COMBINATORIAL GAME THEORY IN CLOBBER

Jeroen Claessen

Master Thesis DKE 11-06

Thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science of Artificial Intelligence
at the Faculty of Humanities and Sciences
of Maastricht University

Thesis committee:

Dr. M.H.M. Winands
Dr. ir. J.W.H.M. Uiterwijk
J.A.M. Nijssen, M.Sc.

Maastricht University
Department of Knowledge Engineering
Maastricht, The Netherlands
July 2011

Preface

This master thesis was written at the Department of Knowledge Engineering at Maastricht University. In this thesis I investigate the application of combinatorial game theory in the game Clobber, and its incorporation in a Monte-Carlo Tree Search based Clobber player, with the aim of improving its performance.

I would like to thank the following people for making this thesis possible. Most importantly, I would like to express my gratitude to my supervisor Dr. Mark Winands for his guidance during the past months. He has supplied me with many ideas and has taken the time to read and correct this thesis every week. Further, I would like to thank the other committee members for assessing this thesis.

Finally, I would like to thank my family and friends for supporting me while I was writing this thesis.

Jeroen Claessen
Limbricht, June 2011

Abstract

The past couple of decades classic board games have received much attention in the field of Artificial Intelligence. Recently, increasingly more research has been done on modern board games, because of the more complicated nature of these games. They are often non-deterministic games with imperfect information, that can be played by more than two players, which leads to new challenges. However, relatively little research has been done on combinatorial games. Combinatorial games are bounded by a more specific set of conditions, that ensure that for many board positions the exact game-theoretic value can be found.

This thesis focuses on the analysis and implementation of combinatorial game-theoretic values in the game Clobber, and their application to a Monte-Carlo Tree Search (MCTS) based Clobber player. Clobber is a 2-player partizan game with perfect information. It is deterministic since no chance is involved. The problem statement of this thesis is: *How can we apply combinatorial game theory in MCTS?* In order to answer this problem statement, combinatorial game theory is explored in general, as well as applied to Clobber. The theory is then incorporated in different phases of the MCTS Search algorithm.

The results of the experiments indicate that the performance of MCTS can be increased considerably by adding knowledge of combinatorial game theory. The use of MCTS that handled the combinatorial game-theoretic values of (sub)game positions resulted in a rather large increase of performance, compared to regular MCTS. Another attempt to add knowledge of combinatorial game theory to MCTS was made by simplifying certain board positions. However, the performance was worse than that of the regular MCTS program.

Contents

Chapter 1

Introduction

This chapter gives an introduction to the research topic discussed in this thesis, as well as a background of the techniques and theory used. Section 1.1 gives an introduction to the field of games and AI. Section 1.2 then explains the main techniques used in that field. Next, in Section 1.3, combinatorial games are introduced. In Section 1.4 game-theoretic values are discussed in general, as well as for combinatorial games specifically. The problem statement and research questions are given in Section 1.5. Finally, in Section 1.6 an outline of the remaining chapters is given.

Chapter contents: Introduction — Games and AI, AI Search Techniques, Combinatorial Games, Game-Theoretic Values, Problem Statement and Research Questions, Thesis Outline

1.1 Games and AI

Board games have been around for centuries, used by humans to test their intelligence or simply for entertainment. The past couple of decades, since computers have been getting increasingly powerful, much research has been dedicated to letting computers play these games at a strong level. One of the first to formulate a computer application for games was Turing (1953). Most of the research since then has been done for classic board games, although lately modern board games are getting increasingly popular. Classic board games usually are deterministic 2-player games with perfect information, and thus form an easier research domain than more complicated games, where often chance is involved and information can be hidden.

Over the years, Chess undoubtedly has received the most attention, resulting in the first big success, when program Deep Blue defeated the reigning world champion at the time (Campbell, Hoane Jr., and Hsu, 2002). Another game in which computer players have surpassed human play is Checkers, which was solved by Schaeffer *et al.* (2007). However, there are still many games which are currently only played at an amateur level, such as Go (Müller, 2002).

Research in games may seem rather limited, because one might think that the only application of a computer player is to play against human players. However, games make an excellent domain for introducing and investigating new AI techniques, which may later be applied to other domains (Nilsson, 1971), since games are always properly defined by their rules.

1.2 AI Search Techniques

The vast amount of research that has been performed in the field of AI has produced a large number of AI search techniques and variations of these techniques. The two most widely used techniques at the moment are $\alpha\beta$-search and Monte-Carlo Tree Search (MCTS).

The $\alpha\beta$-search algorithm is an adaption of the Minimax algorithm (Von Neumann, 1928). Minimax is a depth-first tree search algorithm for 2-player games. The two players are called the Max player and the Min player, where Max is the root player and tries to maximize his score, while Min tries to minimize the score. Minimax searches the game tree up to a certain ply, in order to find the best possible score for Max, considering that Min always tries to minimize the score. Note that the exact score is not always known if a move is not searched to the end of the tree. In that case the score is approximated using an evaluation function.

The Minimax algorithm was adapted into the $\alpha\beta$-search algorithm (Knuth and Moore, 1975). $\alpha\beta$-search stops evaluating a move as soon as a possibility is found that leads to a worse score than could already be reached by one of the previously examined moves. The rest of the children of this move are thus pruned, which results in a decreased computational time for $\alpha\beta$-search, as compared to the Minimax algorithm, while the resulting move is the same. There exist several variations of $\alpha\beta$-search that further aim to optimize the search process (Marsland, 1983; Reinefeld, 1983).

MCTS is a tree search algorithm that is based on Monte-Carlo evaluations. It was first proposed by Coulom (2007). MCTS builds a tree based on randomly simulated games. The tree holds information about the previously simulated games, which is used to decide the first couple of moves of a simulated game. As more simulated games are played, the information in the tree becomes more accurate, based on which the best moves can be approximated. Since MCTS does not require an evaluation function, it works well in games for which it is difficult to create one. It has especially been successful in the game of Go (Gelly, 2007; Coulom, 2007; Gelly and Silver, 2008). MCTS will be explained in further detail in Section 5.2.

1.3 Combinatorial Games

Although classic board games have been researched exhaustively, combinatorial games have received little attention. Combinatorial games are different from classic games in that they are bound by a more specific set of conditions; they are sequential 2-player games with perfect information (the exact conditions are discussed in Section 3.1). These conditions ensure that the combinatorial game-theoretic value of a game can be calculated for many game positions.

The notion of combinatorial game theory was first introduced by Bouton (1902), who did a mathematical analysis of the game Nim. However, it was not until several decades later before this theory was formalized and further investigated, by Conway (1976). This research lays the foundation for combinatorial game theory, and is still influential. A couple of years later, Conway again publishes research on the subject, with Berlekamp, Conway, and Guy (1982). In this book, a large number of games are introduced, along with the relevant theory for analyzing them. This has proven to be another influential publication and is still an important source of information on combinatorial game theory.

Other publications worth mentioning are those by Wolfe (1991) and Siegel (2005). Moreover, efforts have been made to improve standard search techniques by combining combinatorial game theory with $\alpha\beta$-search (Müller, Enzenberger, and Schaeffer, 2004) and MCTS (Cazenave, 2011).

The combinatorial game-theoretic value is usually hard to calculate for complicated positions (i.e. positions with a large game tree), so for most games it is not possible to find the value of the initial position(s), however, as the game progresses, increasingly more positions can be solved. This is especially the case since combinatorial games often fall apart into subgames after a number of moves. Each subgame is a group of pieces that cannot interact with any other pieces on the board anymore. Therefore, subgames can be seen as separate games, where a player may choose in which game to play each turn. The game-theoretic values can be calculated separately, which is easier, since instead of one large position several smaller positions are considered. Moreover, it turns out that the values of all separate subgames of a position can be summed to obtain the value of the whole position (Berlekamp *et al.*, 1982).

Since the combinatorial game-theoretic value of a game position can usually only be calculated after it has fallen apart into separate subgames or when the position itself is small enough to solve, combinatorial game theory is especially useful for playing endgames. However, even if not all subgames can be calculated yet, the values of the subgames that can be calculated may give useful information when playing a game. For instance, a subgame in which no moves are possible never has to be considered. Moreover, subgames that have value 0 and subgames that have a combined value of 0 do not have to be considered. This is explained in detail in Section 5.4.

1.4 Game-Theoretic Values

In the field of AI, the game-theoretic value of a game is an important notion. Usually, it is used to denote the outcome of a game when played optimally by both players from a given position. For a two player game, the outcome can be a win, a loss or a draw for the first player. In some games the game-theoretic value can also be a measure of how large the (dis)advantage is for a win or loss, like in the game of Go (Van der Werf and Winands, 2009). When the value of a position is known, this position is said to be *solved*. There are three different degrees of solving (Allis, 1994):

1. **Ultra-weakly solved**: The game-theoretic value of the initial position(s) is known, but the strategy to arrive at that value is unknown. An example of an ultra-weakly solved game is Hex (Nash, 1952).
2. **Weakly solved**: For the initial position(s) the game-theoretic value is known and a strategy to arrive at that value is obtained for both players, under reasonable resources. Examples of weakly solved games are Nine Men's Morris (Gasser, 1991) and Checkers (Schaeffer *et al.*, 2007).
3. **Strongly solved**: The game-theoretic value is known for all legal positions, and a strategy to arrive at that value is obtained for both players, under reasonable resources. Examples of strongly solved games are Awari (Romein and Bal, 2003) and Connect Four (Tromp, 2008).

In combinatorial game theory, the (combinatorial) game-theoretic value has a slightly different meaning. The combinatorial game-theoretic value of a game not only denotes a win or loss for a given position (a draw is impossible in combinatorial games, see Section 3.1) and a measure of the advantage for the winning player, it also means that a strategy is known to obtain at least this value (Berlekamp *et al.*, 1982). Moreover, the value is *independent* of the starting player. That is, if the value denotes a win for one of the players from a certain position, that player can always win from that position, no matter which player moves first. It is also possible that the combinatorial game-theoretic value denotes a win for either the first or second player to move. In that case, the value is still the same, no matter which player starts, however, it simply denotes that the winner of the game depends on which player's move it is. In this thesis a position is solved if we can determine its exact combinatorial game-theoretic value.

1.5 Problem Statement and Research Questions

The goal of this thesis is to investigate the application of combinatorial game theory in existing AI methods for playing Clobber. MCTS is specifically investigated. The following problem statement is defined:

Problem statement: How can we apply combinatorial game theory in MCTS?

Based on the problem statement, the following three research questions are formulated:

Research question 1: How can we use endgame databases to store the combinatorial game-theoretic value of a subgame in an efficient and effective way?

Although this is not directly related to the application of combinatorial game theory in MCTS, it is an important aspect of the research. MCTS relies on many simulations, and in each step it has to access to the game-theoretic values of subgames. Therefore, the values need to be retrieved in a time efficient manner.

Research question 2: How can combinatorial game-theoretic values be used to improve the selection phase of the MCTS program MC-Mila?

MC-Mila is the MCTS version of the program Mila, which is based on $\alpha\beta$-search (Winands, 2006). Combinatorial game theory can be used during the selection phase of MCTS, since at any time during that phase the values of some subgames might be known. This perfect knowledge can be used to improve the selection phase.

Research question 3: How can combinatorial game-theoretic values be used to improve the playout phase of the MCTS program MC-Mila?

The use of combinatorial game theoretic knowledge can be used to improve the playout phase, in similar manners as the selection phase can be improved. Moreover, if the values of all subgames of a board position are known, the playout of the game can be terminated early.

Research question 4: How can combinatorial game-theoretic values be used to improve the endgame play of the MCTS program MC-Mila?

Near the end of the game, the field will eventually decompose into subgames of which the game-theoretic values can all be calculated. From that point on it is possible to determine an optimal strategy for playing out the game.

1.6 Thesis Outline

This thesis is organized as follows:

Chapter 1 gives an introduction to the field of AI and discusses some of the most widely used AI techniques at the moment. Next, it gives a short introduction to combinatorial games and the notion of game-theoretic values in such games. Finally, the problem statement and research questions are stated, and related research is discussed.

Chapter 2 gives an overview of the game of Clobber. Firstly, the origins of the game are given, followed by an explanation of the rules. Next, we aim to measure the difficulty of the game by approximating the state-space and game-tree complexities. Furthermore, the research interest of Clobber is discussed.

Chapter 3 starts with an introduction to combinatorial game theory. It discusses what games are considered combinatorial games and their characteristics. Next, the relevant theory for this research is explained. Firstly, the basics of combinatorial game theory are explained by means of the game Hackenbush. Secondly, the theory that is required for analyzing Clobber is described. Finally, some rules for simplifying values are given. The chapter ends with a discussion about the implementation of this theory.

Chapter 4 describes how the database is built. First, the structure of the database is explained, as well as the manner in which values are stored in the database. Next, the way in which the database is filled is discussed, and the techniques that are used to speed up this process are given. Furthermore, contents of the database are explored and it is investigated what kind of positions could be solved.

Chapter 5 discusses MCTS. The origins of this technique are given and its workings are explained. Some of the successes that MCTS has achieved in the past couple of years are presented. Furthermore, the MCTS-solver is explained, which is a MCTS enhancement that is used by MC-Mila. Finally, two approaches for enhancing MCTS with knowledge of combinatorial game theory are discussed.

Chapter 6 gives an overview of the experiments that are conducted and the results of these experiments.

Finally, in **Chapter 7**, the research is concluded and the problem statement and research questions will be answered. Additionally, future research is discussed.

Chapter 2

Clobber

In this chapter the game of Clobber is introduced. Clobber is a combinatorial game that was designed in 2001 by Michael Albert, J. P. Grossman and Richard Nowakowski (Albert et al.*, 2005). It is a deterministic 2-player game with perfect information. The origins of the game are discussed in Section 2.1. The rules are given in Section 2.2. In Section 2.3 the complexity of Clobber is analyzed. Section 2.4 gives an overview of earlier research in Clobber.*

Chapter contents: Clobber — Origins, Rules, Complexity Analysis, Research Interest

2.1 Origins

Clobber was designed in 2001 by Michael Albert, J. P. Grossman and Richard Nowakowski (Albert *et al.*, 2005). It was derived from the game Peg Duotaire, which is the 2-player version of Peg Solitaire. In Peg Duotaire, the board exists of rows of holes, which can hold pegs. Players move by jumping a peg over an orthogonally adjacent peg, landing it in an empty hole behind it. Take for example this simple 1-dimensional game: ●●○, where a peg is denoted by ● and an empty hole is denoted by ○. The only possible move in this example is where the leftmost peg jumps over the peg to its right, to the empty hole, resulting in the position ○○●. The players continue alternating moves until one player cannot move anymore, and loses the game. In the example previously given, the first player wins the game, since the second player has no move left.

A game that is similar to Peg Duotaire is the old Hawaiian board game Konane. This game is known to be played at least 200 years before Peg Duotaire. Konane is played on a 10×10 checkerboard, where the pieces are placed on the corresponding squares. The players move by jumping over an opponent's piece, just like in Peg Duotaire. However, in Konane, moves may be chained, as long as the moves are all in the same direction. The last player that is able to make a move wins the game.

2.2 Rules

Clobber is a 2-player game that is played on a checkerboard, usually of size 8×8 or 10×10. Each square is initially occupied by a piece of the same color. The initial board configuration for an 8×8 game is given in Figure 2.1. White starts the game, after which the players continue by alternating moves, where white moves the white pieces, while black moves the black pieces. The players move by 'clobbering' an orthogonally adjacent piece of the opponent, where clobbering is defined as moving one's own piece to the adjacent square, replacing the opponent's piece, which is removed from the board. In the game position ●○● a white piece is denoted by ○, a black piece by ● and an empty square by a space. In this example, white has two moves, by either clobbering the black piece to its left, or the one to its right. The former results in the position ○ ●. The first player that is unable to move loses the game (i.e. the last player to move wins).

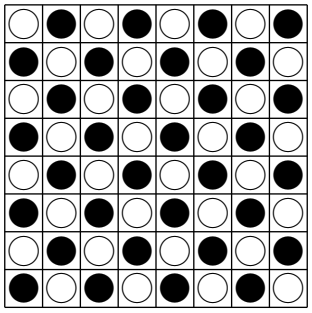


Figure 2.1: The initial board configuration for an 8×8 game of Clobber.

Since pieces can only move to an orthogonally adjacent square, a game of Clobber usually decomposes into smaller subgames. This happens whenever two parts of the board are separated by vacant squares, or even when the parts are connected only by diagonally adjacent pieces. After a game has decomposed into subgames, these subgames can be played independently of each other. This will be further described in Chapter 3.

2.3 Complexity Analysis

This section aims to measure the difficulty of Clobber, by calculating the value of the two main complexity measures: state-space complexity and game-tree complexity (Allis, 1994). The state-space complexity is the number of possible states in the game. In Clobber, an approximation of the state-space complexity can be computed using the following equation:

$$\sum_{i=1}^{n \times m - 1} \binom{n \times m}{\lceil \frac{i}{2} \rceil} \cdot \binom{n \times m - \lceil \frac{i}{2} \rceil}{\lfloor \frac{i}{2} \rfloor},$$

where n is the number of rows and m the number of columns of the board. This equation only considers positions with an equal number of black and white pieces, or where White has exactly one more piece, since black and white pieces are removed from the board alternately, and White is the starting player. The position with exactly $n \times m$ pieces is neglected, because this is the starting position, which is fixed. Still, this equation gives an upper bound of the actual value, since for each number of pieces on the board all possible positions are counted. However, not all positions can be reached from the starting position (the 8×8 starting position is given in Figure 2.1). For the two most commonly used board sizes, the state-space complexities are given in Table 2.1.

board size	state-space complexity	game-tree complexity
8×8	4.16×10^{29}	$40.0^{46.3} = 1.30 \times 10^{74}$
10×10	5.01×10^{46}	$56.7^{71.6} = 3.60 \times 10^{125}$

Table 2.1: The complexities of Clobber for board sizes 8×8 and 10×10.

The game-tree complexity is the number of leaf nodes in the solution tree of the initial position of the tree. A way to find the game-tree complexity is by calculating b^l, where b is the average branching factor and l is the average game length (in ply). The values of b and l where approximated by letting Mila play 400 against itself. For a board size of 8×8, the average branching factor for these 400 games was 40.0 and the average game length was 46.3. For a board size of 10×10, the average branching factor was 56.7 and the average game length was 71.6. The game-tree complexities of these two board sizes are given in Table 2.1.

2.4 Research Interest

Despite its straightforward rules, Clobber is an interesting game from a research point of view. Since it satisfies the conditions for a combinatorial game according to Berlekamp *et al.* (1982), the game-theoretic

value can be computed for simple board positions (See Chapter 3). For this reason, the game has been studied by game theorists such as Demaine, Demaine, and Fleischer (2002).

Research on combinatorial game theory applied to Clobber was first published by Albert *et al.* (2005). Upon the introduction of the game in that same article, the game-theoretic values of several $1 \times n$ and $2 \times n$ positions were given. Furthermore, atomic weights for Clobber positions were introduced, and calculated for some more complicated positions.

Recently, Siegel (2007) developed the Combinatorial Game Suite. This is a useful tool for analyzing Clobber positions. It can compute the game-theoretic values, as well as the atomic weights, for several games, including Amazons, Domineering and Clobber.

Since 2005 Clobber has been an event in the Computer Olympiad. The first edition was won by the $\alpha\beta$-program Mila (Willemson and Winands, 2005). In the second edition the program Pan, by Johan de Koning, came in first. Since then Clobber has not been played in the Computer Olympiad, due to a lack of participants.

Another Clobber competition worth mentioning was the one at Tartu University (Willemson, 2005). Students were required to implement a Clobber program as a part of their course. This resulted in a competition with a total of 34 participants. It turned out that the strategies of the most successful programs were based on the same idea, namely that it is favorable to have a subgame with many own pieces and few of the opponent. This was achieved in two different ways. Firstly, they tried to maximize the difference of own pieces minus opponent's pieces in active subgames (i.e. subgames that contain both black and white pieces). Secondly, they tried to reduce the number of own subgames, which should give the same result. Although the strategies based on these observations generally performed well, it does not always turn out as desired. Large subgames can often be easily split, after which the position might not be as favorable anymore. Moreover, if such a large subgame does not include any opponent's pieces anymore, it becomes useless.

Chapter 3

Combinatorial Game Theory

THIS chapter introduces Combinatorial Game Theory and describes some of its basics. Moreover, its application to Clobber is examined and some of its useful properties are highlighted. Section 3.1 explains what Combinatorial Game Theory is and handles some of its basic theory. Section 3.2 explains how game-theoretic values can be calculated.

Chapter contents: Combinatorial Game Theory — Introduction to Combinatorial Game Theory, Calculating Game-Theoretic Values, Implementation

3.1 Introduction to Combinatorial Game Theory

Combinatorial Game Theory applies to games that satisfy the following conditions (Berlekamp *et al.*, 1982):

1. There are exactly two players. Usually, these players are called Left and Right.
2. The two players alternate moves.
3. A game will always come to an end because one of the players cannot move. Either the player who cannot move wins, or the player who makes the last move wins. Games that can end in a draw by repetition of moves do not satisfy this condition.
4. There are no aspects of chance involved, such as dice rolls or shuffled cards.
5. It is a game with perfect information, so all information is available to both players at any point in the game.

A couple of games that do not satisfy these conditions are Chess and Tic-Tac-Toe, since both might end in a draw; Go, since the last move does not decide the winner; and Backgammon and Poker, because these involve chance. Examples of games that do satisfy the conditions are Nim, Amazons, Domineering and Clobber.

For the games that satisfy the conditions, Combinatorial Game Theory aims to predict the advantage either player might have over the other. There are four different types of outcomes possible (Berlekamp *et al.*, 1982):

1. Left wins the game, no matter who moves next.
2. Right wins the game, no matter who moves next.
3. The first player who moves wins.
4. The second player who moves wins.

The theory tries to give a value for the outcome, which comes down to measuring the advantage for either player. This value is positive if Left has an advantage and negative if Right does.

3.2 Calculating Combinatorial Game-Theoretic Values

In order to explain the basic rules, in Subsection 3.2.1, the game of Hackenbush is used. In Hackenbush it is fairly easy to show the reasoning behind the combinatorial game-theoretic values. Moreover, the values explained in Subsection 3.2.1 do never occur in Clobber. However, they need to be explained to understand the values that do occur in Clobber. Subsection 3.2.2 explains these values, using Clobber examples. Finally, some positions need to be simplified before they can be solved, which is discussed in Subsection 3.2.3. The theory in this section is based on the publication by Berlekamp *et al.* (1982).

3.2.1 Basic Rules

An example of a Hackenbush position is shown in Figure 3.1a. The game is played by two players, called Left and Right, who alternate moves. A move is made by removing an edge from the game. After an edge is removed, all remaining edges that are not connected to the baseline (i.e. the dashed line) anymore are removed as well. For example, removing the middle edge of the leftmost chain in Figure 3.1a results in the position shown in Figure 3.1b. Left is only allowed to remove black edges, while Right may only take gray edges. The game ends when a player cannot move anymore. That player loses the game.

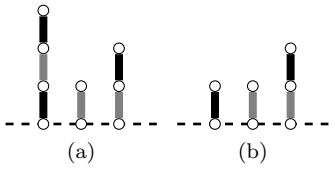


Figure 3.1: An example of a Hackenbush position.

The easiest Hackenbush game is one where all edges are connected directly to the baseline, like in Figure 3.2a. In this position, the players simply continue removing edges, until one player has no edges left. In this particular game, Left will always win, since he has more edges. More specifically, he wins with $3 - 2 = 1$ spare moves. The value of the position is therefore 1. Similarly, Right can win the game in Figure 3.2b with 2 spare moves, so the value of this position is -2, since this time Right has the advantage. The position in Figure 3.2c is somewhat different. It has a value of $2 - 2 = 0$. This means that whichever player moves first runs out of edges the quickest, and thus loses the game. This kind of position is called a *zero-position* and is always a second-player win.

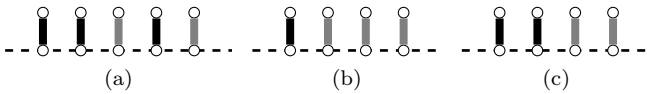


Figure 3.2: Some easy Hackenbush positions.

Of course, not all Hackenbush positions are this straightforward. In the previous examples, the games were made significantly easier to calculate by making sure that no edges were connected, thereby preventing that any extra edges were removed in addition to the ones that the players themselves removed. However, what happens in the position given in Figure 3.3a? If Left moves first he will remove the black edge and win. If Right starts, he will remove the gray edge. Left then takes the black edge and wins. It is clear that Left wins this position, but simply counting moves does not hold anymore, because by that method the value of the game would be $1 - 1 = 0$, which is not the case.

In Figure 3.3b an extra gray edge is added, in an effort to balance the position. The single gray edge has a value of -1, so if the whole position now has a value of 0, Left must have a 1-move advantage in Figure 3.3a. If Left starts the game he must remove the one black edge. Right then has an edge left and wins the game. If Right starts he may either remove the top edge of the left chain or the single edge in the right chain. If he does the latter, Left will then remove the black edge and win the game. If he takes the other edge, however, Left must then remove the black edge and Right wins the game, because he has an edge left. It turns out that now Right can always win the game, so left cannot have a whole move advantage in Figure 3.3a. The value of the game must therefore be somewhere between 0 and 1.

In another effort to balance the position, in Figure 3.3c another chain like the one in Figure 3.3a is added. Now, if this position is a zero-position, it would mean that Left has exactly half a move advantage in Figure 3.3a. In this new position, if Left moves first he can remove either one of the black edges. Right can best counter this move by removing the top edge of the second chain, as became clear in the previous example. Left then must take the left edge and loses the game, because Right has an edge left. If Right starts, he can take the one edge in the right chain, but that leaves two chains of which it is already known that Left wins. If Right takes one of the top edges from the left two chains, Left can counter by taking the bottom edge from the same chain. In that case it is already known that Right will win the resulting game. If Left counters by taking the bottom edge of the other chain, a zero game remains and Left will win the position. Figure 3.3c turns out to be a zero-game. Therefore, Left must have a half move advantage in the position in Figure 3.3a, so the value of the position is $\frac{1}{2}$.

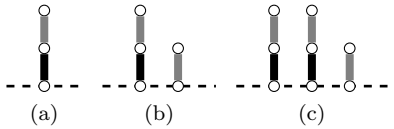


Figure 3.3: Winning by only a fraction.

Similarly, there are positions with even smaller values. Figure 3.4a shows such a position. It turns out that two of these chains are balanced out by half a move for Right, so Figure 3.4a must be worth a quarter move. It is easy to see that Figure 3.4b is in fact a zero-position.

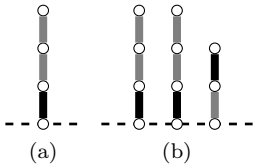


Figure 3.4: A Hackenbush position with value $\frac{1}{4}$.

Figure 3.5 gives a Hackenbush position with value $3\frac{1}{2}$, because the left chain has a value of 3 and the right chain is worth $\frac{1}{2}$. A different notation of this value is $\{3|4\} = 3\frac{1}{2}$, since Left can move to a position worth 3 and Right can move to a position worth 4. More generally, for every n it holds that $\{n|n+1\} = n + \frac{1}{2}$.

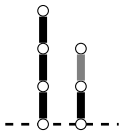


Figure 3.5: A Hackenbush position with value $3\frac{1}{2}$.

The notation $\{a, b, c, \ldots | d, e, f, \ldots\}$ is called the *canonical form* of a value. It represents a position from which Left can move to positions with values $a, b, c, \ldots$ and Right can move to positions with values $d, e, f, \ldots$. These values are called Left's and Right's *options*. An option is different from a move; an option merely denotes the value of a position that a player may move to. The position itself or how to get there does not matter. Therefore, a player may have multiple options with the same value, resulting from different moves. Using this notation, the zero-position is denoted by $0 = \{\ |\ \}$, where neither Left nor Right has any options and the first player to move loses. The whole numbers are given by

$$1 = \{0|\ \}, \quad 2 = \{1|\ \}, \quad \ldots, \quad n+1 = \{n|\ \}.$$

Similarly, if Right has a whole move advantage it is denoted by

$$-1 = \{\ |0\}, \quad -2 = \{\ |-1\}, \quad \ldots, \quad -(n+1) = \{\ |-n\}.$$

A general notation for values with half moves was already given. A few simple examples, including negative values, are

$$\ldots, \quad -1\tfrac{1}{2} = \{-2|-1\}, \quad -\tfrac{1}{2} = \{-1|0\}, \quad \ldots, \quad \tfrac{1}{2} = \{0|1\}, \quad 1\tfrac{1}{2} = \{1|2\}, \quad 2\tfrac{1}{2} = \{2|3\}, \quad \ldots$$

A position with value $\frac{1}{4}$ was discussed in Figure 3.4a. In that position Left can move to a position with value 0 and Right to one with value $\frac{1}{2}$, so the value of the position can be denoted as $\frac{1}{4} = \{0|\frac{1}{2}\}$. This can be extended to smaller fractions:

$$\tfrac{1}{2} = \{0|1\}, \quad \tfrac{1}{4} = \{0|\tfrac{1}{2}\}, \quad \tfrac{1}{8} = \{0|\tfrac{1}{4}\}, \quad \ldots$$

Moreover, it is possible to denote any sum of fractions $\frac{1}{2^n}$ in canonical form. Take for example the position in Figure 3.6. It is composed of a chain with value $\{0|1\} = \frac{1}{2}$ and one with value $\{0|\frac{1}{4}\} = \frac{1}{8}$, so the whole position is worth $\frac{5}{8}$. For a detailed explanation of how these values can be denoted in terms of Left's and Right's options, see Berlekamp *et al.* (1982).

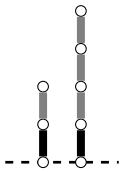


Figure 3.6: A Hackenbush position with value $\frac{5}{8}$.

3.2.2 Infinitesimals

So far, positive (Left always wins), negative (Right always wins) and zero-positions (second player wins) were discussed. However, there is one more possibility, namely the one where the first player always wins. Such a position is called a *fuzzy* position. Figure 3.7 shows the simplest fuzzy position in Clobber, where Left is the black player and Right is the white player. The value of this position is $\{0|0\} = \star$ (pronounced 'star'). It is easy to see that both players move to a zero-position, winning the game.

Figure 3.7: The simplest fuzzy position in Clobber.

The value $\star$ is an example of a *nimber*. Nimbers occur when both Left's and Right's options are nimbers as well. It is important to note that the value 0 is also a nimber, and can thus be written as $\star 0$. Similarly, $\star$ can be written as $\star 1$. The value of a nimber is the easiest to see when Left's and Right's options are equal and of the form $0, \star, \star 2, \ldots, \star n$. The value of this position is then $\star(n+1)$. So

$$\begin{aligned} \star 0 &= \{\ |\ \}, \\ \star 1 &= \{0|0\}, \\ &\vdots \\ \star 4 &= \{0, \star 1, \star 2, \star 3|0, \star 1, \star 2, \star 3\}. \end{aligned}$$

When Left's and Right's options are not consecutive, but still equal and all nimbers, the value of this position is the first absent nimber. So

$$\{0, \star, \star 3, \star 5|0, \star, \star 3, \star 5\} = \star 2.$$

To see why this is true, consider the situation that one of the players would move to $\star 3$. The other could simply reverse that move by moving back to $\star 2$, since $\star 3 = \{0, \star, \star 2|0, \star, \star 2\}$. Therefore, the moves to $\star 3$ and $\star 5$ are of no use. This is referred to as the *mex rule* (i.e. minimal-excluded rule).

In a fuzzy position the value of the game is not known exactly. Therefore, it is not possible to compare fuzzy games. It is, however, possible to sum fuzzy games. Nimbers cannot be summed like regular numbers (i.e. $\star 2 + \star 3 \neq \star 5$), but rather using the *nim-sum* of the nimbers. That is, the bitwise exclusive disjunction of the numbers corresponding to the nimbers, so the nim-sum of $\star 2$ and $\star 3$ is calculated as follows:

$$\begin{array}{rcccc} 2 & = & 1 & 0 & \\ 3 & = & 1 & 1 & \\ \hline & & 0 & 1 & = \quad 1 \end{array}$$

So $\star 2 + \star 3 = \star$.

The most important feature of the nim-sum is the fact that $\star + \star = 0$, which means that star-positions cancel each other out. Star-positions occur far more often than any other nimber in Clobber, especially in smaller subgames. The fact that they cancel each other out means that a move in one star-position can easily be countered by moving in another star game, which often makes playing out the game less complicated.

Although in Clobber most nimbers $\star n$ are positions with $n = 1$, there are positions for which $n > 1$. One such example is shown in Figure 3.8. The value of this position is $\star 2$ (Albert *et al.*, 2005). The calculation of this value, however, is complicated and is therefore omitted here.

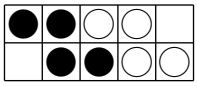


Figure 3.8: A Clobber position with value $\star 2$.

Table 3.1 gives an overview of all possible combinations for summing two types of games G and H. The entry '?' that occurs for some combinations does not mean that they cannot be summed, however, the type of outcome depends on the exact values of G and H. Furthermore, for the entries fuzzy/positive and fuzzy/negative, in order for the sum of G and H to be positive or negative, the positive or negative value must be sufficiently large to make sure the sum is outside the fuzzy area. How large these values should be is discussed later in this section.

		H			
		positive	negative	0	fuzzy
G	positive	positive	?	positive	fuzzy/positive
	negative	?	negative	negative	fuzzy/negative
	0	positive	negative	0	fuzzy
	fuzzy	fuzzy/positive	fuzzy/negative	fuzzy	?

Table 3.1: Sums of games G and H.

Next, consider the Clobber position in Figure 3.9. On the one hand, Left (black) has one move, to a zero-position. Right (white), on the other hand, can only move to a star-position. The resulting value can therefore be denoted as $\{0|\star\}$. Previously it was shown that the value of a position of the form $\{0|x\}$ must be somewhere between 0 and x. Although the value of $\star$ is not exactly known, it must be smaller than any value $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, ..., since $\star = \{0|0\}$. The position $\{0|\star\}$ must in turn also be smaller than any positive number, but, unlike $\star$, this position can always be won by Left; his move to a zero-position wins the game. If Right starts he must move to a star-position, from which Left can again move to a zero-position and win. As a result, $\{0|\star\}$ must be positive. This value is denoted by $\uparrow$ (pronounced 'up'). Similarly, the position $\{\star|0\}$ has value $\downarrow$ (pronounced 'down'), which is a negative, but greater than any negative value.

The values $\uparrow$, $\downarrow$ and $\star$ are called *infinitesimals*, because they are all smaller than any positive number and greater than any negative number. In Clobber, the value of any position exists exclusively of sums of infinitesimals, since in any non-terminal position both players have at least one move. Such a game is called *all-small* (Berlekamp *et al.*, 1982).

Figure 3.9: A Clobber position with value $\{0|\star\} = \uparrow$.

Even though $\uparrow$ and $\downarrow$ do not have exact values, they can be summed, so $\uparrow + \uparrow$ is equal to $2\cdot\uparrow$ (which from now on will be referred to as $\Uparrow$), and $\Uparrow + \uparrow = 3\cdot\uparrow$, etc. Furthermore, these sums of ups can be compared. For instance, $\Uparrow$ is greater than $\uparrow$. Moreover, $\downarrow$ is equal to $-\uparrow$, so a sum of ups and downs can be denoted in terms of only ups. Thus, the sum $\uparrow + \downarrow$ equals 0.

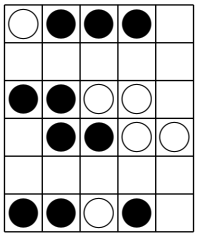


Figure 3.10: A Clobber position with a total value of $\uparrow \star 2$.

Sums of ups and nimbers can also be summed themselves. The ups and the nimbers are then added separately. For example, the position in Figure 3.10 consists of three separate subgames. From top to bottom, these subgames have values of $\Uparrow \star$, $\star 2$ and $\downarrow \star$. Therefore the sum of these games is as follows:

$$\begin{aligned} &\Uparrow \star + \star 2 + \downarrow \star \\ &= \Uparrow + (-\uparrow) + \star + \star 2 + \star \\ &= \uparrow \star 2. \end{aligned}$$

Earlier, it was noted that the sum of a positive or negative value and a fuzzy value is sometimes positive/negative and sometimes fuzzy. This depends on the number of ups, as well as the size of the fuzzy area, which in turn depends on the nimber. For the nimber $\star$, the fuzzy area lies between $\downarrow$ and $\uparrow$. That means that the value of $\star$ is somewhere between $\downarrow$ and $\uparrow$, but the exact value is not known. Therefore, a value $n\cdot\uparrow + \star$ is positive if $n > 1$, negative if $n < -1$ and fuzzy if $n = -1$, 0 or 1 (e.g. $\uparrow \star$ is fuzzy, but $\Uparrow \star$ is positive). For the nimber $\star k$, with $k > 1$, the fuzzy area lies around 0. In this case, the value $n\cdot\uparrow + \star k$, with $k > 1$, is positive if $n > 0$, negative if $n < 0$ and fuzzy if n is exactly 0 (e.g. $\star 2$ is fuzzy, but $\uparrow \star 2$ is positive).

An overview of the simplest sums of infinitesimals and their corresponding canonical forms (Berlekamp *et al.*, 1982) is given in Table 3.2. In the third column $\star n$ is a nimber with $n > 1$, and $\star m = \star n + \star$.

$3\cdot\downarrow = \{\Downarrow \star \mid 0\}$	$3\cdot\downarrow + \star = \{\Downarrow \mid 0\}$	$3\cdot\downarrow + \star n = \{\Downarrow \star m \mid 0\}$
$\Downarrow = \{\downarrow \star \mid 0\}$	$\Downarrow \star = \{\downarrow \mid 0\}$	$\Downarrow \star n = \{\downarrow \star m \mid 0\}$
$\downarrow = \{\star \mid 0\}$	$\downarrow \star = \{0 \mid 0, \star\}$	$\downarrow \star n = \{\star m \mid 0\}$
$0 = \{\ \mid\ \}$	$\star = \{0 \mid 0\}$	
$\uparrow = \{0 \mid \star\}$	$\uparrow \star = \{0, \star \mid 0\}$	$\uparrow \star n = \{0 \mid \star\, m\}$
$\Uparrow = \{0 \mid \uparrow \star\}$	$\Uparrow \star = \{0 \mid \uparrow\}$	$\Uparrow \star n = \{0 \mid \uparrow \star m\}$
$3\cdot\uparrow = \{0 \mid \Uparrow \star\}$	$3\cdot\uparrow + \star = \{0 \mid \Uparrow\}$	$3\cdot\uparrow + \star\, n = \{0 \mid \Uparrow \star m\}$

Table 3.2: The simplest sums of infinitesimals and their corresponding Left and Right options.

3.2.3 Simplifying Positions

Of course, not all positions are of such simple forms as given in Table 3.2. Often Left and Right will have more than one option. In that case, the options can often be simplified, so that the position can be solved nevertheless. We discuss three ways of simplifying positions below.

The first way to simplify a position is to remove all duplicate options the players may have. Take for example the position in Figure 3.11 with canonical form $\{0, 0|\uparrow, \uparrow\}$, where Left has two different moves that lead to a zero-position and Right has two different moves that lead to a position with value $\uparrow$. It does not matter which of those they choose, the resulting value will be the same, so the position can be simplified to $\{0|\uparrow\} = \Uparrow \star$.

Figure 3.11: A Clobber position with duplicate options.

A second way of simplifying a position is to remove all *dominated* options. An option dominates another if it is at least as good for the player whose option it is. That is, in the position $\{a, b, c, \ldots|d, e, f, \ldots\}$, a dominates b if $a \geq b$, and d dominates e if $d \leq e$. If the difference $a - b$ is positive, then $a \geq b$ holds. Similarly, $d \leq e$ holds if the difference $d - e$ is negative. If the difference is 0 or fuzzy, neither option dominates the other.

Figure 3.12 shows a Clobber position for which calculating the value requires removing a dominated option. The canonical form of this position is $\{0, \downarrow|0\}$. Since $0 - \downarrow = 0 - (-\uparrow) = \uparrow$ is positive, 0 dominates $\downarrow$ for Left. Therefore, the value can be rewritten as $\{0|0\} = \star$.

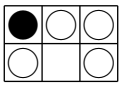


Figure 3.12: A Clobber position with dominated options.

The third way a position can be simplified is by bypassing *reversible* options. If Left has a reversible option, it means that if Left chooses that particular option, from the resulting position Right can move to a position that is at least as favorable for him as the original position. Left's reversible option might then as well be replaced with the options he has from this new position, since if he plays the reversible option he will end up with those options anyway, because Right will always choose the option that is more favorable for him. The advantage of bypassing a reversible option is that the new options are always of a simpler form than the original option, since a position's options are always simpler than its value (see Table 3.2).

Consider, for example, the position in Figure 3.13. It is already known that the value of this position is $\Uparrow$, since it consists of two subgames $\circ\bullet\bullet$ with value $\uparrow$, which can simply be summed. However, the actual calculation of the value of this position requires bypassing reversible options.

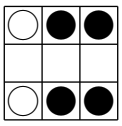


Figure 3.13: A Clobber position with reversible options.

Figure 3.14 shows a part of the game tree based on the position in Figure 3.13. Its canonical form is $\{\uparrow, \uparrow \mid \uparrow \star, \uparrow \star\}$, which is not in Table 3.2, so the value is unknown. However, it turns out that Left's options are reversible and can be bypassed. If Left chooses to play option a, Right can counter this move

by playing option a^R. Position a^R is better for Right than the original position, since it has value $\star$, while his options d and e from the original position both have value $\uparrow\star$. It is easy to see that $\star$ is better for Right than $\uparrow\star$, since $\star$ dominates $\uparrow\star$ for him (i.e. $\star - \uparrow\star = \downarrow$ is a negative value). Therefore, Left's option a can be bypassed by replacing it with Left's options from position a^R, which in this case is just a^{RL}. The new canonical form of G then becomes $\{0, \uparrow \mid \uparrow\star, \uparrow\star\}$. The value of this position is still unknown, however, it turns out that Left's option b can be bypassed in a similar manner, resulting in the canonical form $\{0, 0 \mid \uparrow\star, \uparrow\star\}$ for G. After removing duplicate options the new value of G is $\{0 \mid \uparrow\star\} = \Uparrow$.

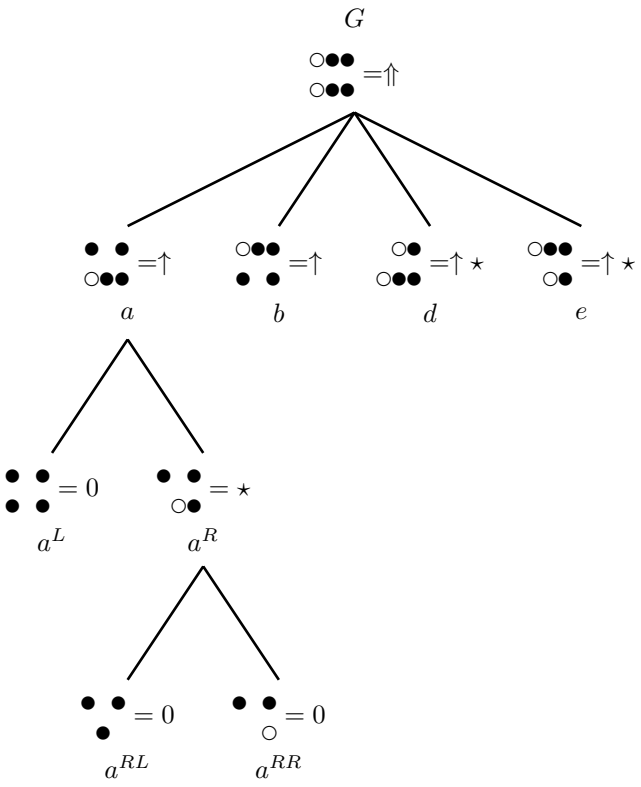


Figure 3.14: Bypassing a reversible option.

3.3 Implementation

This section discusses how the theory in the previous section is implemented in order to build a database of Clobber positions and their values. Firstly, in order to calculate the value of a Clobber position, the separate subgames need to be identified. After the subgames are known, the value of each subgame can be calculated. Summing all these values gives the total value of the complete game. Calculating a single subgame consists of the following six steps:

1. Find all moves for both players.
2. For each move, find the value of the resulting position, obtaining lists of Left's and Right's options (i.e. the canonical form of the subgame).
3. Remove duplicate options.
4. Bypass reversible options.
5. Remove dominated options.
6. Try to solve the position using
    - The mex-rule, or

- The rules given in Table 3.2.

In step (2), the value of the resulting positions can either be found by solving each of them in a recursive manner or, more preferably, by looking them up in a database (see Chapter 4 for more information about the database). Steps (3) to (5) are then applied to both Left's and Right's list of options, resulting in the simplest possible form. As long as these steps are performed in this particular order, no new duplicate or reversible options can occur after performing any of the steps (Siegel, 2005). Pseudo-code for bypassing Left's reversible moves in step (4) is given in Algorithm 1. Right's reversible moves can be bypassed in a similar manner. Algorithm 2 gives pseudo-code for determining whether one option dominates another, which is required for Algorithm 1, as well as for step (5). Note that in lines 6 and 8 a value only dominates another if the difference is $\uparrow \star k$ and $\downarrow \star k$, respectively, where $k \neq 1$, since these values are fuzzy for $k = 1$, but not for any other value of k, as explained in Subsection 3.2.2.

Algorithm 1 Bypass Left's reversible options.

Input: $left\_options, right\_options$
1: **for all** $optionL$ such that $optionL \in left\_options$ **do**
2: $lr\_options \leftarrow get\_right\_options(optionL)$
3: **for all** $optionLR$ such that $optionLR \in lr\_options$ **do**
4: $is\_reversible \leftarrow$ **true**
5: **for all** $optionR$ such that $optionR \in right\_options$ **do**
6: **if** $optionsLR \neq optionR$ **and not** $dominates(optionLR, optionR) > 1$ **then**
7: $is\_reversible \leftarrow$ **false**
8: **end if**
9: **end for**
10: **if** $is\_reversible$ **then**
11: replace $optionL$ with all $optionLRL \in get\_left\_options(optionLR)$
12: **break**
13: **end if**
14: **end for**
15: **end for**
Output: $left\_options$

Algorithm 2 The *dominates* method. For Left it returns 1 if a dominates b and -1 if b dominates a. For Right it returns -1 if a dominates b and 1 if b dominates a. For both players it returns 0 if a and b are equal or incomparable.

Input: a, b
1: $dominates \leftarrow 0$
2: **if** $ups(a) - ups(b) \geq 2$ **then**
3: $dominates \leftarrow 1$
4: **else if** $ups(a) - ups(b) \leq -2$ **then**
5: $dominates \leftarrow -1$
6: **else if** $ups(a) - ups(b) \geq 1$ **and** $nimsum(nim(a), nim(b)) \neq 1$ **then**
7: $dominates \leftarrow 1$
8: **else if** $ups(a) - ups(b) \leq 1$ **and** $nimsum(nim(a), nim(b)) \neq 1$ **then**
9: $dominates \leftarrow -1$
10: **end if**
Output: $dominates$

Since Table 3.2 and the mex-rule only give solutions for a limited set of forms, not all values can be calculated. For example, Figure 3.15 shows a position that cannot be solved. The canonical form of this position is $\{\star, 0, \uparrow \mid 0, \uparrow, \downarrow\}$, which can be simplified to $\{\star, \uparrow \mid \downarrow\}$, but even the simplified value is not in the table.

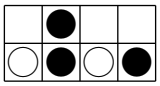


Figure 3.15: A Clobber position that cannot be solved.

Moreover, in our implementation a value of which one or more of Left's and Right's options cannot be calculated is considered to be impossible to calculate. However, it is in fact often possible to calculate such a value, as implemented by Siegel (2007), who allows an option in the canonical form $\{a, b, \dots | d, e, \dots\}$ to be in canonical form itself, if the exact value cannot be computed. For example, option b could be of the form $\{u, v, \dots | x, y, \dots\}$, resulting in a position that looks like $\{a, \{u, v, \dots | x, y, \dots\}, \dots | d, e, \dots\}$. By simplification, the unsolvable position might be removed, so that the value of the position can still be calculated. Algorithm 3 gives an algorithm for comparing regular values as well as canonical forms (Siegel, 2005). However, allowing canonical forms as options requires a different approach for bypassing the reversible moves than the one explained in Section 3.2. Since applying this approach in combination with the reversible options technique would complicate our implementation, we omitted Algorithm 3 entirely in our research.

Algorithm 3 The *lEq* method. Compares values a and b and returns true if $a \leq b$. The values can either be of the form $n\cdot\uparrow + \star k$ or $\{a, b, \dots | d, e, \dots\}$.

Input: a, b

if a equals b **then**
  return true
else if a and b are both of the form $n\cdot\uparrow + \star k$ **then**
  if $ups(b) - ups(a) \geq 2$ **then**
    return true
  else if $ups(b) - ups(a) \geq 1$ **and** $nimsum(nim(a), nim(b)) \neq 1$ **then**
    return true
  else
    return false
  end if
else if b is of the form $n\cdot\uparrow + \star k$ **then**
  for all *optionL* such that *optionsL* is a left option of a **do**
    if not $lEq(optionL, b)$ **then**
      return false
    end if
  end for
else if a is of the form $n\cdot\uparrow + \star k$ **then**
  for all *optionL* such that *optionsL* is a left option of b **do**
    if not $lEq(a, optionL)$ **then**
      return false
    end if
  end for
else
  return true
end if

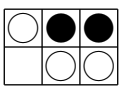


Figure 3.16: A Clobber position that can be solved by Siegel's Combinatorial Game Suite, but not by our implementation.

An example of a position that can be solved by Siegel's implementation, but not by ours is given in Figure 3.16. The canonical form of this position is $\{\star, \{\uparrow, \star | \downarrow, \star\} | \Downarrow \star, \downarrow, \star\}$. Since one of the options has an unknown value, in our implementation we consider this whole position impossible to solve. However, solving the position with Siegel's implementation, it turns out that the value of this position is 0. Note that the position given in Figure 3.15 which we are unable to solve cannot be solved by Siegel's implementation either.

Chapter 4

Database

THIS chapter describes the database that is used to store the values of the smaller Clobber positions. In order to be useful, the values should be retrieved in a fast manner. Section 4.1 explains the architecture of the database, as well as how the database is filled. Section 4.2 examines the contents of the database, after it was built.

Chapter contents: Database — Building the Database, Contents of the Database

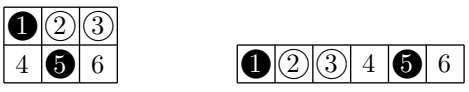
4.1 Building the Database

The database is built with speed in mind. Therefore, the contents are saved in memory, so that they are quickly accessible. For several board sizes, all positions were calculated. The values of each board size were saved in separate tables, in which each position could be accessed using a unique index.

The index of a board position is calculated by representing each square on the board as a *trit* (the ternary equivalent of a bit in the binary system) in the ternary numeral system (Hayes, 2001), where an empty square has value 0, a square with a black piece has value 1, and a square with a white piece has value 2. Consider the example in Figure 4.1a, where the squares are numbered 1–6. The first square is represented by the rightmost trit, the second square by the second trit from the right, etc., so that the ternary value of the position in Figure 4.1a is 010221. The equivalent decimal value of a board position can easily be found using the equation $3^0 \times square_1 + 3^1 \times square_2 + \cdots + 3^{n-1} \times square_n$. Hence, the decimal value of the position in Figure 4.1a is

$$3^0 \times 1 + 3^1 \times 2 + 3^2 \times 2 + 3^3 \times 0 + 3^4 \times 1 + 3^5 \times 0 = 106.$$

Note that this index is unique for the board size 2×3, however, there are positions with different board sizes that have the same index. For instance, it is easy to see that the 1×6 position in Figure 4.1b also has index 106, while it is not equivalent to the one in Figure 4.1a. However, since the values of different board sizes are stored in separate tables this is not an issue.



(a) (b)

Figure 4.1: The 2×3 and 1×6 Clobber positions with index 106.

For each board size the table is filled using a retrograde analysis approach (Ströhlein, 1970). The values of the smallest board sizes are computed first, so that these values can be accessed fast when calculating the values of larger board sizes. For each board size, we scan through all possible board configurations repeatedly. For each configuration the value is calculated using the database entries for

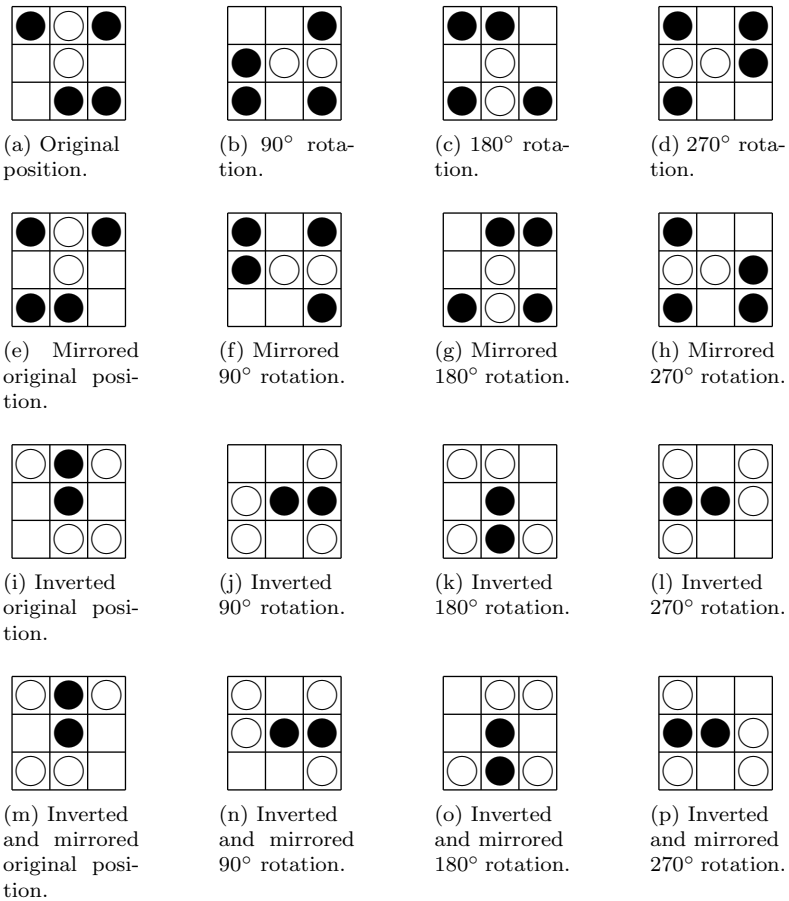
smaller board sizes as well as the values of positions with the same board size that are already calculated. If calculating a value requires the value of a position with the same size that is not calculated yet, this value is skipped and reconsidered in the next scan. This process terminates when no value has changed during the last scan. This way, for each position either the value is obtained or it is found that we are not able to calculate the value.

In order to speed up building the database, only $n \times m$ boards for which $m \geq n$ are considered. The values of $n \times m$ boards with $n > m$ can be looked up in the database by simply rotating the position by 90°, since this position has the same value. Moreover, the fact that equivalent positions have the same value is used. Therefore, whenever the value of a position is found, the equivalent positions can be updated as well. Three types of equivalence are considered.

Firstly, rotating a position does not change the value, as mentioned earlier, so a position is equivalent to its 180° rotation. For $n \times m$ positions with $n = m$, the 90° and 270° rotations can also be updated. For positions with $n \neq m$ this does not hold, since these positions are not in the database, as mentioned before. Figures 4.2a–4.2d depict a 3×3 board position and its rotations.

Secondly, for all these rotations the mirrored positions are also equivalent to the original position. Only horizontally mirrored positions are considered, since the vertically mirrored position is the same as the vertical mirroring of the 180° rotated position. The mirrored positions of the positions in Figures 4.2a–4.2d are shown in Figures 4.2e–4.2h.

Thirdly, for all the rotated and mirrored positions, the inverse position is considered. That is, the position for which all black pieces are replaced by white pieces and all white pieces are replaced by black pieces. The inverse of a position does not have the same value, however, the value of the original position can simply be negated to obtain the value of the inverse position. The inverse positions of Figures 4.2a–4.2h are depicted in Figures 4.2i–4.2p.



(a) Original position. (b) 90° rotation. (c) 180° rotation. (d) 270° rotation.

(e) Mirrored original position. (f) Mirrored 90° rotation. (g) Mirrored 180° rotation. (h) Mirrored 270° rotation.

(i) Inverted original position. (j) Inverted 90° rotation. (k) Inverted 180° rotation. (l) Inverted 270° rotation.

(m) Inverted and mirrored original position. (n) Inverted and mirrored 90° rotation. (o) Inverted and mirrored 180° rotation. (p) Inverted and mirrored 270° rotation.

Figure 4.2: All equivalent positions of a 3×3 position.

4.2 Contents of the Database

In order to get an idea of how many positions could be solved for each board size, the contents of the database were investigated. Table 4.1 shows the percentage of positions that was solved for all the board sizes that were considered. For the smaller board sizes all games can still be solved, but as the boards get larger, the percentages quickly drop. It should be noted that the percentages drop considerably faster when the numbers of rows and columns are both increased, whereas positions with size $1 \times n$ can often still be solved, even for larger values of n. It is also worth noting that for board size 1×4, there are 2 (equivalent) out of 81 positions that could not be solved. One of these positions is shown in Figure 4.3, the other one that could not be solved is the 180° rotation of the one shown. After simplification, these positions have a canonical form of $\{\uparrow, \star | \downarrow, \star\}$, which cannot be solved.

rows \ columns	1	2	3	4	5	6	7	8
1	100%	100%	100%	97.5%	94.2%	89.8%	86.9%	83.8%
2	100%	100%	81.6%	63.1%	50.6%	41.0%	33.0%	26.5%
3	100%	81.6%	50.6%	31.5%	20.7%			
4	97.5%	63.1%	31.5%	16.3%				
5	94.2%	50.6%	20.7%					
6	89.8%	41.0%						
7	86.9%	33.0%						
8	83.8%	26.5%						

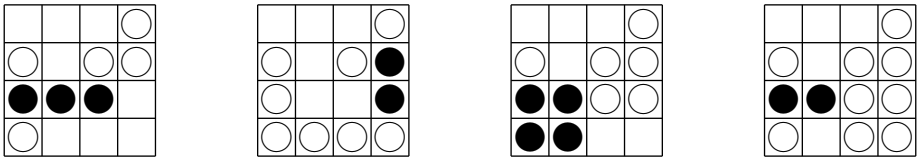
Table 4.1: The percentages of positions that were solved for each board size.

Figure 4.3: One of the 1×4 positions that could not be solved.

As the board size increases, often the positions consist of several smaller subgames. Because these subgames are already in the database, it is interesting to see how many *non-decomposable* positions could be solved. That is, a position that cannot be decomposed into subgames, touches all four borders and has at least one black and one white piece. Table 4.2 shows the percentages of non-decomposable positions that were solved for all board sizes. Obviously, the percentages now drop far more rapidly as the board size increases and for positions with size 4×4 only a small percentage of the positions can be solved. A couple of such positions is shown in Figure 4.4. Usually these positions have chain-like structures with relatively few possible moves. Two of these chain-like structures are depicted in Figures 4.4a and 4.4b. Some examples of solvable 4×4 positions that do not have a chain-like structure are given in Figures 4.4c and 4.4d. In positions with blocks of pieces, such as these, the blocks usually exist mostly of pieces of the same color, so that still not many moves are possible.

rows \ columns	1	2	3	4	5	6	7	8
1	100%	100%	100%	86.7%	67.7%	36.5%	30.7%	18.4%
2	100%	100%	62.3%	27.2%	10.8%	4.3%	1.6%	0.6%
3	100%	62.3%	19.6%	4.6%	1.2%			
4	86.7%	27.2%	4.6%	0.8%				
5	67.7%	10.8%	1.2%					
6	36.5%	4.3%						
7	30.7%	1.6%						
8	18.4%	0.6%						

Table 4.2: The percentages of non-decomposable positions that were solved for each board size.



(a) A 4×4 position with value $\star$.

(b) A 4×4 position with value $5 \cdot \downarrow + \star$.

(c) A 4×4 position with value $\uparrow$.

(d) A 4×4 position with value $3 \cdot \downarrow + \star$.

Figure 4.4: A couple of 4×4 positions that can be solved.

Chapter 5

Monte-Carlo Tree Search

In this chapter Monte-Carlo Tree Search (MCTS) is discussed, of which we aim to improve the performance in this research. MCTS builds a tree based on repeated random simulations. Based on the information in this tree the most promising move can be selected for a certain board position. Section 5.1 discusses general Monte-Carlo methods. Section 5.2 explains MCTS in more detail. Section 5.3 introduces MCTS-solver, an enhancement of MCTS. In Section 5.4 two approaches are proposed for improving the performance of MCTS by applying combinatorial game theory.

Chapter contents: Monte-Carlo Tree Search — Monte-Carlo Methods in Games, Monte-Carlo Tree Search, MCTS-Solver, MCTS and Combinatorial Game Theory

5.1 Monte-Carlo Methods in Games

The use of Monte-Carlo methods was first introduced by Abramson (1990), who applied it to Othello, Tic-Tac-Toe and Chess. A couple of years later, it was also applied to Go by Brügmann (1993). Since the results were unimpressive, the idea was not taken seriously at the time. However, the idea was revived by Bouzy and Helmstetter (2003) and Monte-Carlo methods have become increasingly popular since. Monte-Carlo methods were successfully applied to games such as Go (Gelly and Silver, 2008), Scrabble (Sheppard, 2002) and Poker (Billings *et al.*, 2002).

Remarkably, Monte-Carlo methods can achieve relatively high levels of gameplay in games that require a strategic approach, such as Go, without incorporating much domain knowledge. For such games it is hard to define a good evaluation function, however, Monte-Carlo methods do not require an evaluation function. Instead, many random games are played. Moves that often lead to good results during these random games are considered to be good moves.

Domain knowledge can be used to further improve the gameplay, as long as it is done in a balanced way. For example, weights can be assigned to the moves, giving moves that are usually good higher weights. This way, a better exploration-exploitation tradeoff can be achieved. That is, all moves are 'explored' thoroughly enough so that a good idea of the value of each move is developed. Then, the most promising moves are further 'exploited,' such that a better approximation of the value of these moves is obtained. However, one must be careful when incorporating domain knowledge, since it could go at the expense of randomness and therefore the whole idea of Monte-Carlo methods.

In an attempt to improve the performance of Monte-Carlo methods, a combination of tree search and Monte-Carlo simulations was proposed by Bouzy (2006). His approach involved growing a tree as the simulations were performed and pruning the least promising nodes, based on the information of previous simulations. However, this might also prune good moves that are approximated to be less promising, due to the randomness of Monte-Carlo evaluations. In order to solve this problem MCTS was proposed. MCTS does not prune less promising moves, but rather gives them a low priority. MCTS is explained in more detail in the next section.

5.2 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) (Chaslot *et al.*, 2006; Kocsis and Szepesvári, 2006; Coulom, 2007) grows a tree as it randomly simulates games from a given position, exploring the possible moves of future game positions. This tree holds information about previous simulations (e.g. the value of a move, the visit count of a move). The stored information differs depending on the MCTS variant that is used. As more simulations are done, a more accurate approximation of the value of a move is obtained. The information in the tree influences the first few moves of a simulated game, in order to get a more efficient approach in which promising moves are explored more often, while still exploring less promising moves as well. MCTS consists of four steps that are repeated (Chaslot *et al.*, 2008a):

1. **Selection**: the next move to explore is selected according to a selection function. This function selects a move based on the data in the game tree. This must be done in such a way that there is a good exploration-exploitation tradeoff for the moves. The most widely used selection function is the UCT (Upper Confidence bounds applied to Trees) selection function (Kocsis and Szepesvári, 2006). Given the children C of a position p, UCT chooses the move that leads to $c \in C$ that has the highest score according to the following formula:

$$uct(c) = v_c + K \times \sqrt{\frac{\ln n_p}{n_c}},$$

   where v_c is the value of c, n_c is the visit count of c, n_p is the visit count of p, and K is a constant. Several other selection functions have been proposed as well (Chaslot *et al.*, 2008a; Gelly and Silver, 2008).

2. **Expansion**: new nodes are added to the tree whenever a position is encountered that was not already stored in the game tree. Usually, only one position is added to the tree; the first position that was encountered that was not stored in the tree yet. However, it is also possible to add more positions at once. For instance, the children of the first node, up to a certain ply.

3. **Playout**: the game is played out randomly to the end, according to a playout strategy. The easiest way to play out a game is to select moves in a completely random way. However, the performance of MCTS can often be improved by using a more sophisticated playout strategy. Several such strategies have been proposed (Bouzy, 2005; Gelly *et al.*, 2006).

4. **Backpropagation**: all the nodes that were traversed during the first three steps are updated to reflect the outcome of the game. This is done by storing the average value of the node over all node visits, as well as incrementing the number of node visits. Additionally, sometimes other information is stored, which is updated in a manner specific to this information.

An overview of these steps is given in Figure 5.1.

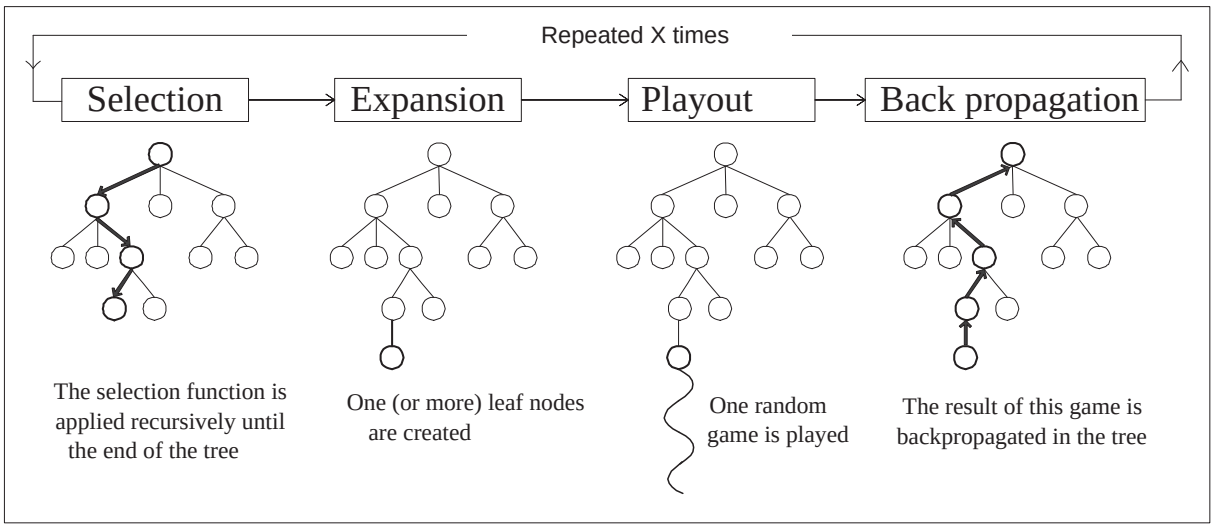


Figure 5.1: An overview of the steps of MCTS (slightly adapted from Chaslot (2008)).

MCTS was successfully applied to several games. In Go, strong programs were usually based on $\alpha\beta$-search. However, MCTS has vastly improved the performance of Go programs (Coulom, 2007; Gelly, 2007; Gelly and Silver, 2008). MCTS programs have dominated Go tournaments ever since the introduction of MCTS, winning every 9×9 Go tournament since 2006, and every 19×19 tournament since 2007 at the Computer Olympiad. Furthermore, MCTS has led to good results in games like Hex (Arneson, Hayward, and Henderson, 2010), Amazons (Lorentz, 2008; Kloetzer, Iida, and Bouzy, 2009; Kloetzer, 2010) and Lines of Action (Winands, Björnsson, and Saito, 2008; Winands and Björnsson, 2010).

5.3 MCTS-Solver

In addition to the standard MCTS algorithm, MC-Mila uses an enhancement called MCTS-solver (Winands, Björnsson, and Saito, 2010). MCTS-solver is able to solve the game-theoretic value of a position and incorporate this value in the search tree, so that it can be used for future simulations. This method was proposed because in some (especially tactical) games MCTS needs too many simulations before it converges to the correct (i.e. winning) move during endgames. Instead an endgame can be solved using $\alpha\beta$-search or combinatorial game theory. In order to use the game-theoretic values to solve a game with MCTS, some modifications are required in the backpropagation and selection steps, as well as the procedure for choosing the final move to play.

1. **Backpropagation**: in regular MCTS terminal nodes in the search tree are marked the same as the result of a simulated game: a win, a loss or a draw. In MCTS-solver, however, the win and loss terminal nodes are marked differently, namely ∞ for a win and $-\infty$ for a loss. These values are called a *proven win* and a *proven loss*. When updating nodes with proven values a few considerations must be made.

   If a proven loss is backpropagated to a parent, this parent is labeled with a proven win, since in the parent node the player can always select the child with a proven loss for the opponent and win the game. However, if a proven win is backpropagated to a parent, the remaining children also need to be considered. The parent can only be marked as a proven loss if all the other children are also proven wins, otherwise in the parent node the player can always choose a move that is not a proven loss for him. In the latter case, where at least one child is not a proven win, the proven wins in other children will be considered regular (i.e. simulation) wins and the parent will be updated as in regular MCTS.

2. **Selection**: in the selection phase the use of proven values can give rise to some problems. More specifically, if a parent node has a non-proven value, but some of its children have proven values, the value of the parent node might be estimated inaccurately. Consider the example in Figure 5.2a, where one of the children of node A is a non-proven value and the rest are proven losses for the parent node. In that case, if the proven loss nodes are pruned, since they will never be considered, it may lead to an overestimation of node A. It would for instance be chosen over node E in Figure 5.2b if all moves are chosen with the same likelihood, even though node E has more good moves than node A.

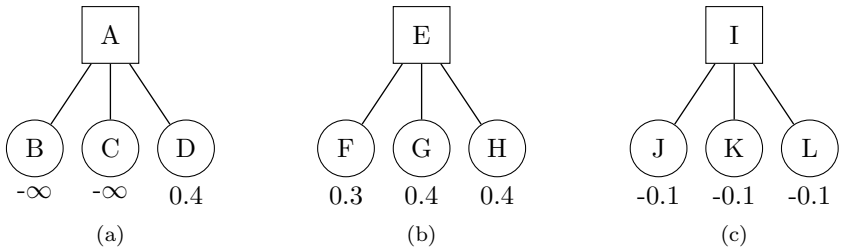


Figure 5.2: Monte-Carlo subtrees (Winands *et al.*, 2010).

However, if the proven loss nodes are not pruned the value of A may be underestimated, since those values have a strong influence on the value of the parent node. Node I in Figure 5.2c may even be considered better than node A, although it has no good moves at all. Therefore, in order to

get a better performance, in the selection phase a proven loss node is never considered, but in the simulation phase it is considered. This strategy was proposed by Winands *et al.* (2010).

3. **Final move selection**: there exist several ways to select the move that is ultimately played by regular MCTS in the actual game. For MCTS-solver a slightly unusual strategy is used, namely *secure child* (Chaslot *et al.*, 2008b), which deals well with the sudden drops and rises that are caused by the backpropagation of proven values. It chooses the child that has the highest score $v + (A)/(\sqrt{n})$, where v is the node's value, n is its visit count, and A is a constant. Moreover, if the root node is a proven win, the winning move from the root node is always chosen.

MCTS-solver can drastically reduce the number of simulations required to select the correct move during endgame play, and thus improve the performance of MCTS, especially in games that require a strategic approach.

5.4 MCTS and Combinatorial Game Theory

Combinatorial game theory can be used to improve the performance of a MCTS program. It can be applied to several stages of the algorithm. We discuss two ways of incorporating combinatorial game theory in MCTS.

Firstly, the playout and selection phases of each MCTS simulated game can be improved by terminating early if a position is solved before the game is played out completely. After each move the new board position is attempted to be solved. If the new position can be solved, it is known whether Black or White can win the game, so the simulation can be terminated. Algorithm 4 shows the algorithm that decides whether a game can be terminated given a certain board position. If so, it also returns the winner of the game. By terminating the simulation early, computational time can be saved, so that more games can be simulated, which results in a better approximation of the value of each possible move. Moreover, by using the MCTS-solver technique, as explained in Section 5.3, the combinatorial game-theoretic values can be incorporated in the search tree, which improves the performance even further. Moreover, endgames can then be played out optimally as soon as the root node is a proven win.

Figure 5.3 shows a board position which is a proven win for Black, assuming that he is the next player to move, since the value of the position is $\uparrow +0+\star+\star+\star+\downarrow+\star+\star+0=\star$. MCTS-solver will pick one of the winning moves, which in this case can be any move, except b8xc8, which leads to a proven loss.

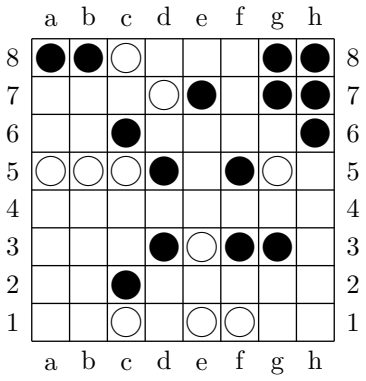


Figure 5.3: A Clobber position that can be played out optimally using combinatorial game theory.

Secondly, combinatorial game theory can be used to simplify the board that MCTS uses to evaluate the moves. If for some subgames the value is known, it is possible that there exist subgames with values that cancel each other out. Two values cancel each other out if their sum is equal to 0. For instance, two positions with values $\uparrow$ and $\downarrow$ cancel each other out, since $\uparrow + \downarrow = 0$. During the selection and playout phases of MCTS any two subgames that cancel each other out can be ignored, so that moves in those subgames are not considered. Moreover, subgames that have a value of 0 themselves need not be considered. In this way, less moves have to be evaluated, so a better approximation of the remaining moves can be obtained. Algorithm 5 gives the algorithm that simplifies the board.

Algorithm 4 The algorithm to determine whether a game can be solved and which player wins the game in that case. It returns *win_for_black* if Black wins the game, *win_for_white* if White wins the game, and 0 if the game could not be solved.

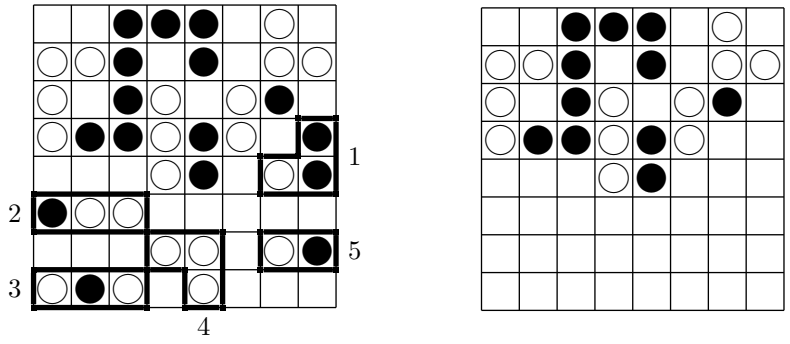
Input: *board*
1: $value \leftarrow solve(board)$
2: **if** *value* is unknown **then**
3: **return** 0
4: **else if** *value* is positive **then**
5: **return** *win_for_black*
6: **else if** *value* is negative **then**
7: **return** *win_for_white*
8: **else if** $value = 0$ **then**
9: **if** Black is next to move **then**
10: **return** *win_for_white*
11: **else if** White is next to move **then**
12: **return** *win_for_black*
13: **end if**
14: **else if** *value* is fuzzy **then**
15: **if** Black is next to move **then**
16: **return** *win_for_black*
17: **else if** White is next to move **then**
18: **return** *win_for_white*
19: **end if**
20: **end if**

Algorithm 5 The simplifyBoard algorithm. It looks for pairs of subgames that cancel each other out. Subgames that cancel each other out are removed from the board. It returns the board after removing these subgames.

Input: *board*
1: $subgames \leftarrow getSubgames(board)$
2: **for** $i \leftarrow 0$ **to** $size(subgames) - 1$ **do**
3: $value1 \leftarrow getValueFromDatabase(subgames[i])$
4: **if** $value1 = 0$ **then**
5: $removeFromBoard(subgame[i])$
6: **continue**
7: **end if**
8: **for** $j \leftarrow i + 1$ **to** $size(subgames)$ **do**
9: $value2 \leftarrow getValueFromDatabase(subgames[j])$
10: **if** $value2 = 0$ **then**
11: $removeFromBoard(subgame[j])$
12: remove $subgames[j]$ from *subgames*
13: **continue**
14: **else if** $value1 + value2 = 0$ **then**
15: $removeFromBoard(subgame[i])$
16: $removeFromBoard(subgame[j])$
17: remove $subgames[j]$ from *subgames*
18: **continue**
19: **end if**
20: **end for**
21: **end for**
Output: *board*

An example of a board that can be simplified is given in Figure 5.4a. The subgames that can be solved are marked 1–5, the rest of the game cannot be solved yet. The values of subgames 1–5 are ↑, ↓,

$\star$, 0 and $\star$, respectively. It is easy to see that subgame 4 can be removed immediately, since it has value 0. Further, subgame 1 and 2 can be removed as well, since the sum of their values is $\uparrow + \downarrow = 0$. Finally, subgames 3 and 5 can be removed, because their values sum up to $\star + \star = 0$ also. This results in the board position shown in Figure 5.4b, in which only 8 moves need to be considered, instead of 13.



(a) (b)

Figure 5.4: A Clobber position for which the board can be simplified and its simplified version.

Experiments were performed to see whether these approaches of incorporating combinatorial game theory in MCTS improve its performance. The results of these experiments can be found in Chapter 6.

Chapter 6

Experiments and Results

In this chapter we describe the experiments that were performed. In these experiments we aimed to improve the performance of MC-Mila *by incorporating combinatorial game theory in the two ways discussed in Section 5.4. Section 6.1 describes the experimental setup that was used. In Section 6.2 the performance of* MC-Mila *with different enhancements is tested against* $\alpha\beta$-Mila*. Section 6.3 discusses the experiments where* MC-Mila *with different enhancements plays against the default* MC-Mila*.*

Chapter contents: Experiments and Results — Experimental Setup, Playing Against an $\alpha\beta$ Player, Playing Against a Regular MCTS player

6.1 Experimental Setup

The following two ways of incorporating combinatorial game-theoretic knowledge in MCTS were tested:

1. **Solved positions**: during the playout and selection phases of MCTS a simulated game is terminated early if a position can be solved before a terminal position is reached, thus allowing more random simulations, which should result in a better approximation of the values of moves. Moreover, the value is incorporated in the tree using MCTS-solver.

2. **Simplifying the board**: during the playout phase of MCTS some subgames are removed from the board if they cancel each other out. As a result, fewer moves need to be considered, which should lead to a better approximation of the remaining moves.

In each experiment MC-Mila enhanced with knowledge of combinatorial game theory played 1000 games against either $\alpha\beta$-Mila (Winands, 2006) or the default MC-Mila, 500 as the starting player and 500 as the second player. All games were played on a board of size 8×8. For each game, the number of won games as the starting player and as the second player were saved, as well as the number of moves that were played in every game. The players each received 5 seconds of thinking time per move. All experiments were run on a AMD64 Opteron 2.4GHz processor.

6.2 Playing Against an $\alpha\beta$ Player

This section presents the results of MC-Mila playing against $\alpha\beta$-Mila. $\alpha\beta$-Mila was written by Mark Winands. It participated in the Clobber Computer-Olympiad events of 2005 and 2006 where it came in first (Willemson and Winands, 2005) and second (Winands, 2006), respectively. $\alpha\beta$-Mila uses the following enhancements: multi-cut (Björnson and Marsland, 1999), transposition tables (Breuker, Uiterwijk, and Van den Herik, 1996), history heuristic (Schaeffer, 1989), late move reductions (Romstad, 2006), enhanced forward pruning (Winands *et al.*, 2005) and enhanced transposition cutoffs (Schaeffer

and Plaat, 1996). Additionally, it uses a limited amount of knowledge of combinatorial game theory; it can recognize several simple subgames with value 0 or $\star$. For reference, MC-Mila is first tested without any enhancements (except for the default enhancements it always contains). Then MC-Mila is tested with solving games and simplifying boards separately, as well as combined.

Table 6.1 shows the win percentages of the default MC-Mila playing against $\alpha\beta$-Mila. In total MC-Mila wins 16.3% of the games, with a 95% confidence interval of 2.3%. It seems to be an advantage to be the first player to move, since it performs slightly better as a starting player. Furthermore, it shows the average number of moves that were played during won and lost games. MC-Mila requires around 1.5 moves more than $\alpha\beta$-Mila, which is expected behavior, since $\alpha\beta$-Mila is generally the stronger player.

	win percentage	confidence interval	avg. moves/win	avg. moves/loss
as player 1	16.8%	±3.3%	45.6	44.0
as player 2	15.8%	±3.2%	46.2	44.4
total	16.3%	±2.3%	45.9	44.2

Table 6.1: The performance of the regular MCTS player.

Table 6.2 gives the results of MC-Mila enhanced with solved positions playing against $\alpha\beta$-Mila. The performance has increased considerably; in total the win percentage has increased with 72% from 16.3% to 28.1%. Remarkably, the advantage of starting the game is much larger in this experiment. Although the win percentage of MC-Mila as the second player still is considerably larger than that of the default MC-Mila, the win percentage as the starting player is almost 10% higher.

The average number of moves required to complete a game has increased. This is an indication that the games in general are more close. Moreover, the fact that MC-Mila wins more close games indicates better endgame play, which is probably a result of the MCTS-solver technique. It especially seems to be the case for won games as a first player, since the number of moves has increased the most for that case.

	win percentage	confidence interval	avg. moves/win	avg. moves/loss
as player 1	33.0%	±4.2%	46.7	44.5
as player 2	23.2%	±3.7%	46.8	45.0
total	28.1%	±2.8%	46.7	44.8

Table 6.2: The performance of a MCTS player enhanced with solved positions.

Figure 6.1 shows the number of games that were simulated per second on average by the MCTS algorithm with different combinations of the enhancements, for different stages of the game (i.e. after different numbers of moves have been played). These statistics were gathered over 50 games for each combination of enhancements. There is a significant drop in the number of performed simulations when any of the enhancements is used. This is due to the fact that it is not possible to retrieve the value of an 8×8 board position from the database at once. Instead, the individual subgames need to be identified, after which their values can be retrieved from the database and summed to obtain the value of the whole board. Obviously, this compromises the speed of finding a board value, however, it is necessary, since building a database with values of all 8×8 positions is infeasible. Remarkably, the performance has increased rather much, even though the number of simulations that were performed by MC-Mila with the solving boards enhancement is significantly lower. This can be credited to the use of MCTS-solver, which was proposed to reduce the number of simulations required to select the best move (Winands *et al.*, 2010).

The performance of MC-Mila enhanced with simplifying boards is shown in Table 6.3. The win percentage has dropped drastically, from 16.3% to 2.3%. This drastic drop can be explained by the large decrease of the number of simulations for each move, as show in Figure 6.1. Whereas the performance of MC-Mila with solved positions increased despite the decrease of simulations, due to MCTS-solver, the positive effect of the simplified boards is not large enough to outweight the decrease of simulations.

Finally, MC-Mila was tested with both solved positions and simplifying boards. Table 6.4 shows the results of this experiment. The performance has increased compared to the default MC-Mila, however, not as much as MC-Mila with just the solving boards enhancement. As already noted, the number of simulations per second significantly drop for MC-Mila with any enhancement. When using both enhancements the numbers of simulations that can be performed are slightly lower than when only using

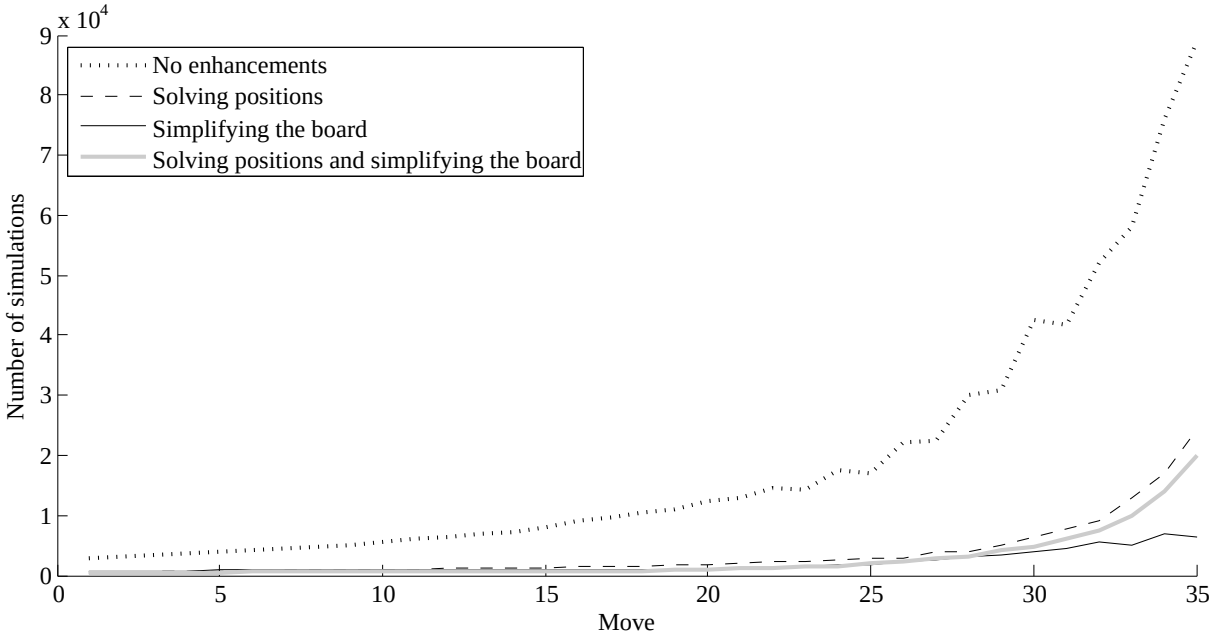


Figure 6.1: The average number of simulations second of MC-Mila with various enhancements.

	win percentage	confidence interval	avg. moves/win	avg. moves/loss
as player 1	2.8%	±1.5%	46.0	44.4
as player 2	1.8%	±1.2%	46.0	44.7
total	2.3%	±1.0%	46.0	44.6

Table 6.3: The performance of a MCTS player enhanced with simplifying boards.

the solving positions enhancement. The benefit of simplifying boards does not outweight the decreased number of simulations that could be performed. It can therefore be concluded that the performance increase compared to MC-Mila with no enhancements is only due to the solving boards enhancement, so combining the two enhancements is not advantageous.

	win percentage	confidence interval	avg. moves/win	avg. moves/loss
as player 1	25.6%	±3.9%	47.4	45.1
as player 2	16.6%	±3.3%	46.9	45.3
total	21.1%	±2.6%	47.2	45.2

Table 6.4: The performance of a MCTS player enhanced with both solved positions and simplifying boards.

6.3 Playing Against a Regular MCTS player

This section presents the results of MC-Mila with several combinations of enhancements playing against the default MC-Mila.

Firstly, the performance of MC-Mila with solved positions was tested. The results of this experiment can be found in Table 6.5. With a win percentage of 98.1% the performance has increased significantly. It is clear that the gameplay of MCTS can be improved much by adding knowledge of combinatorial game theory in this way. Furthermore, the average number of moves per game is lower than during the previous experiments against $\alpha\beta$-Mila. This indicates that the games during this experiment were not as close. The fact that lost games also were played in fewer moves is probably a result of bad gameplay during the early phases of the game.

	win percentage	confidence interval	avg. moves/win	avg. moves/loss
as player 1	99.0%	±0.9%	42.0	43.2
as player 2	97.2%	±1.5%	42.0	41.3
total	98.1%	±0.9%	42.0	41.8

Table 6.5: The performance of a MCTS player enhanced with solved positions.

Table 6.6 shows that the performance of MC-Mila with simplifying boards is slightly worse than that of the default MC-Mila; it wins 46.1% of the games. This is remarkable, given the bad performance of MC-Mila with this enhancement playing against $\alpha\beta$-Mila. One explanation for this observation may be that $\alpha\beta$-Mila has a better gameplay during early stages of the game, so that near the end of the game, once MC-Mila can approximate the value of moves reasonably despite the decreased number of simulations, it cannot catch up. However, no further tests were done to validate this reasoning.

	win percentage	confidence interval	avg. moves/win	avg. moves/loss
as player 1	45.4%	±4.4%	41.8	40.8
as player 2	46.8%	±4.4%	42.1	41.0
total	46.1%	±3.1%	42.0	40.9

Table 6.6: The performance of a MCTS player enhanced with simplifying boards.

The results of MC-Mila with both solved positions and simplifying boards playing against the default MC-Mila are given in Table 6.7. The performance increase from 16.3% to 92.7% is rather high, although not as high as during the experiment with MC-Mila enhanced with solved positions alone, where the win percentage was 98.1%. Therefore, it can be concluded that this good performance can be contributed to the solved positions enhancement.

	win percentage	confidence interval	avg. moves/win	avg. moves/loss
as player 1	93.4%	±2.2%	42.0	43.0
as player 2	92.0%	±2.4%	42.1	42.8
total	92.7%	±1.7%	42.1	42.9

Table 6.7: The performance of a MCTS player enhanced with both solved positions and simplifying boards.

Chapter 7

Conclusions and Future Research

This chapter presents the conclusions of this research. Section 7.1 answers the research questions that were stated in Section 1.5. In Section 7.2 the problem statement is discussed. In Section 7.3 some ideas are given for future research in this area.

Chapter contents: Conclusions and Future Research — Answering the Research Questions, Answering the Problem Statement, Future Research

7.1 Answering the Research Questions

In this section the research questions are answered, in order to answer the problem statement in Section 7.2.

Research question 1: How can we use endgame databases to store the combinatorial game-theoretic value of a subgame in an efficient and effective way?

This research question was answered in Chapter 4. The most important aspect of the endgame database is that the contents have to be quickly accessible, since they need to be accessed several times during each MCTS simulation. To that end, each value was stored using the unique index of its corresponding board position, so that it is not required to search the database when retrieving a value; it can be retrieved immediately.

In Section 4.2 the contents of the database were discussed. It holds values for board positions of various sizes, up to a size of 4×4. All combinatorial game-theoretic values could be calculated for the smallest board sizes. For subgames of size 2×3 62.3% was solved. This percentage has already dropped to 19.6% for subgames of size 3×3. As the board size gets bigger, the subgames become increasingly harder to solve. Subgames of sizes 3×4 and 4×4 can only be solved in 4.6% and 0.8% of the cases, respectively.

Research question 2: How can combinatorial game-theoretic values be used to improve the selection phase of the MCTS program MC-Mila?

The values in the database were used to try to solve the board position after every move during the MCTS simulations. The use of MCTS-solver allowed for the values of subgames to be used in the selection phase. Solved wins and losses were stored in the tree, so that they could be used to gain a more precise approximation of the value of a move. Since MCTS-solver applies to all phases of MCTS, the performance boost due to applying combinatorial knowledge to the selection phase was not tested separately. However, there was a considerable performance increase by using the combinatorial game-theoretic values. Playing against $\alpha\beta$-Mila the win percentage increased from 16.3% to 28.1%, and playing against the default MC-Mila the performance increased from 50% to 98.1%.

Research question 3: How can combinatorial game-theoretic values be used to improve the playout phase of the MCTS program MC-Mila?

It was tried to improve the playout phase of MCTS with combinatorial game theory in two ways. Firstly, after each move during the playout phase the board position was attempted to be solved. Whenever a position could be solved the game could be terminated, since the winner of a game can be predicted by its value. The winner of the game was then stored in the tree so that it could later be used by MCTS-solver. This resulted in a considerable performance boost, as already noted in the discussion of the previous research question.

Secondly, after each move if the game could not be solved entirely, some subgames that could be solved may be removed from the board. This applies to subgames with value 0 and subgames that have a combined value of 0. During the experiment this strategy was not successful. In fact, it caused a drop in performance. In games against $\alpha\beta$-Mila the performance dropped from 16.3% to 2.3%, and playing against the default MC-Mila it dropped from 50% to 46.1%. This performance drop was caused by the vastly decreased number of simulations that could be performed when using this technique.

The combination of solved positions and removing subgames from the board was tested as well. Although the performance increased compared to the default MC-Mila, both when playing against MC-Mila and $\alpha\beta$-Mila, the performance boost was not as big as when only using solved positions.

Research question 4: How can combinatorial game-theoretic values be used to improve the endgame play of the MCTS program MC-Mila?

As soon as the combinatorial game-theoretic value of a complete board can be calculated, the game can be played out optimally. Whenever the root node is a proven win in the MCTS tree, MCTS-solver chooses the winning move at each stage during the remainder of the game. In the case that the root node is a proven loss, it does not matter which move is chosen, since each move leads to a proven loss. Since MCTS-solver applies to all steps of MCTS, the specific performance increase by the improved endgame play was not measured. However, as already discussed in previous research question, the performance increase by using MCTS-solver is considerable, and it can be assumed that this is partly due to better endgame play.

7.2 Answering the Problem Statement

Having answered all research questions, the problem statement can be answered. The problem statement is the following:

Problem statement: How can we apply combinatorial game theory in MCTS?

Knowledge of combinatorial game theory was injected in MCTS in two different ways. Firstly, at each step during the selection and playout phase, it was attempted to find the combinatorial game-theoretic value of the board position. If the value could be found it was stored in the MCTS tree, so that it could be used by MCTS-solver. MCTS-solver improved the selection and playout phases, as well as the endgame play.

Secondly, whenever the value of the whole board could not be found, but some subgames could be solved, some of these solvable subgames need not be considered, so they can be removed from the board. However, the experiments indicated that the performance of MCTS decreased by applying this enhancement, due to the vast drop of the number of simulations that could be performed for each move. Combining the simplifying boards enhancement with MCTS-solver resulted in a better performance than the default MC-Mila, since MCTS-solver is able to handle a decrease of simulations. However, the performance was not better than when using only MCTS-solver.

7.3 Future Research

Although combinatorial game theory is a field that has received little attention over the years, its theory is quite extensive. However, only little research has been done on its applications in the field of Artificial Intelligence. Hence, there are still many areas that can be explored.

Firstly, this thesis investigates the application of combinatorial game theory to a MCTS program with only few enhancements. However, during the past few years much research has been done on MCTS, resulting in many variations and enhancements. Combining some of these enhancements with the techniques proposed in this paper might further improve the performance. Moreover, combinatorial game theory can be applied to MCTS in different ways than the ways proposed in this thesis. The simplifying boards enhancement might be improved by only using it at the root and at the start of the playout phase, instead of after every move. This would reduce the decrease of the number of performed simulations, which was the cause of its bad performance. Additionally, since hardly any solvable subgames form during the first 20–25 moves of a game, neither the solved subgames nor the simplifying boards enhancement need to be used during the first part of the game.

Secondly, the techniques that were investigated in this thesis also have applications in other search methods, such as $\alpha\beta$-search. It would be interesting to see if combinatorial game theory can be used as effectively to improve the performance of these other methods.

Next, it was already mentioned that the database we used could be expanded by implementing Algorithm 3. Siegel's Combinatorial Game Suite (Siegel, 2007) is able to solve considerably more game positions as a result of implementing Algorithm 3, especially for larger (sub)games. For a board size 3×3 it can solve 70.1% of the positions, in contrast with the 50.6% of the positions we solved. Obviously, being able to solve more board positions results in a better performance of the MCTS player.

Furthermore, for all-small games (i.e. games for which the combinatorial game-theoretic values only consist of infinitesimals) it is possible to calculate the *atomic weight* of a position, also called its *uppitiness*, since it is the number of ups the value of the position is most likely equal to. It turns out that the outcome of a position can be predicted with its atomic weight (Berlekamp *et al.*, 1982). Since the atomic weight can often even be calculated for positions with a complicated canonical form, incorporating them in MCTS has much potential.

References

Abramson, B. (1990). Expected-Outcome: A General Model of Static Evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 2, pp. 182–193. [25]

Albert, M. H., Grossman, J. P., Nowakowski, R. J., and Wolfe, D. (2005). An Introduction to Clobber. *Integers, The Electronic Journal of Combinatorial Number Theory*, Vol. 5, No. 2, p. 12. [5, 7, 13]

Allis, L. V. (1994). *Searching for Solutions in Games and Artificial Intelligence*. Ph.D. thesis, University of Limburg, Maastricht, The Netherlands. [3, 6]

Arneson, B., Hayward, R. B., and Henderson, P. (2010). Monte Carlo Tree Search in Hex. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 251–257. [27]

Berlekamp, E. R., Conway, J. H., and Guy, R. K. (1982). *Winning Ways for your Mathematical Plays*, Vol. 1. Academic Press, Waltham, MA, USA. [2, 3, 6, 9, 10, 12, 13, 14, 37]

Billings, D., Davidson, A., Schaeffer, J., and Szafron, D. (2002). The Challenge of Poker. *Artificial Intelligence*, Vol. 143, No. 1, pp. 210–240. [25]

Björnson, Y. and Marsland, T. A. (1999). Multi-cut alpha-beta pruning. Vol. 1558 of *Lecture Notes in Computing Science*, pp. 15–24, Springer-Verlag, Berlin, Germany. [31]

Bouton, C. L. (1902). NIM, A Game with a Complete Mathematical Theory. *Annals of Mathematics, Princeton*, Vol. 3, No. 2, pp. 35–39. [2]

Bouzy, B. (2005). Associating Domain-Dependent Knowledge and Monte Carlo Approaches within a Go Program. *Information Sciences, Heuristic Search and Computer Game Playing IV*, Vol. 175, No. 4, pp. 247–257. [26]

Bouzy, B. (2006). Associating Shallow and Selective Global Tree Search with Monte Carlo for 9×9 Go. *Computers and Games* (eds. Y. Björnsson H.J. van den Herik and N. Netanyahu), Vol. 3846 of *LNCS*, pp. 67–80. [25]

Bouzy, B. and Helmstetter, B. (2003). Monte-Carlo Go Developments. *Advances in Computer Games 10* (eds. H.J. van den Herik, H. Iida, and E.A. Heinz), Vol. 263, pp. 159–174, Kluwer Academic, Boston, MA, USA. [25]

Breuker, D. M., Uiterwijk, J. W. H. M., and Herik, H. J. van den (1996). Replacements schemes and two-level tables. *ICCA Journal*, Vol. 19, No. 3, pp. 175–180. [31]

Brügmann, B. (1993). Monte Carlo Go. Technical report, Physics Department, Syracuse University, Syracuse, NY, USA. [25]

Campbell, M., Hoane Jr., A. J., and Hsu, F-H. (2002). Deep Blue. *Artificial Intelligence*, Vol. 134, Nos. 1–2, pp. 57–83. [1]

Cazenave, T. (2011). *Monte-Carlo Approximation of Temperature*. To appear in Games of No Chance 4. [2]

Chaslot, G., Saito, J-T., Bouzy, B., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2006). Monte-Carlo Strategies for Computer Go. *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium* (eds. P-Y. Schobbens, W. Vanhoof, and G. Schwanen), pp. 83–91. [26]

Chaslot, G. M. J-B., Winands, M. H. M., Uiterwijk, J. W. H. M., Herik, H. J. van den, and Bouzy, B. (2008a). Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, Vol. 4, No. 3, pp. 343–357. [26]

Chaslot, G. M. J-B., Winands, M. H. M., Uiterwijk, J. W. H. M., Herik, H. J. van den, and Bouzy, B. (2008b). Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, Vol. 4, No. 3, pp. 343–357. [28]

Conway, J. H. (1976). *On Numbers and Games*. Academic Press, Waltham, MA, USA. [2]

Coulom, R. (2007). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *Proceedings of the 5th International Conference on Computer and Games* (eds. P. Ciancarini and H.J. van den Herik), Vol. 4630 of *LNCS*, pp. 72–83, Springer-Verlag, Heidelberg, Germany. [2, 26, 27]

Demaine, E. D., Demaine, M. L., and Fleischer, R. (2002). Solitaire Clobber. *Theoretical Computer Science*, Vol. 313, No. 3, pp. 325–338. [7]

Gasser, R. (1991). Applying Retrograde Analysis to Nine Men's Morris. *Heuristic Programming in Artificial Intelligence II*, pp. 161–173. Ellis Horwood Ltd., Chichester, UK. [3]

Gelly, S. (2007). *Une Contribution à l'Apprentissage par Renforcement; Application au Computer-Go*. Ph.D. thesis, Université Paris-Sud, Paris, France. In French. [2, 27]

Gelly, S. and Silver, D. (2008). Achieving Master Level Play in 9 × 9 Computer Go. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pp. 1537–1540. [2, 25, 26, 27]

Gelly, S., Y.Wang, Munos, R., and Teytaud, O. (2006). Modications of UCT with Patterns in Monte-Carlo Go. Technical report, INRIA, France. [26]

Hayes, B. (2001). Third Base. *American Scientist*, Vol. 89, No. 6, pp. 490–494. [21]

Kloetzer, J. (2010). *Monte-Carlo Techniques: Applications to the Game of the Amazons*. Ph.D. thesis, Japan Advanced Institute of Science and Technology, Kanazawa, Japan. [27]

Kloetzer, J., Iida, H., and Bouzy, B. (2009). Playing Amazons Endgames. *ICGA Journal*, Vol. 32, No. 3, pp. 140–148. [27]

Knuth, D. E. and Moore, R. W. (1975). An Analysis of Alpha-Beta Pruning. *Articial Intelligence*, Vol. 6, No. 4, pp. 293–326. [2]

Kocsis, L. and Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. *Machine Learning: ECML 2006* (eds. J. Fürnkranz, T. Scheffer, and M. Spiliopoulou), Vol. 4212 of *Lecture Notes in Articial Intelligence*, pp. 282–293. [26]

Lorentz, R. J. (2008). Amazons Discover Monte-Carlo. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *LNCS*, pp. 13–24, Springer-Verlag, Berlin Heidelberg, Germany. [27]

Marsland, T. A. (1983). Relative Efciency of Alpha-Beta Implementations. *Proceedings of the 8th International Joint Conference on Articial Intelligence (IJCAI-83)*, pp. 763–766, Karlsruhe, Germany. [2]

Müller, M. (2002). Computer Go. *Artificial Intelligence*, Vol. 134, Nos. 1–2, pp. 145–179. [1]

Müller, M., Enzenberger, M., and Schaeffer, J. (2004). Temperature Discovery Search. *AAAI 2004*, pp. 658–663. [2]

Nash, J. (1952). Some Games and Machines for Playing Them. Technical Report D-1164, Rand Corp, USA. [3]

Neumann, J. von (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, Vol. 100, No. 1, pp. 295–320. In German. [2]

Nilsson, N. (1971). *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, NY, USA. [1]

Reinefeld, A. (1983). An Improvement to the Scout Search Tree Algorithm. *ICCA Journal*, Vol. 6, No. 4, pp. 4–14. [2]

Romein, J. W. and Bal, H. E. (2003). Solving Awari with Parallel Retrograde Analysis. *IEEE Computer*, Vol. 36, No. 10, pp. 26–33. [3]

Romstad, T. (2006). An introduction to late move reductions. http://www.glaurungchess.com/lmr.html. [31]

Schaeffer, J. (1989). The History Heuristic and the Performance of Alpha-Beta Enhancements. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 11, pp. 1203–1212. [31]

Schaeffer, J. and Plaat, A. (1996). New advances in alpha-beta searching. pp. 124–130, ACM Press, New York, NY, USA. [31]

Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S. (2007). Checkers Is Solved. *Science*, Vol. 317, No. 5844, pp. 1518–1522. [1, 3]

Sheppard, B. (2002). *Towards Perfect Play of Scrabble*. Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands. [25]

Siegel, A. N. (2005). *Loopy Games and Computation*. Ph.D. thesis, University of California, Berkeley, CA, USA. [2, 17, 18]

Siegel, A. N. (2007). Combinatorial Game Suite. http://cgsuite.sourceforge.net/. [7, 18, 37]

Ströhlein, T. (1970). *Untersuchungen über Kombinatorische Spiele*. Ph.D. thesis, Fakultät für Allgemeine Wissenschaften der Technischen Hochschule München, Germany. In German. [21]

Tromp, J. (2008). Solving Connect-4 on Medium Board Sizes. *ICGA Journal*, Vol. 31, No. 2, pp. 110–112. [3]

Turing, A. M. (1953). Digital Computers Applied to Games. *Faster Than Thought*, pp. 286–297. [1]

Werf, E. C. D. van der and Winands, M. H. M. (2009). Solving Go for Rectangular Boards. *ICGA Journal*, Vol. 32, No. 2, pp. 77–88. [3]

Willemson, J. (2005). Computer Clobber Tournament at Tartu University. *ICGA Journal*, Vol. 28, No. 1, pp. 51–54. [7]

Willemson, J. and Winands, M. H. M. (2005). MILA Wins Clobber Tournament. *ICGA Journal*, Vol. 28, No. 3, pp. 188–190. [7, 31]

Winands, M. H. M. (2006). PAN Wins Clobber Tournament. *ICGA Journal*, Vol. 29, No. 2, pp. 106–107. [3, 31]

Winands, M. H. M. and Björnsson, Y. (2010). Evaluation Function Based Monte-Carlo LOA. *Advances in Computer Games (ACG 2009)* (eds. H. J. van den Herik and P. H. M. Spronck), Vol. 6048 of *LNCS*, pp. 33–44, Springer-Verlag, Berlin Heidelberg, Germany. [27]

Winands, M. H. M., Herik, H. J. van den, Uiterwijk, J. W. H. M., and Werf, E. C. D. van der (2005). Enhanced forward pruning. *Information Sciences*, Vol. 175, No. 4, pp. 315–329. [31]

Winands, M. H. M., Björnsson, Y., and Saito, J-T. (2008). Monte-Carlo Tree Search Solver. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *LNCS*, pp. 25–36. [27]

Winands, M. H. M., Björnsson, Y., and Saito, J-T. (2010). Monte Carlo Tree Search in Lines of Action. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 239–250. [27, 28, 32]

Wolfe, D. (1991). *Mathematics of Go: Chilling Corridors*. Ph.D. thesis, Division of Computer Science, University of California, Berkeley, CA, USA. [2]