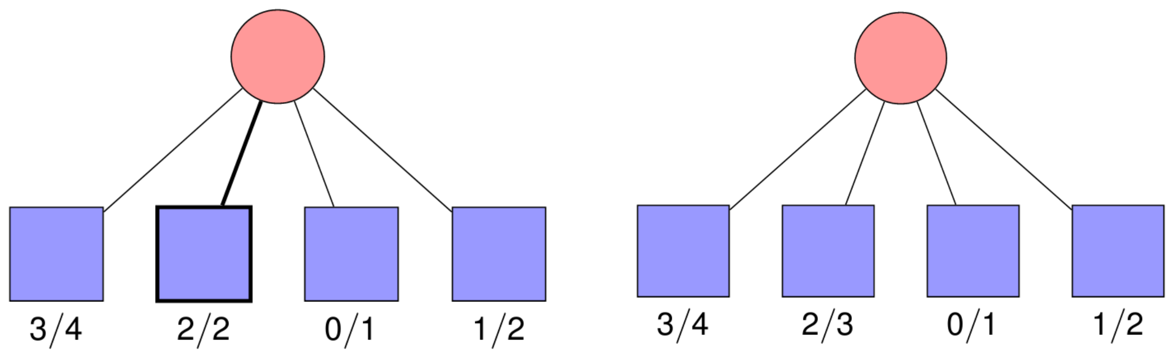


Figure 3.9 Illustration of UCB

Chapter 4 Two-level Monte Carlo Search

To iterate is human, to recurse divine.

L. Peter Deutsch

In this chapter, we describe more precisely a new method, Two-level Monte Carlo Search, and how we use the Monte Carlo algorithms to deal with Phantom Go. We will firstly describe the basic idea, and then the improvements we did on it.

4.1 Basic idea

The basic idea of the two-level algorithm is to use the Imperfect Information Monte Carlo Search, but adapted to our problem. To do so, we look at the article of Furtak and Buro. In this article, [7], Furtak and Buro explain that for games, nesting Monte Carlo once is sufficient. Moreover, as we are going to see, nesting the algorithm more than once would be far too huge.

Before showing the effect of the two-level algorithm, let have a look on the algorithm itself, on Algorithm 4.1.1. This algorithm is identical to the Algorithm 3.2.2, but written in another way.

Algorithm 4.1.1. Two-level MCS for Phantom Go

```
function LMC(InfoSet IS, int level, Player player)
   best score ← -1
# initialization of best score
   while not end of the game do
      Position position ← Sample(IS )
      if level = 1 then
# the last level, we do not nest anymore
         position  play(position;m; player)
         score ← playout(position)
      else
# we nest the algorithm
         position ← play(position;m)
         infoSet ← getIS (position; next(player))
         score ← LMC (InfoSet; level - 1; next(player))
      end if
      if score > max then
         max ← score
         move ← m
      end if
   end while
   position ← play(position;m)
   return max
end function
```

In this algorithm, there is a subroutine: getIS. This subroutine takes a player and a position as parameters and returns the supposed Information set of the next player. Now, let have a look on the effect on the board. Let take the example of the Figure 3.2. Let suppose it is Black’s turn. To begin with, Black samples the Referee’s board (for instance as shown on Figure 4.1).

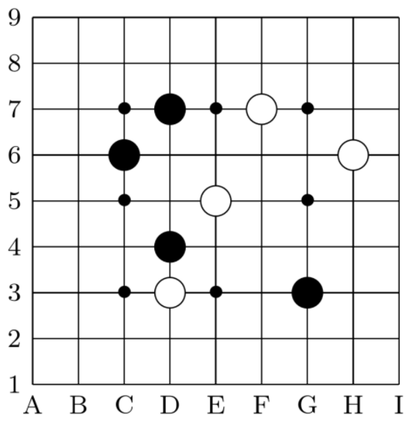


Figure 4.1. Sample of the Referee's board by Black

For each legal move, we do the following : play the move (we show on Figure 4.2 the Referee's board after playing on C3).

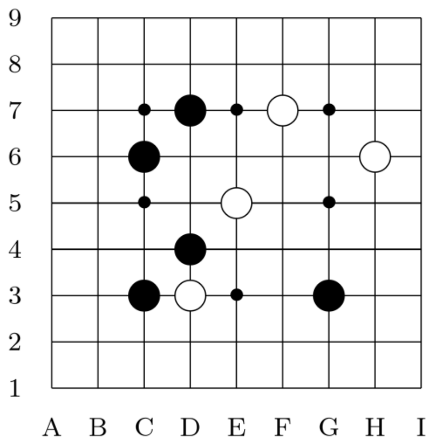


Figure 4.2. Black plays on C3

Then, we sample the White's Board, *i.e.*, the Information Set of White (effect of *getIS*). We suppose that Black does not know that White knows that on C6 is a black stone. The result is shown on Figure 4.3.

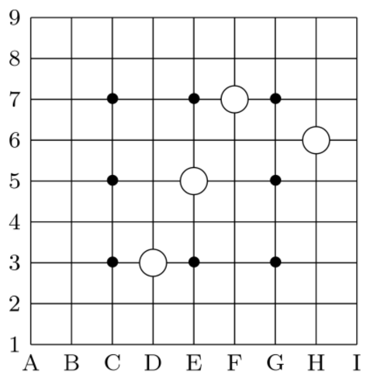


Figure 4.3. Sample of the White's board by Black

The black board still remains as before. Notice that to get the information set of White, we need to remember what White knows about Black, or at least as many information as possible. Now we nest the algorithm.

So, White samples the board. We give an example of what this sampling could be on Figure 4.4.

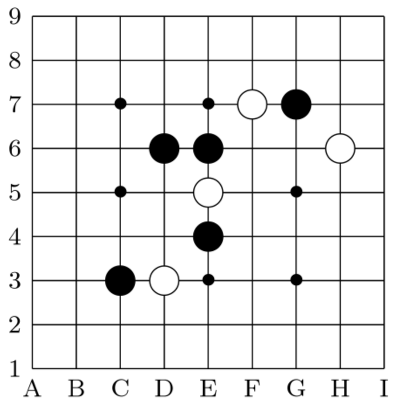


Figure 4.4. Sample of the Referee's board by White

We now make one playout (or more) for each move, for instance for B8 (Figure 4.5).

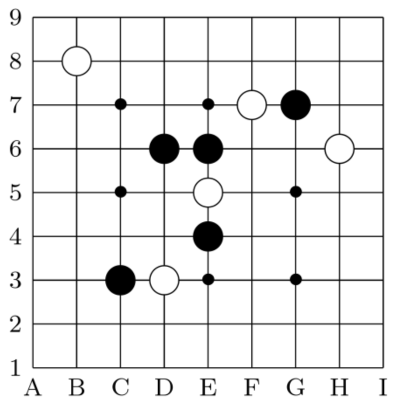


Figure 4.5. White plays on B8 before making a playout

Then, we get a score by making a playout. For instance, after making the playout (the final referee’s board is shown on Figure 4.6), we get the score (here, 5.5 for Black).

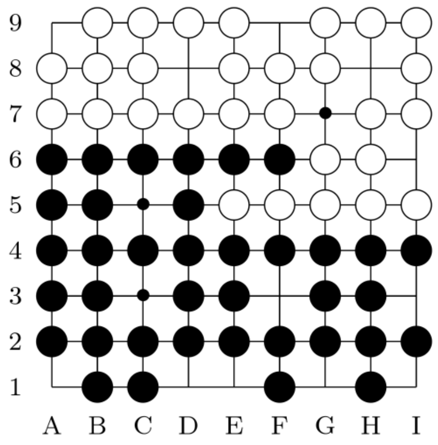


Figure 4.6. The playout

According to these playouts, White chooses a move, then, it is turn of Black (which also chooses a move according to flat Monte Carlo Search), then of White, till the end of the game. We then get a score, which will be the score of the move of black.

4.2 Complexity

The method leads to some problems. The main problem is the complexity. Let express the complexity in terms of playouts.

To get the score of a legal move, we have to make White and Black choose alternately till the end of the game. For each move, each player has to make a certain number of playouts for each legal move. So the total number of playout for one move is approximately 100 (number of moves until the end of the game) times 80 (number of legal moves). So to choose a game, we have to do that for each move, so to choose a move, we have to multiply the number of playouts by 80. So to choose a move, we have approximately to make $6 \cdot 10^5$ playouts. For the moment, the number of playouts is too important, and we need at least 20 minutes to choose one move (with 3 playouts and 3 nested playouts only on Pentium i5).

4.3 Improvements

Despite the different drawbacks, we tried to make some improvements so that this method is better, and can, if we give time enough, be at least as good as the flat Monte Carlo algorithm. To do so, we used two usual improvements for Monte Carlo: UCB (*cf.* subsection 3.3.3) and RAVE (*cf.* subsection 3.3.2). We decided to improve only the "high level" (level 2) of the two-level Monte Carlo algorithm.

We also considered two different versions of the playouts. The first one is the classical "random" playout. The moves played during the playout are randomly chosen among the legal moves. The second one is identical to the playout used by MoGo[1]. This playouts takes far more in consideration. Firstly, it tries to save a string in Atari if such a move is available. Then it takes patterns in consideration. A pattern is a configuration for which we can

[1] For more precision on MoGo, please refer to [25]

immediately now what to do to win. The configurations are put in a table, with the corresponding action to make. After that, it tries to capture a string in Atari if such a move is available. If none of these moves are available, it chooses a legal move randomly, without filling the eyes.

The paper of Drake and Uurtamo, [13] gives some indication about the expected results of what would be the better between random (“heavy”) playouts and move ordering, but as this paper deals with go (not with phantom go), we decided to implement both.

Chapter 5 Results

I didn't fail the test, I just found 100 ways to do it wrong

Benjamin Franklin

As we have seen, it is impossible to get results with a 9×9 go board. Processing a single move is too long to get results this way. This is why we decided to work on 7×7 Phantom Go. We mainly used two programs. The first one is a flat Monte Carlo one, as described in the section 3.1. In the following, we call it FlatMC. We also tried the two-level Monte Carlo one, which we call LvlMC. As the results of Monte Carlo Tree search were worse than Flat Monte Carlo, we have decided not to take them in consideration in our study. More about the results of MCTS for Phantom Go are described in more detail in [3][4].

The version of the two-level Monte Carlo program is a UBC and RAVE one, but only for the second level of Monte Carlo. The first level has not got any of these improvements. It is a classical Flat Monte Carlo algorithm, with 10 playouts per move. The opponent version (the Flat Monte Carlo one) is without UCB nor RAVE. We decided to use a random version of the playouts, more than a rule-based one, as the rules would apply on suppositions (because of the sampling). About the structure of the game (code of the board, the strings etc.) we adapted the initial code of Golois (cf. [10]).

To process our programs, we used an Intel Xeon, 2.40 GHz. We decided to impose a time limit to our programs, but we quickly understood that there would be the same numbers of playouts for each player, as the playouts are the longest elements in the two-level algorithm. For instance, for 4 minutes, we approximately process 1'300'000 playouts per move.

time of processing for one move (sec)

number of wins of RecMC

secs	60	120	240
wins	1	0	3
wins (handicap)	1	1	2

Table 1. LvlMC (Black) playing against FlatMC (White). The number is the number of "wins" of Black, above 20 games

The results are given in the Table 1 and the Table 2. As we can see, the results are quite bad for LvlMC. For equivalent time of processing, the Flat Monte Carlo program is much better than the two-level one. We played 20 games for each duration (1 minute, 2 minutes and 4 minutes).

time of processing for one move (sec)

number of wins of FlatMC

secs	60	120	240
wins	20	19	19
wins (handicap)	19	19	18

Table 2. FlatMC (Black) playing against LvlMC (White). The number is the number of "wins" of Black, above 20 games

We then decided to give a handicap to the two-level program, one stone placed in the center of the board.

Table 3 represents the games between LvlMC and LvlMC. We notice that the results are slightly the same as Flat Monte Carlo ones. Notice that the performance is proportionally less important between 60 and 120 than between 60 and 240, which correspond to the observations of Furtak and Buro in [7] (cf. Figure 5.1)..

Time of processing for White

Time of processing for Black

secs	60	120	240
60	9	9	6
120	12	11	12
240	12	11	11

Table 3. LvlMC playing against LvlMC. The number is the number of “wins” of Black, above 20 games

5.1 Explanation to the results

Now, we let explain the results and why the two-level method is less efficient than the Flat Monte Carlo one. Firstly, we cannot make as many “top level” playouts as Flat Monte Carlo (for which every playout is a top level playout). Thus, the two-level method will be better than the Flat Monte Carlo one if and only if each level 2 playout is much more efficient than a classical one, and if there are sufficient playouts so that the numeral difference is not so important that the approximation will be accurate enough. In [7], we can see that the effectiveness of Monte Carlo methods do not increase linearly with the number of playouts. The Figure 5.1, deduced from this paper shows the performance of a Skat program, Kermit (sampling 160 worlds), playing against itself, sampling the indicated number of consistent worlds. The number given is the percentage of wins of Kermit baseline. The authors indicate that, above 160 worlds, the different players have significantly identical level.

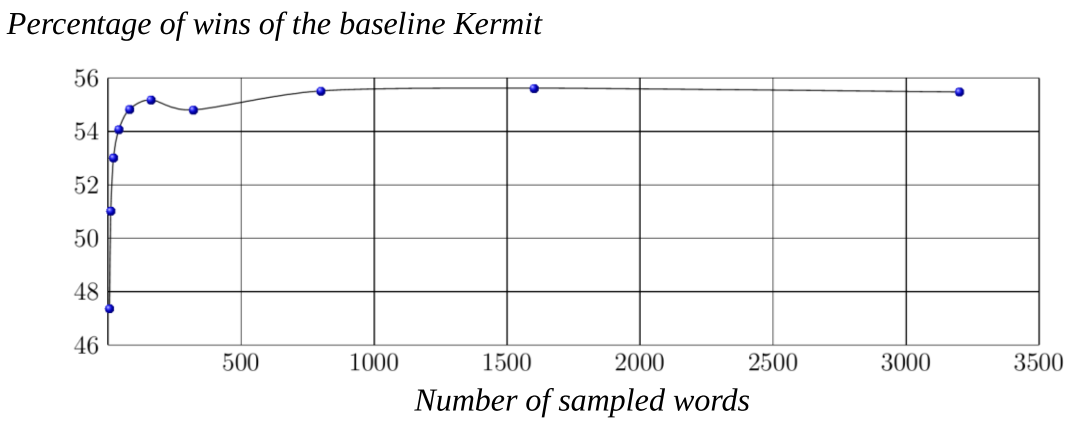
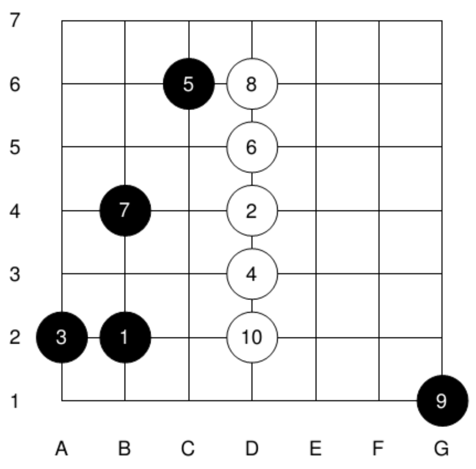


Figure 5.1. Kermit (baseline) VS Kermit160

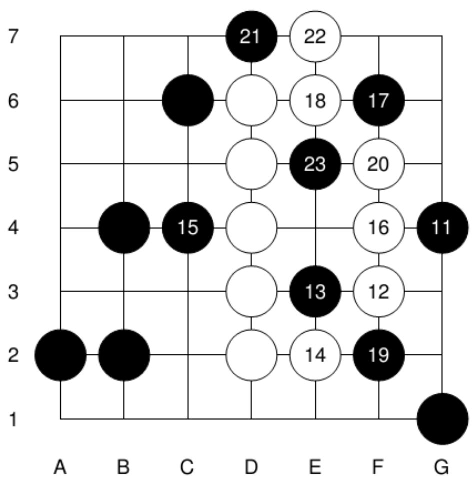
Another element to take in consideration is the relevance of the idea of a multi-level

Monte Carlo search. Even if each level 1 playout is considered not to have knowledge of the total situation, when we make our opponent choose a move, we take in consideration our sampling of his situation. This means we assume he will play according to this sampling, while it can be quite far from the real situation. Moreover, as the article of JR Long [27] shows, the classical Monte Carlo algorithm is already quite efficient. The paper of Buro and Furtak indicates that this method can be surpassed by a recursive one, but maybe not in our domain of study.

Furthermore, they address another kind of games, the card games. While the uncertainty is maximal at the beginning of the game and goes decreasing in such games, phantom Go has mainly two phases. In the early game, the uncertainty grows, then we start getting information from the referee, and the uncertainty decreases. That can explain quite important mistakes of the program we could observe at the beginning of the game. Indeed, [7] shows that for the games with "low disambiguation" this method performs worse than the flat one. But especially at the beginning Phantom go has a negative disambiguation (we add uncertainty). Figure 5.2 shows some steps of a real game between FlatMC (Black) and LvlMC (white). It performs very wrongly at the beginning, while it improves in the last moves, creating a living group and a territory. We only represent the referee's board, to get a general point of view.



a. The 10 first moves



b. After 13 more moves

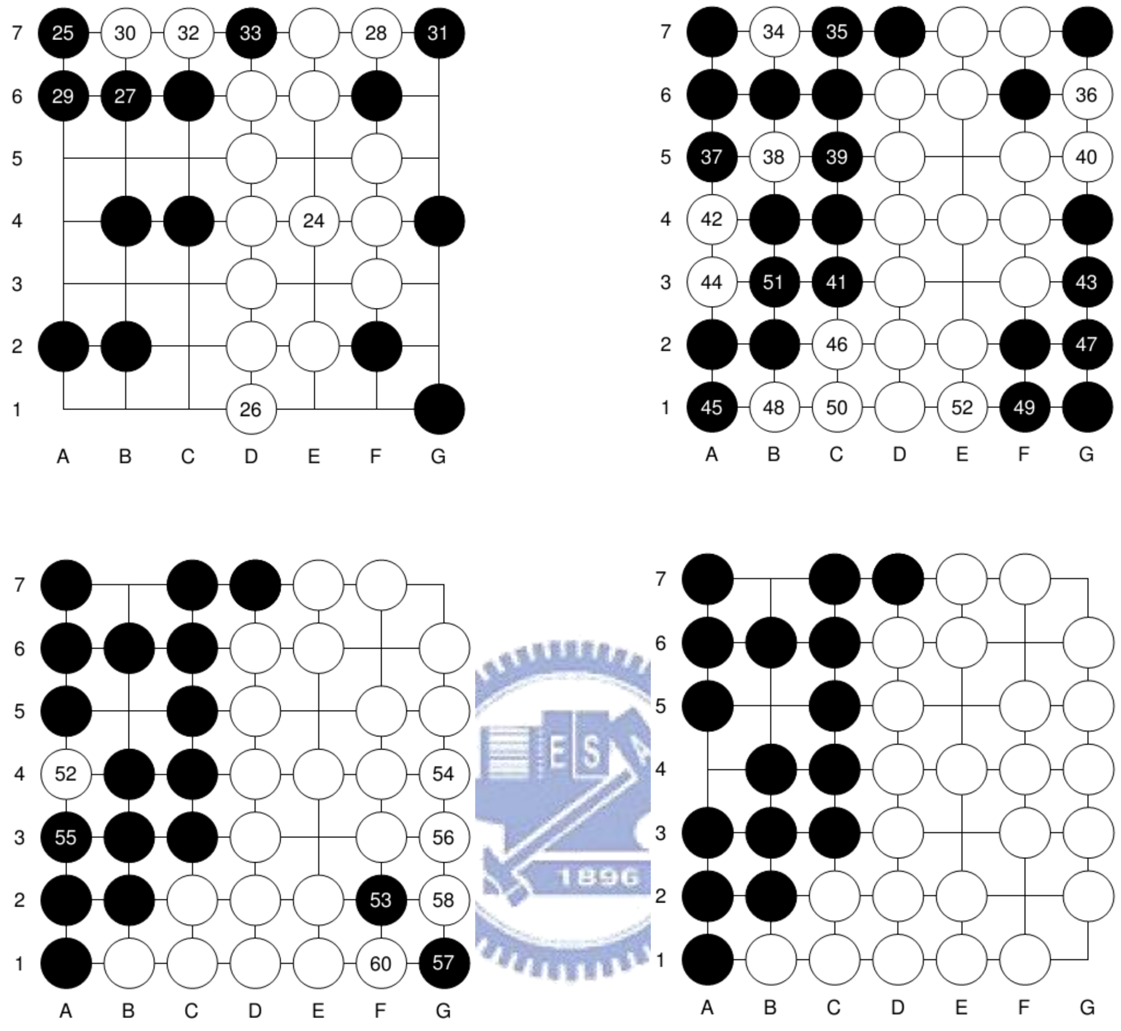


Figure 5.2. Summary of a game between FlatMC (Black) and LvlMC (White)

5.2 Discussion

The first thing is the way to deal with improvements. We decided to use UCB and AMAF only on the second level playouts, but we should be able to get better results working on the two levels, at least for UCB , which would avoid playing bad moves in the second level playouts would lead to more trustable ones. For AMAF, an adaptation could be taken from [1], which uses AMAF on each level, and seems to indicate that in spite of using the AMAF value, the number of times a move has been played is more efficient. Nonetheless, we have to remember that our work is about two players Imperfect Information Game, while this paper deals with Morpion Solitaire.

Chapter 6 Conclusion and future work

In this thesis, we compared several methods to build a playing program for Phantom Go. We also adapted a new algorithm, the Two-level Monte Carlo Search to an Imperfect Information game, Phantom Go. We improved this program using classical methods as AMAF and UCB. We tested the two-level program against the classical Flat Monte Carlo one and observed that it performed worse. We gave an explanation of this result, and explained how to improve programs.

As we could see in the previous chapter, it is hard to use the two-level algorithm to make an actual playing program. Nonetheless, it seems that the flat Monte Carlo method, which is much simpler, which performs well, can still stay the best option for this game.

Maybe the research to better address Phantom Go and beat the current version is to use domain-specific knowledge. For instance, in spite of searching the whole board, searching only a part of the board could better (the border is not a good place to play in). Another way of doing it would be to partially or totally script the first moves, which are closely always the same.

In fact, a hybrid version of two-level search and the Flat Monte Carlo one could also be good: the two-level method is useful when the uncertainty about the opponent is high, while for low uncertainty Flat Monte Carlo, which is quicker, will be better.

Bibliography

[1] Haruhiko Akiyama, Kanako Komiya, and Yoshiyuki Kotani. Nested monte-carlo search with amaf heuristic. In *Proc. Int. Conf. Tech. Applicat. Artif. Intell., Hsinchu, Taiwan*, pages 172–176, 2010.

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[3] Joris Borsboom. Phantom Go. Master's thesis, Universiteit Maasstricht, The Netherlands, 2007.

[4] Joris Borsboom, Jahn-Takeshi Saito, Guillaume Chaslot, and JWMH Uiterwijk. A comparison of monte carlo methods for phantom go. In *Proc. BeNeLux Conf. Artif. Intell., Utrecht, Netherlands*, pages 57–64, 2007.

[5] Mark Braverman, Omid Etesami, and Elchanan Mossel. Mafia: A theoretical study of players and coalitions in a partial information environment. *The Annals of Applied Probability*, pages 825–846, 2008.

[6] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[7] Michael Buro and Timothy Furtak. Recursive monte carlo search for imperfect information games. In *Proceedings of the 8th international conference on computer and games*, 2013.

[8] Tristan Cazenave. A phantom go program. *Advances in Computer Games*, pages

120–125, 2006.

[9] Tristan Cazenave. Nested monte-carlo search. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 456–461, 2009.

[10] Tristan Cazenave and Joris Borsboom. Golois wins phantom go tournament. *ICGA journal*, 30(3):165–166, 2007.

[11] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte carlo tree search: A new framework for game ai. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 216–217, 2008.

[12] Pierre-Arnaud Coquelin and Rémi Munos. Bandit algorithms for tree search. *arXiv preprint cs/0703062*, 2007.

[13] Peter Drake and Steve Uurtamo. Move ordering vs heavy playouts: Where should heuristics be applied in monte carlo go. In *Proceedings of the 3rd North American Game-On Conference*. Citeseer, 2007.

[14] Ian Frank and David Basin. Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence*, 100(1):87–123, 1998.

[15] Ian Frank and David Basin. A theoretical and empirical investigation of search in imperfect information games. *Theoretical Computer Science*, 252(1):217–256, 2001.

[16] Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM, 2007.

[17] Sylvain Gelly and David Silver. Monte carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[18] Matthew L Ginsberg. Gib: Imperfect information in a computationally challenging game. *J. Artif. Intell. Res.(JAIR)*, 14:303–358, 2001.

[19] John Charles Harsanyi. Games with incomplete information played by "bayesian" players, i–iii: part ii. bayesian equilibrium points. *Management Science*, 14(5):320–334, 1968.

[20] John Charles Harsanyi. Games with incomplete information played by "bayesian" players, i–iii: part iii. the basic probability distribution of the game. *Management Science*, 14(7):486–502, 1968.

[21] John Charles Harsanyi. Games with incomplete information played by "bayesian" players, i–iii: part i. the basic model. *Management science*, 50(12 supplement):1804–1817, 2004.

[22] David P Helmbold and Aleatha Parker-Wood. All-moves-as-first heuristics in monte-carlo go. In *Proc. Int. Conf. Artif. Intell., Las Vegas, Nevada*, pages 605–610, 2009.

[23] Michael Johanson, Nolan Bard, Marc Lanctot, Richard Gibson, and Michael Bowling. Efficient nash equilibrium approximation through monte carlo counterfactual regret minimization. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 837–846. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[24] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–293, 2006.

[25] Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, J-B Hoock, Arpad Rimmel, Olivier Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, and Tzung-Pei Hong. The

computational intelligence of mogo revealed in taiwan’s computer go tournaments. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(1):73–89, 2009.

[26] David H Li. *Kriegspiel: Chess Under Uncertainty*. Premier Publishing Company, 1994.

[27] Jeffrey Richard Long, Nathan R Sturtevant, Michael Buro, and Timothy Furtak. Understanding the success of perfect information monte carlo sampling in game tree search. *Proc. Assoc. Adv. Artif. Intell., Atlanta, Georgia*, pages 134–140, 2010.

[28] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[29] John Forbes Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[30] Martin J Osborne. *An introduction to game theory*, volume 3. Oxford University Press New York, 2004.

[31] Erik Cornelis Diederik van der Werf. *AI techniques for the game of Go*. UPM, Universitaire Pers Maastricht, 2005.

[32] John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior (commemorative edition)*. Princeton university press, 2007.