Master Thesis

Implementing a Computer Player for Abalone using Alpha-Beta and Monte-Carlo Search

Pascal Chorus

Master Thesis DKE 09-13

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science of Artificial Intelligence at the Department of Knowledge Engineering of the Maastricht University

Thesis committee:
Dr. ir. J.W.H.M. Uiterwijk
Dr. M.H.M. Winands
M.P.D. Schadd, M.Sc.
J.A.M. Nijssen, M.Sc.

Maastricht University
Faculty of Humanities and Sciences
Department of Knowledge Engineering
Master Artificial Intelligence

June 29, 2009

Preface

This master thesis was written at the Department of Knowledge Engineering of Maastricht University. It is the result of a research project done in the area of Artificial Intelligence. The goal was to implement an effective and efficient computer player for the board game Abalone. During the work alpha-beta search and Monte-Carlo search were investigated as possible approaches for implementation.

I would like to thank some people that helped me during my work on this thesis. First of all I thank my supervisor, Dr. ir. Jos Uiterwijk, for his guidance during the whole project. He always gave me new ideas and important advices. Thank you for reading and commenting the thesis and for the discussions during our meetings and your lectures *Games and AI* and *Intelligent Search Techniques*.

Furthermore, I would like to thank Dr. Mark Winands who was involved in both courses mentioned above. Thanks for the useful information and impulses you gave in the lectures and talks.

Pascal Chorus
Aachen, June 29, 2009

Abstract

For over 3000 years board games are played by humans. They play against each other to have fun and to train their strategical thinking.

Since the computer was invented people do not only play board games against each other, but also against computer players. Many games that we know today are available as a computer version. On the other hand, there still exist recent games that are not analyzed very detailed.

One of these games is Abalone. It was invented in 1987. Nowadays it is very popular. Tournaments are played regularly, but until now there is not much research done in this game.

This thesis focuses on the implementation of an effective and efficient computer player for Abalone. At the beginning of the thesis the game Abalone is introduced. Also, its state-space complexity and game-tree complexity are given and compared to other games.

After that the two algorithms that were implemented and tested for playing Abalone are described in detail. The first one is alpha-beta search. To strengthen its performance some improvements were implemented for it, namely several move-ordering techniques, a transposition table and quiescence search. The second algorithm that was investigated is Monte-Carlo search. Two improvements were implemented for it: depth-limited simulated games and a sophisticated evaluation of the outcomes of the simulated games.

It turned out that the alpha-beta search performs much better. The implemented extensions all advanced the player to varying degree.

The alpha-beta player is able to challenge advanced human players.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter an introduction to games is given. First of all, board games are introduced together with some well-known examples. Afterwards, the use of artificial intelligence in board games and especially in Abalone is discussed. Finally, the problem statement and the research questions are formulated and an outline of the remaining thesis is given.

1.1 Board Games

Since humans live in civilization, they play games. Games are amusement and they bring people together, make them forget about their real-life problems for hours and entertain them. The first mentionings of board games go back to 3500-3000 BC. The Egypts invented some games then. Investigators figured out that the famous game backgammon was mentioned for the first time already 3000 BC in the Burnt City of Iran [12]. Over the years more and more games were invented [16]. To mention only some well-known of them [28]:

Backgammon A similar ancestor of backgammon was invented between 476 and 481.

Chess Predecessors were invented in the early 6th century, modern chess as it is known today was invented in 1475.

Reversi Invented in 1883-1884.

Halma The origin of Chinese Checkers, invented in 1883.

Checkers Ancestor called Alquerque was invented in 600 BC. Modern checkers invented in the 12th century when the rules of Alquerque were combined with an 8×8 game board [26]

Go Invented about 500 BC.

Monopoly Invented in the early 1930s.

Nowadays board games are still famous. The variety of different games increases more and more and goes from card games and games with dice to more extensive board games that can contain pawns, tiles, dice and cards. To play these games is of course great fun, but it is also a competition between the players. One's goal is to challenge the opponents and to beat them by working out sophisticated and clever strategies. Often the game contains aspects of chance, e.g. rolling dice or taking a card from a deck. One cannot foresee exactly what will happen. Thus, the game can even get more exciting.

A particular category of games are the so-called classic board games. These are two-player games in which both players have perfect information, i.e. there are no hidden cards or some other hidden information. Another important aspect is that they do not contain any actions based on luck. Thus, the players have absolute control over their actions. Mostly, the whole game can be described in some sentences. The rules are simple. Nevertheless, the number of different possible moves is high which makes the game complex and challenging. Famous examples of such games are chess and Go.

1.2 AI used in Board Games

Since the computer was invented (between 1940 and 1950) people try to make computers play games. In 1950 Claude Shannon [21] and Alan Turing [22] came up with first ideas to build a chess computer. From then on research in the area of computers playing games has continued to date. People tried to implement computers that can act in a humanlike fashion, i.e. they can reason about their possibilites and make decisions that let them appear as intelligent players. Since computers get faster and cheaper the preconditions for research projects are easier to fulfil. Today one can find several approaches for playing agents for a variety of games. Still chess is one of the most famous games for research.

In the recent years of computer history new technologies allowed programmers to design huge digital worlds. The games are interactive and no longer only board games are played, but also strategy games and first-person shooters in which the player is part of the world. He interacts with other characters and tries to reach his goals to finish quests. The most recent category of computer games are the so-called massively multiplayer online roleplaying games where humans all around the world meet in a digital kingdom to talk, argue, fight, trade, etc.

However, a huge part of research in computer games is still concerned with classic board games. They have simple rules and perfect information. Thus, they can be represented in a computer game easily since they build a perfect description of a world without exceptions.

Although the world is perfect the number of possible actions players can take is enormous. Due to the fact that it is computationally intractable to consider all possible moves and their consequences to the end of the game, programmers invented search techniques to find well-performing moves the computer player can take. While computers become faster the implemented techniques can be

further enhanced and programmers have new possibilities.

But why are computer games that interesting for research? Since the complexity of games is high one cannot observe all possible future game positions. Therefore, it is not possible to find an optimal move to take in a certain time range. Hence the computer cannot act optimally, but it has to find a way to make a good guess about the accuracy of the possible moves. "In this respect, games are much more like the real world than the standard search problems..." [19]. When research in games leads to some algorithm to solve the problem or to find a non-optimal, but accurate solution, these results can be applied to other sciences and research areas [17]. So the algorithms are not limited to the game-solving problem, but they can be adapted to sciences like economics.

1.3 AI and Abalone

One of the recent games is Abalone. It was invented in 1987 by Laurent Levi and Michel Lalet [27]. Abalone belongs to the group of classic board games. There is already some research done in this game. Some researchers came up with implementations of game playing agents. But since the game is relatively new the research done so far is not that extensive as for other games.

In the following three implementations of agents playing Abalone are described. The research papers are available on the internet. All of them use an approach based on alpha-beta search.

1. Nyree Lemmens wrote a bachelor thesis at Maastricht university titled "Constructing an Abalone Game-Playing Agent" in June 2005 [15]. The agent's alpha-beta search uses move ordering and transposition tables to fasten the search. The game tree is searched with a depth of 2. However, the evaluation function seems to be sophisticated since it uses several features weighted according to the current progress of the game.

2. The report "A Simple Intelligent Agent for Playing Abalone Game: ABLA" written by Ender Ozcan and Berk Hulagu [18] is about an alpha-beta searching agent for Abalone as well. They implemented an evaluation function using two features: the distance to the game board's center and a measure of the marbles' grouping. They made experiments with using only one or both of the features at a tree depth of either 3 or 4. Thus, they tried to find a well performing trade-off between complexity of the evaluation function and the depth of search in the game tree.

3. "Algorithmic Fun - Abalone" written by Oswin Aichholzer, Franz Aurenhammer and Tino Werner [3] programmed an agent for Abalone using alpha-beta search with an evaluation function only consisting of calculating the center of mass of each player's marbles with respect to the board game's center. They implemented strong heuristics to early prune large parts of the game tree. That leads to the ability to search the game tree to a depth of 9-10.

1.4 Problem Statement and Research Questions

This thesis focuses on the implementation of an effective and efficient playing agent for Abalone called AIBA.

Accordingly the problem statement is:

Is it possible to implement a strong computer player for Abalone?

To implement a player agent, we have to solve problems like "What is the best move in a certain situation" or "Is my current position a strong or a weak one?".

Two of the most well-known algorithms that are promising to answer such questions are Monte-Carlo search and alpha-beta search. This leads to the research questions:

RQ1: Can an Abalone player be implemented using the alpha-beta search?

RQ2: Can an Abalone player be implemented using the Monte-Carlo search?

Furthermore, it seems worthwhile to improve the algorithms to achieve a higher quality of decision making:

RQ3: Is it possible to implement some improvements to enhance the decision-making process of both approaches?

Finally, it is interesting to compare the best implementation of both approaches in order to find out which is the more promising approach.

RQ4: Which one of the two approaches performs better?

1.5 Outline of the Thesis

The remainder of this thesis is structured in the following way:

- Chapter 2 introduces the game Abalone. The game rules are explained in detail and some examples are given.
- Chapter 3 discusses some basic facts of Abalone from a programmer's point of view. Additionally, the calculation of the game's complexity is described.
- In chapter 4 the two basic algorithms, alpha-beta search and Monte-Carlo search, are explained together with some approaches to improve their performance.

- Chapter 5 describes the experiments that were performed and discusses the results.
- Chapter 6 concludes the research done in this master thesis and gives some suggestions for further research.

Chapter 2

Abalone

In this chapter the game Abalone will be explained in detail. First of all, some basic information is given. After that the possible moves and special game situations are discussed.

2.1 Basics

Abalone is a two-player strategic board game. The game board has a hexagonal shape with 61 fields. Each player has 14 marbles. One player gets the black ones, the other one the whites. The two players have to move in turns. The black player begins the game.

The objective of the game is to eject the opponent's marbles from the game board. The player who first ejects 6 marbles of the opponent wins the game [1].

There are several start formations for the marbles. The two most popular ones are the Standard formation which is proposed by the inventors of the game. The second one is called Belgian Daisy. This is a formation often used in tournaments, because the positions are more offensive right at the beginning of the game [23]. These two formations are illustrated in the figures 2.1 and 2.2.

How marbles can be ejected from the game board is described in the next sections where the possible moves are explained.

2.2 Possible Moves

In every turn the player is allowed to move one, two or three marbles of his own colour. Multiple marbles may only be moved if they are connected to each other and they are in a row. Furthermore, all marbles must be moved in the same direction. It is possible to move the marbles inline, which means along the axis of their current position (see figure 2.3). Another possibility is to make a broadside move, i.e. a move not along the axis of the marbles, but sideways (see figure 2.4).

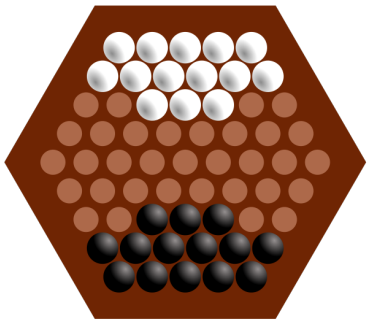

Figure 2.1: Standard position

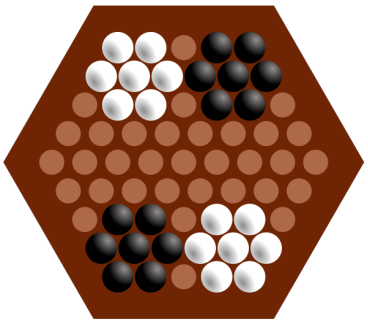

Figure 2.2: Belgian Daisy position

Independent from whether an inline move or a broadside move is performed the player may only move the marbles one field further. A move is still allowed even if a row of marbles of the same colour is split by the move.

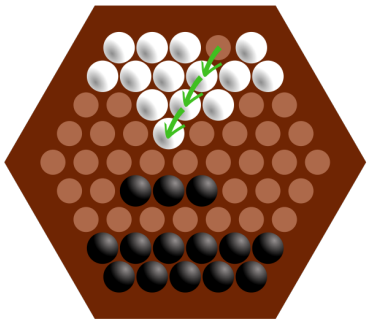

Figure 2.3: Sample inline move

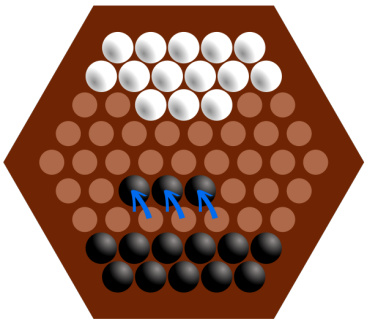

Figure 2.4: Sample broadside move

Broadside moves are only allowed if the fields the marbles are moved to are empty. Inline moves are possible if one of the following two conditions is fulfilled: Either the field behind the last marble is empty or the player is able to perform a *Sumito* move. Sumito moves are those where some of the opponent's marbles are pushed. In the next section the preconditions and the consequences of Sumito moves are described.

2.3 Sumito Moves and Special Situations

A Sumito move is a move where some opponent's marbles are pushed by moving some of the player's own marbles. A Sumito situation is reached in case that there is a row of marbles where on one side there are only black marbles and on

the other side there are white marbles. Between the marbles there must not be a gap. If the number of marbles of one player exceeds the number of the other player's marbles he is in a Sumito situation. To determine such a situation for each player at most three marbles of each player are taken into account since a player is not allowed to move more than three marbles per turn. Actually, there exist only three different Sumito situations:

- two marbles against one
- three marbles against one
- three marbles against two

In figure 2.5 all possible Sumito situations are illustrated. The black player can choose one of them by moving a row of his marbles to the right.

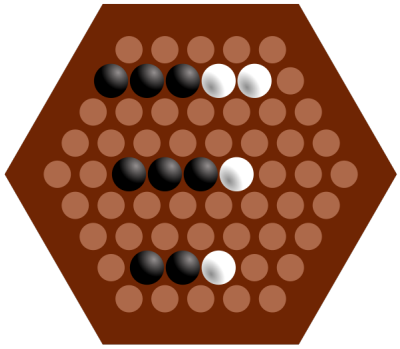

Figure 2.5: Sumito situations

Another important term in Abalone is the so-called pac situation. In that situation neither player is able to perform a Sumito move, because the number of marbles of the same colour are equal. Thus, there exist three different pac situations:

- one marble against one
- two marbles against two
- three marbles against three

In figure 2.6 some pac and pac-like situations are illustrated:

The marbles on top of the board are arranged in a simple 2-on-2 pac situation.

The marbles in the middle are in a pac situation as well. Although there are 4 black marbles against 3 white marbles it is a pac situation. This is because a player is allowed to move 3 marbles at most. Thus, it is a 3-on-3 pac situation.

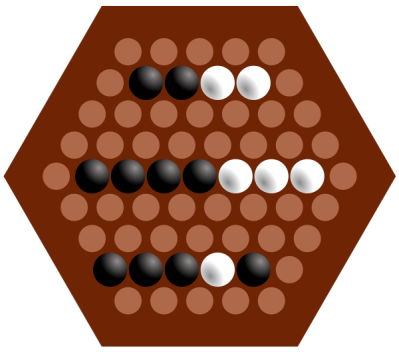

Figure 2.6: Pac(-like) situations

The setup of marbles at the bottom is no Sumito situation, because there is no free space behind the white marble.

To achieve the objective of the game, which is ejecting six opponent's marbles from the board, the player has to perform a Sumito move in a position where opponent's marbes are at the border of the game board. After this move one of the opponent's marbles is lost. Figure 2.7 shows such an ejecting move performed by the white marbles. Note that after the move the black player is able to eject a white marble by performing the move D2C2.

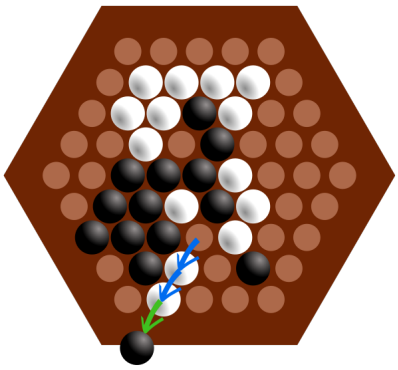

Figure 2.7: Ejecting move

It can happen that both players' goal is to defend and prevent the marbles from being ejected. Such a game can last forever. To limit the game an additional rule can be introduced, e.g. each player has a certain amount of time,

maybe 10 or 15 minutes. After the time of one player is over, this player loses the game. Another approach is to limit the total number of plies, i.e. the number of turns. If the limit is reached the game ends in a draw.

To describe the single board fields a notation is introduced. It is based on the cartesian coordinate system. Since the game board has a hexagonal shape the y-axis is the one from downright to the upper left. The x-axis is simply the horizontal line at the bottom (see figure 2.8).

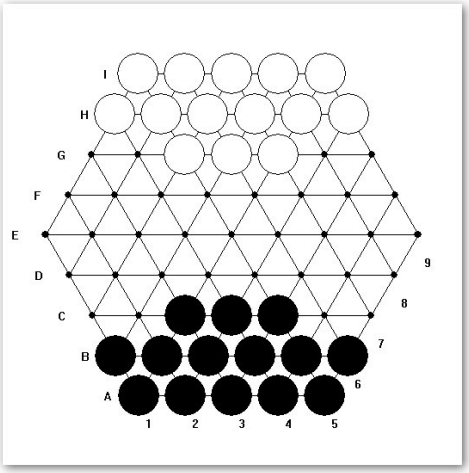

Figure 2.8: Coordinate system for notation

For clearness, the columns are numbered from 1 to 9 (x coordinates) and the rows are named from A to I (y coordinates). A field is then indicated by its y coordinate followed by its x coordinate. For example, the field occupied by the black marble in the center of the upper row is the field C4.

A move can also be represented using this notation. A simple notation can be used for inline moves. Only the field occupied by the last marble to move is noted followed by the field the marble is moved to. For example, the move showed in figure 2.3 is described by I8H7.

To notate a broadside move one has to refer to three fields. First of all one mentions the first and the last field of marbles in a row that are moved. The third field that is noted indicates the new field for the marble mentioned first. For example, the move showed in figure 2.4 is noted as C3C5D3.

Chapter 3

Analysis of Abalone

This chapter gives some information about the properties of Abalone. First of all some basic information is given. In section 3.2 Abalone's complexity is explained.

3.1 Basics

Abalone is a classic board game. It is a recent game, invented in 1987. The standard version is a two-player game. Nowadays there exist extensions to play it with up to six players, but this thesis focusses on the original version having two players. Abalone belongs to the class of abstract strategy games. It is completely based on strategy, involving no luck at all. Each player has control over his actions. Additionally, both players have perfect information [5] about the current game position. There is no hidden information. Furthermore, Abalone belongs to the group of zero-sum games [5]. Thus, there are no aspects of cooperation or trading. Each player is only interested in achieving the best position for himself and at the same time forcing the opponent into a bad position.

Due to the characteristics of Abalone alpha-beta search appears to be a good approach to implement a strong computer player. Since the game has perfect information it represents a perfect world that can easily be implemented as a computer game. It is possible to represent the game progress by two players, max and min, where max tries to maximize his position while min tries to minimize it. The values are calculated by an evaluation function.

Monte-Carlo search serves as a possible approach as well. There are no hard restrictions that a game has to fulfil in order to use Monte-Carlo search.

3.2 Complexity

In this section the complexity of Abalone is discussed. The complexity is already investigated by Lemmens [15] and Aichholzer [2]. So we refer to their research

at this point. The ideas about how to calculate a good approximation to the complexity are described.

3.2.1 State-Space Complexity

The state-space complexity of a game is the number of possible board positions [5]. First of all the game board contains 61 fields. Furthermore, each player starts with 14 marbles leading to 28 marbles on the board at the beginning of the game. The game ends when one of both players has lost 6 of his marbles. Therefore, the least number of marbles that can be on the game board is 17, where one player has lost 5 marbles and the other one 6. This is a terminal position.

The following formula is a good approximation of an upper bound for the number of possible game positions:

$$\sum_{k=8}^{14} \sum_{m=9}^{14} \frac{61!}{k!(61-k)!} \times \frac{(61-k)!}{m!((61-k)-m)!}$$

Since Abalone has a hexagonal shaped game board there exist 6 rotations and 6 mirrors. These positions represent identical board positions. Thus, the calculated number has to be divided by 12.

This calculation results in an approximation of 6.5×10^{23} [15]. Aichholzer [2] calculated an approximation for Abalone's complexity as well. He found an approximation of 10^{25}. Thus, the complexity seems to be somewhere between 10^{24} and 10^{25}.

3.2.2 Game-Tree Complexity

The game-tree complexity is a measure to represent the complexity of the game tree, i.e. how many possibilities exist to play the game [5].

The game tree visualizes all possible moves a player is allowed to choose. The root of the tree is the initial position where the first player is to move. Every edge leading to a node one level below represents a possible move for the first player. In each of the underlying nodes the second player is to move having a number of possibilities for his first move. A whole move in a game consists of two half moves, i.e. the first player moves and the second player moves. A half move is also called a ply.

Lemmens [15] found that an average Abalone game consists of 87 plies. This number will be of interest for the later calculation of the game-tree complexity. The second number that has to be known is the average branching factor of the game-tree. In other words the branching factor is the number of possible moves a player can perform in a given position. Since in Abalone the number of possible moves can vary for several game positions an average number is used. Lemmens [15] found an average branching factor of 60. Experiments done during this master thesis resulted in a similar value. Aichholzer [2] mentions a branching factor of 65-70 in his research work.

With the help of both the average branching factor and the average number of plies per game an approximation of the game-tree complexity can be calculated. The game-tree complexity is the number of leaf nodes in the smallest full-width decision tree. Using an average branching factor of 60 and an average number of plies per game of 87 the approximation of the game-tree complexity is 5.0×10^{154}

3.2.3 Comparison with other Games

The former section described the complexities of Abalone. This section compares these values with other board games. Table 3.1 lists some well-known board games together with their complexities.

Game	board size	state-space complexity (log)	game-tree complexity (log)	average game length (plies)
Tic-tac-toe [29]	9	3	5	9
Pentominoes [11]	64	12	18	10
Connect Four [11]	42	14	21	36
Checkers [11]	32	21	31	70
Nine Men's Morris [11]	24	10	50	50
Draughts [11]	50	30	54	90
Chinese checkers (2 sets) [29]	121	23		
Chinese checkers (6 sets) [29]	121	35		
Lines of Action [29]	64	24	56	63
Reversi / Othello [11]	64	28	58	58
Hex (11x11) [11]	121	57	98	40
Chess [11]	64	46	123	80
Backgammon [29]	28	20	144	50-60
Xiangqi [29]	90	48	150	95
Abalone	**61**	**24**	**154**	**87**
Shogi [11]	81	71	226	110
Go (19x19) [11]	361	172	360	150

Table 3.1: Board games and their complexities

This table shows that Abalone has a state-space complexity similar to Checkers. Since Checkers has been solved it may be possible to solve Abalone with a high-performance machine as well. On the other side the game-tree complexity is similar to Xiangqi which is quite complex. Therefore, it may be not that easy to solve the game.

Chapter 4

Algorithms

This chapter introduces the two basic approaches that are followed to implement a strong Abalone playing agent. First of all there is a detailed description of the alpha-beta search which is based on the simpler minimax algorithm. Afterwards, the Monte-Carlo search is explained. At the end of each algorithm some improvements are pronounced.

4.1 Alpha-Beta Search

In this section the alpha-beta search will be explained. Therefore, first of all the minimax algorithm is described. After that the extension known as alpha-beta pruning is discussed which is the essential improvement of the alpha-beta search.

4.1.1 Minimax Algorithm

The minimax algorithm is intended to be applied to two player games [13]. One of the players is called the max player and the other one is the min player. The max player is the next player to move. His goal is to find a sequence of moves that leads him to win the game. Thus, he searches the game tree down to the leaves. Every leaf represents a terminal position in the game. Therefore, every leaf has a state in which the max player wins or loses. What the max player actually has to do is to find a sequence of moves that leads him to a leaf in which he wins the game, if possible. To evaluate a game position, an evaluation function as already described in section 4.3.1 is used. One evaluation function can be the one described above which simply evaluates the terminal position. Since his opponent, the min player, may choose moves he wants to, the max player cannot be sure about how the exact path down to the leaf will look like. Therefore, the max player tries to choose a path that gives him the most promising position regardless of what move his opponent is going to take. Thus, the max player tries to maximize the value function. If we assume that

the min player acts rationally his goal is to minimize the value function. Under these assumptions the minimax algorithm works as follows:

```
1. Generate the whole game tree, from the root to all terminal
    positions.
2. Calculate the value for every terminal position, i.e. for all
    leaves
3. Calculate the values of all other nodes by propagating the
    values of the leaves towards the root
    a) The value of a max player node is always the maximum among
        its child nodes
    b) The value of a min player node is always the minimum among
        its child nodes
```

A simple example taken from [19] is given in figure 4.1. Here a more sophisticated value function is used leading to different values in the leaves between 2 and 14. The branching factor is 3 meaning that there are 3 possible moves in each turn. In the left of the graph the terminal positions have the values 3, 12 and 8. Since the min player tries to minimize the value it chooses the move that leads to value 3. Therefore, the value of the min player's node is 3. Using this method one finds for the other two nodes the value 2. The root is the max player's node. He chooses the path leading to the highest value. Therefore, it decides for the move leading to the node with value 3.

This example is very simple since the tree has only a depth of 2 and a branching factor of 3. However, for more complex trees the algorithm works in the same way.

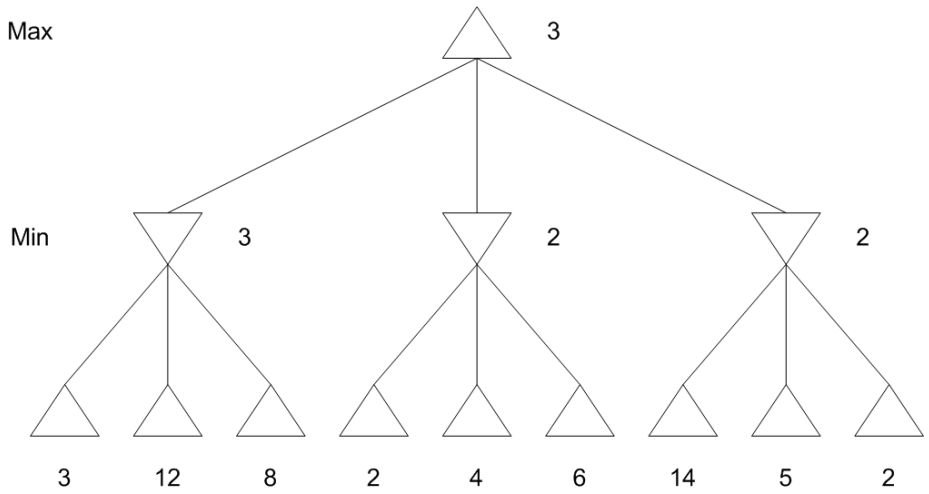

Figure 4.1: Sample minimax tree

Note that the number of nodes per tree level increases exponentially. Hence a higher branching factor leads to a much higher number of nodes in the tree. This results in an exponential growth of time needed to search the tree. If the tree has a maximum depth of m and there are b possible moves in each position, the complexity of the minimax algorithm is $O(b^m)$. Since Abalone's game tree has an average branching factor of about 60 (see section 3.2) and a much deeper tree the searching of the whole tree is intractable. Thus, the value function cannot be applied to terminal positions, but the game tree can only be observed to a certain depth. The objective is to search the tree as deep as possible since the deeper the tree is searched the more accurate the values of the tree nodes are.

4.1.2 Alpha-Beta Pruning

The most significant drawback of the minimax algorithm is that the whole game tree has to be searched. Since time requirements grow exponentially minimax allows only a relatively small tree depth to be searched. A well-known approach to handle this problem is alpha-beta pruning. It eliminates branches from being searched. The resulting move is always the same as in minimax. So there is no loss of accuracy, but only improvements regarding the computational effort. Donald E. Knuth and Ronald W. Moore proved the correctness of alpha-beta pruning in 1975 [13].

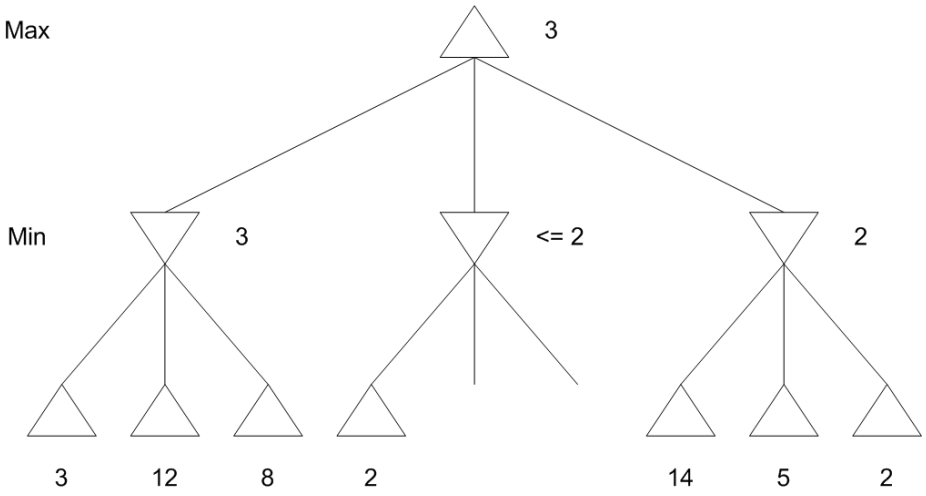

Figure 4.2: Sample minimax tree after alpha-beta pruning

An example to show how the pruning works is given in figure 4.2. It is taken from [19] as well. It is the same tree as the one seen before in figure 4.1. Suppose the leftmost 3 leaves were considered and their values are already computed. The min player has the chance to minimize the value to 3. So the

best move the max player can choose at the root is actually this 3. Then A_2 is considered as possible move. The first leaf in this branch leads to a value of 2. Thus, the min player would be able to choose a move which is less or equal to 2 for sure. Since the max player has already the chance to take a move leading to a value of 3 he will not choose the second branch. Thus, after observing the value 2 in the leaf the whole branch is pruned and the two other leaves of that branch need not to be considered.

The pruning is implemented by introducing two bounds: the alpha as a lower bound and the beta as an upper bound. Again, after the leftmost 3 leaves were investigated the max player at the root knows it can achieve at least a value of 3. Thus, the minimum bound is $alpha = 3$. When the first child of the second min node is investigated the min player can achieve at most 2. Therefore, the beta value is 2. In that case the two remaining leaves are pruned, because $beta < alpha$.

Iterative Deepening

One problem of the alpha-beta search is that you cannot foresee how much time is needed to search a specific depth in the game tree. Thus, using a static defined tree depth is not a solution. Furthermore, playing Abalone with official championship rules leads to a predefined time each player has for the whole game. So there is a need to implement the search in a way where one can define a time range that is not exceeded. A well-known approach is the so-called iterative deepening [14]. Its basic idea is that the search starts to find a solution with a tree depth of 1. After that search is done the time is checked. If there is still time left for the current search the alpha-beta search is repeated, but now with tree depth 2. This is repeated until the time is over. After implementing this simple algorithm some problems emerge. First of all it is not known if the next search with increased tree depth finishes in time. Therefore, it is useful to save the durabilities of the previous searches. To make a good prediction of the time the next search seems to take the ratio between the two previous searches is calculated. By multiplying the time of the last search by this ratio one can make a guess for the next search. For example, assume that the search with tree depth 3 took 2 seconds, the search with tree depth 4 took 10 seconds. In that case the search with a tree depth of 5 will probably take about 50 seconds. With this information it can then be decided whether the next search is started or not.

As long as the time predictions are accurate this approach will work well, but if the real time needed will exceed the predicted time significantly the player consumes time that he does not have. To prevent that an alarm timer is implemented. After investigating a certain number of nodes (e.g. after 200 nodes) the algorithm checks the time. If time is over the search stops. So the tree search is stopped occasionally. Since it is not finished it does not generate a useful result. So in general that case should be avoided.

The disadvantage of this algorithm is that in every tree search all the steps of the previous search must be repeated. That can be compensated by a technique

that leads to a better move ordering. It is described in section 4.1.3.

4.1.3 Improvements

In this section some approaches to improve the plain alpha-beta search are explained in general and in particular how they are implemented for the Abalone playing agent. The results of their performance tests are discussed in chapter 5.

Move Ordering

Since the minimax algorithm always searches the whole game tree the order the possible moves are investigated makes no difference for the calculation time. In contrast to minimax it makes a difference for the alpha-beta search. The alpha-beta algorithm searches the game tree regarding of which moves were already investigated and how well they performed. Thus, the max player only investigates game tree branches where it can exceed the value of the best move found so far. Therefore, the search algorithm should consider the most promising moves first to maximize the chance of tree branches being pruned [8, 19].

In Abalone the strongest moves are those that attack the opponent or those that keep marbles together in a group for a better defense. Therefore, moves consisting of several marbles are often more valuable. So first of all moves with three marbles are investigated. Then the moves containing two marbles follow and moves containing only a single marble are investigated last. The calculation time decreases dramatically as shown in chapter 5.

An additional ordering can be made by taking the type of move into account. So the distinction between inline moves and sidemoves is made. Since attacking is only possible by performing an inline move these are most often the stronger ones. Sidemoves can be worth, but most often they are valuable when the player is defending or has to escape from a position where he is attacked by his opponent. Furthermore, the inline moves are ordered by their strength, i.e. capturing moves are investigated before attacking moves somewhere in the middle of the board. Inline moves that do not push opposing marbles are investigated last. So the ordering could be further improved by ordering the moves in the following way:

1. 3 marbles, capturing
2. 2 marbles, capturing
3. 3 marbles, attacking
4. 2 marbles, attacking
5. 3 marbles, inline
6. 2 marbles, inline
7. 3 marbles, sideways

8. 2 marbles, sideways
9. 1 marble

Since a single marble move can never attack the opponent it is more like the sideway moves than like the inline moves.

An alternative to the move ordering above is to sort the moves in descending order with respect to the number of marbles, but moves with equal number of marbles are ordered by their type:

1. 3 marbles, capturing
2. 3 marbles, attacking
3. 3 marbles, inline
4. 3 marbles, sideways
5. 2 marbles, capturing
6. 2 marbles, attacking
7. 2 marbles, inline
8. 2 marbles, sideways
9. 1 marble

Additionally, the ordering can be further improved due to the fact that iterative deepening (4.1.2) is used in the alpha-beta search implementation.

The disadvantage of iterative deepening is that in every tree search all the steps of the previous search must be repeated. That can be compensated by using information of the previous searches. This technique arises due to the iterative fashion of the algorithm. It works as follows. In every iteration the possible moves at every node in the tree are ordered as described above. Additionally the moves of the principal variation found by the previous iteration are investigated first. Since this ensures that the most promising moves are tried first we assume that the best pruning is achieved. In many games this advantage outperforms the disadvantage of researching some nodes.

Killer Moves

The killer-move heuristic [4] is an approach to minimize the number of investigated nodes in a search. It has proven to work well in other board games. Thus, it was also implemented in the Abalone player.

The killer-move heuristic assumes that if a move led to a pruning at a certain depth it may lead to a pruning again in another node at the same depth. Therefore, such a move should be investigated first provided that the move is legal in the current position. By using the killer-move heuristic one assumes

that if a move is the best one for a certain position it is probably also the best one for similar positions.

In Abalone for each depth in the search tree some killer moves are stored. Section 5.1.2 describes experiments where the number of killer moves per depth is set to 1, 2 and 3. In each node first of all the killer moves are checked whether they are legal moves in the current position or not. The moves that are legal are investigated first.

Transposition Table

Basics Another improvement that was tested for utility in Abalone is a transposition table [7]. It is based on the fact that in a board game so-called transpositions can appear. Imagine the following case: The game is started with the Standard formation. Player 1 takes the move A5B5. Afterwards, player 2 takes the move I5H5.Then player 1 moves A1B2. This move sequence leads to the position shown in figure 4.3.

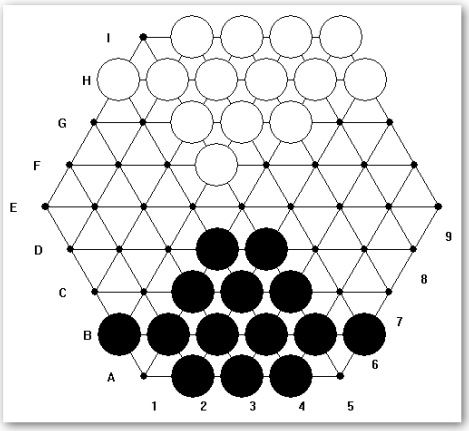

Figure 4.3: Position after player 1's second move

If player 1 would first take move A1B2 and as his second move A5B5 both paths in the game tree would lead to the same position. So one path is a transposition of the other one. Obviously, both positions have the same value since they are equal, but during alpha-beta search each transposition of a position is searched. This is a waste of time, because if you found a value and its corresponding best move for this position once the retrieved information can be reused and does not have to be calculated again.

The basic idea for a transposition table is as follows:
Imagine a certain node in the game tree that represents a certain game position. The subtree of this position is searched to find the value of this position. After the search is done, the information about the game position is saved together with the found value.

If a transposition of a position already searched is reached during alpha-beta search the information of the previous node is reused.

Which information can be used depends on the node that was searched before. First of all the value of a game position can only be reused if the depth of the saved game position's subtree is greater or equal to the depth the basic alpha-beta search would search for the current position. An example is illustrated in figure 4.4.

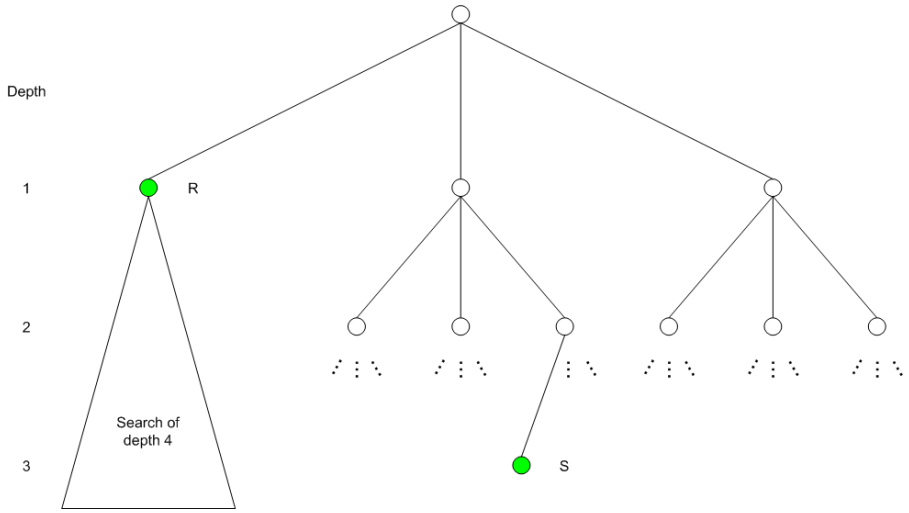

Figure 4.4: Using a value of a previous transposition

Imagine that the alpha-beta search would search to a depth of 5. So usually the search from node S would go 2 levels deep. Since S is a transposition of the game position at node R the information of node R can be reused, because the search from R goes 4 levels deep which is better than a depth of 2.

If it is the other way around, i.e. the node searched before was on level 3 and the current node that has to be searched is on level 1 the value cannot be reused, because a search that has to go 4 ply deep cannot be substituted by a value of a search 2 ply deep. At least the information can be used for move ordering. Since there is a move that was best for the search of the shallower depth before it is the most promising one and should be investigated first.

The second problem arises from the pruning. If a complete minimax search would be performed you always find an exact value for a node. Thus, it is possible to save this exact value in the transposition table. If this information is reused in a node deeper in the game tree, there is no effort at all. The value of the search before can simply be reused. However, in alpha-beta search values are not always exact. If subtrees of a node are pruned the result is only a value bound either a lower or an upper bound.

That leads to the following:

1. If the value of the game position saved in the transposition table is the result of a search in a shallower tree than the search for the currently investigated node would be the saved information is only used for move ordering.

2. Otherwise, the value can be reused. If the value is a lower or an upper bound it is used to reset the alpha or the beta which may lead to a narrower search window.

3. If the value is an exact value it is just reused for the currently investigated node.

Until now it is only described to reuse information, but it can be also smart to update the saved information. For example, if a transposition of a game position is found in a shallower depth of the tree than the transposition that is currently saved in the transposition table it should be replaced, because a transposition in a shallower depth can be reused by more transpositions than a saved transposition at a deeper level. Additionally, a value resulting from a search in a deeper tree is more accurate than a search of a shallower one.

Implementation After explaining the idea of a transposition table this paragraph explains how this concept is implemented in Abalone.

In the transposition table there is one entry per game position. Checking whether a position is saved in the table or not has to be very fast since this check is performed at every node in the game tree. Therefore, a hash table using a smart hash function is used.

The utility of the hash table strongly depends on the quality of the hash function. The hash function calculates an integer value for a given position. It is required that different game positions lead to different hash values with a probability of almost a hundred percent. Otherwise collisions occur and values read from the table are not reliable.

A sophisticated hash function which is used in many different board game implementations is the Zobrist hashing called after Albert L. Zobrist [25]. In the following the main idea of this hashing method is explained.

First of all one has to initialize some random numbers. For each of the 61 fields on the game board one has to calculate two random numbers. One random number for the case that a white marble is on the field and one that there is a black one. Additionally, one needs a random number for each player to indicate who is the player to move next. These values are generated at the beginning of the game and they stay the same for the whole game.

The hash function works as follows: For each field the value corresponding to whether a white or a black marble occupies the field is read from the initialized values list. Afterwards, all these values are combined by the XOR operator. Then, the number of the player who is to move next is also combined to the result by the XOR operation. The resulting integer is the full index for the game position. First the XOR operator is very fast and second there is no need to calculate the index after each move from scratch. It is possible to recalculate

only the values of the fields that are involved in the move the player takes. For example if an inline move with three marbles is performed without moving opposing marbles it is sufficient to compute two XOR operations: The first empty field in the direction of the move turns from no marble to black or white marble. The last field turns from black or white marble to no marble. The latter can also easily be calculated by applying the XOR operator since a XOR $a = 0$. Additionally, the player who moves next also changes. Therefore, one has to perform two more XOR operations after every move.

An essential question is how long should the used random numbers and thus the indexing numbers be? The length of the index is to be chosen so that the probability of having collisions and other errors is almost zero. The probability of making an error is calculated by

$$P = 1 - e^{-\frac{M^2}{2N}}$$

where $N = 2^n$ is the number of different hash codes and M the number of game positions mapped to the hash table. Thus, using 64 bit indices leads to the following probability for Abalone: Assuming that mostly a search at a depth of 6 or 7 is performed leading to an absolute maximum number of investigated nodes of 10,000,000 the probability having an error is

$$P = 1 - e^{-\frac{10{,}000{,}000^2}{2 \times 2^{64}}} = 2.710501754 \times 10^{-6}$$

Using 64 bit numbers results in having $2^{64} = 1.844674407 \times 10^{19}$ different possible numbers. It is not possible to build an array of such a size. Therefore, only the 20 least significant bits are used for the index in the array leading to an array having $2^{20} = 1048576$ entries. The other bits are saved within the entry so that they are still used for matching transpositions.

The information that is saved as an entry in the hash map contains:

- The tree depth of the search the value is calculated with
- The value of the game position
- The type of the game position's value, i.e. upper bound, lower bound or exact
- Complete hash key containing all 64 bits
- The best move for this position

In Abalone there are many transpositions, because all marbles of a player are identical. There are no different pieces like in chess. Thus, the number of investigated nodes per alpha-beta search for a move can be redruced dramatically as can be seen in section 5.1.3.

Quiescence Search

One of the biggest problems using alpha-beta search is the horizon effect. It emerges in situations where the evaluation of the position changes drastically. An example for that are capturing moves. If the player searches in the game tree to a depth of 5 and in that depth it is possible to capture a piece of the opponent the position has a high value. On the other hand if it would be possible for the opponent to capture a piece just one step further, the player should not capture the piece in order to save his own piece.

In Abalone exactly this situation can appear. When ejecting a single opposing marble one of his own marbles is on a field at the border. Thus, the opponent could eject that marble when he has attacking marbles in the right position.

Quiescence search [6] is an approach to solve the horizon problem. It tries to adapt the deliberation a human would do, i.e. do not invest too much time in bad moves, but concentrate on good moves and think deeper for such moves where actions influencing winning and losing are taking place. The algorithm distinguishes between quiescent and non-quiescent positions. A quiescent position is one where no move can change the value of the game position drastically. Otherwise the position is called to be a non-quiescent position. These are positions as described above. So non-quiescent positions are all positions where any marble is attacked.

In the following it is described how quiescence search is added to the basic alpha-beta search. First of all alpha-beta search is executed to the defined tree depth. The leaf node is now checked for being either quiescent or non-quiescent. If it is quiescent the board position is evaluated and the search finishes as usual. If the node is non-quiescent the quiescence search starts. From the leaf position a further alpha-beta search is started regarding only the capturing moves. It does not stop until all new leaf positions are quiescent.

4.2 Evaluation Function

In a minimax tree search all nodes and thus all leaf nodes are investigated. The leaf nodes serve a final outcome of the played game path since they represent terminal game positions. Therefore, an investigated position at a leaf is either a win, a draw or a loss. This information is propagated upwards to the root node where the player is now able to assess which move is the best to take in order to maximize the probability of a win. This idea is straightforward and leads to an optimal behaviour of the game-playing agent.

For the alpha-beta search it is not that simple. Search does not reach the terminal positions in the game tree, because of the high branching factor. It is computationally impossible to do so. Therefore, the search stops at a certain depth. Since the position is in general not a terminal position it is not possible to assess it by either a win, a draw or a loss, because none of these cases applies to the situation. Instead one has to evaluate the game position not by the

outcome of the game, but by some other features representing the value of a position.

For all implementations of an alpha-beta search the evaluation function [19] is essential. It has to be both fast and accurate in assessing a given game position. To achieve a good assessment one needs to find features, which can be evaluated for a given game position. Every feature then contributes with a certain weight to a final value. This value is the quality of the game position. By these values game positions are compared. The maximizing player chooses at the root the move that is contained by the path leading to the most promising position at the lowest level of the search tree.

For Abalone it is possible to find several features that seem to contribute to an accurate evaluation. The following features are implemented. Experiments to find a good evaluation function, i.e. to find a good weighting between the features, are described in chapter 5.

1. **Win or Loss**
   At the end of a game it may happen that game positions are reached in the search tree that are terminal positions. In that case the evaluation function should measure the outcome of the game, i.e. win or loss. Since winning is the best position that can be reached the evaluation function should return a very high value exceeding all other features. On the contrary a losing position leads to a very low value making the player avoiding this position. A non-terminal position is valued as a 0.

2. **Lost Marbles**
   Since the objective of the game is to eject the opposing marbles off the game board the number of marbles on the board is important. First of all ejected opposing marbles give a positive reward since it brings you nearer to your goal. On the other hand it is important to keep own marbles on the board to not lose the game. Furthermore, ejecting an opposing marble weakens the position of the opponent while it strengthens the own position, because the more marbles one has the better one can attack and defend.

3. **Center Distance**
   A player should try to occupy the center of the game board. It gives the player a strong position. First marbles occupying board fields at the edges are at risk to be ejected by the opponent. So own marbles should avoid the border of the board. Secondly, occupying the center of the board forces the opposing marbles to stay around. They are not able to group together and they are kept at the outside. Thus, both attacking and defending is better when marbles are at the center than at the borders. The center distance is measured by the so-called Manhattan distance. It counts the fields that have to be visited before reaching the middle field. The distances of every marble to the center are added. The best (lowest) value that can be achieved is 20. The board position where all marbles are in a group symmetrically around the center. The worst value is 56. In that

case all marbles are at the border of the game board. These two numbers are considered to establish a return value representing the quality of the Center-Distance feature.

4. **Own-Marbles Grouping**
   The next important thing a player should try is to keep marbles together in groups. Having a single marble on the board is bad, because it has not the ability to attack the opponent. To attack at least two marbles are needed. On the other hand it is even not able to defend. It can easily be attacked by two opposing marbles. Therefore, a group of marbles gives you both a higher value for attacks and for defending opposing attacks. The strongest position is reached by having all marbles stick together in one group. To measure the "grouping factor" of the marbles for every marble the number of neighbouring marbles is counted. All of these are added and give the final number for the grouping measurement. The best value for grouping is 58, having all marbles in a huge single group. The worst value is 0, meaning no marble has a neighbour. According to these bounds a return value is calculated.

5. **Opposing-Marbles Grouping**
   Since grouping of marbles gives a strong position it is valuable to hamper the opponent grouping his marbles together. Since the best grouping for the opponent is 58, this is the worst value for the own player. 0 is the worst value for the opponent. Thus, it is the best value for the own player.

6. **Attacking Opponent**
   To beat the opponent and to eject opposing marbles off the game board one needs to attack. Attacking is done by so-called Sumito moves as described in section 2.3. To make such a move possible the player has to reach a Sumito position. These are positions where the player can attack the opponent. The Attacking-Opponent feature counts all attacking positions in a given game position. The worst value that can be reached is 0. It is the case if there are only pac situations or opposing marbles do not occupy neighbouring board fields. The best value that can be achieved in theory is 19. The corresponding position is given in figure 4.5, but in general this position will not be reached in a real game.

7. **Attacked By Opponent**
   This feature also measures attacks, but it measures the attacking positions of the opponent on own marbles. Thus, a lower attacking value is better for the player.

Figure 4.6 illustrates an example game position. Underneath, the measurements of each feature are given.

Example of measuring the features for the black player:

1. **Win or Loss**
   Value: 0

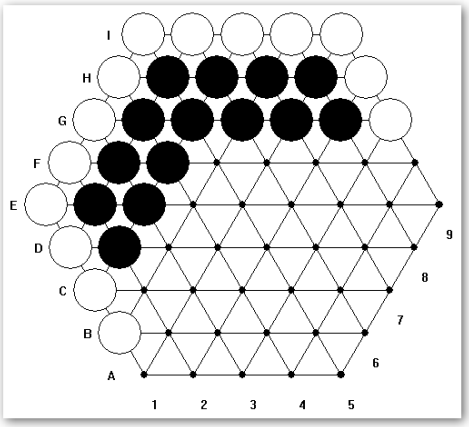

Figure 4.5: Game position with best attacking value for the black player

2. **Lost Marbles**
   Value: +1
3. **Center Distance**
   Value: 28
4. **Own Marbles Grouping**
   Value: 32
5. **Opposing Marbles Grouping**
   Value: 23
6. **Attacking Opponent**
   Value: 6
7. **Attacked By Opponent**
   Value: 3

4.3 Monte-Carlo Search

One of the most famous search techniques is the Monte-Carlo search [10]. Its name is based on the casino in Monte Carlo. That is because the search is based on statistics and probabilities as it is the case in gambling. Due to the law of large numbers in some games Monte-Carlo search is an accurate method to determine good moves in a certain game situation.

4.3.1 Basic Monte-Carlo Search

The general idea is to simulate a certain number of games starting at the current real situation on the game board. For each possible move a number of games

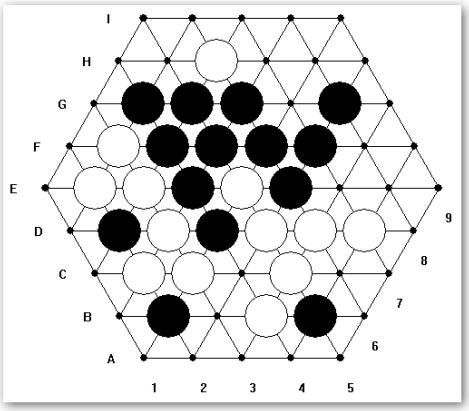

Figure 4.6: Example game position

are simulated and the outcome of each of these games is scored. After all simulations are finished, the player takes the move with the highest winning since it is probably the most promising move.

For a better explanation, here is an example of the method: Suppose it is the first player's turn which is a player that uses Monte-Carlo search. There are 60 possible legal moves in the current game situation. The player now has to simulate several games for each of these possible moves. It starts by taking the first move. Starting at this position it simulates now a number of games, e.g. 100. After every simulation it scores the outcome of the simulated game. This is done by either simply scoring if the player wins $(+1)$, loses (-1) or if the outcome is a draw (0) or by a more sophisticated value function (see section 4.2). The scores of the 100 games are summed up and the simulations are carried out for the next move. This goes on until for every move the simulated games are finished. At the end the move with the best rating is determined. This is the move the player takes in the real game.

But what is meant by "simulating a game"? A simulation game is a game between two players that take their moves randomly. More precisely that means that the player who moves next determines all his legal moves in the certain game situation and chooses absolutely randomly which move to take. This sounds not very useful at first glance, but if the number of simulated games per possible move is large enough, it could lead to quite adequate scores.

Another important aspect is the valuation of the simulation outcomes. The basic method to simply score the outcome of the game which is winning or losing is only possible if the simulation games run until the game ends. Of course playing to the end will probably give the best results, but most often it is not possible to play to the end, because it would take to much time. In that case the simulation does not run to the end, but stops after an ending condition is reached. This can be for example a certain number of turns or a predefined

time range. In this case it is no longer possible to evaluate who wins, because there is no winner. A value function is needed which can evaluate the current game situation and returns a measure of who is in a better position, often called the winning position.

For convenience the Monte-Carlo search is explained by means of pseudo code.

```
1. Get all possible moves in the current situation of the game
2. For each of these moves
    a) Take the current move
    b) Simulate a number of games with the actual situation
    c) Evaluate the outcome of each simulated game and accumulate
       them
3. Take the move with the highest value for the regular game
```

Random players that choose a move from all possible moves are used in the plain Monte-Carlo search. To improve the algorithm it is possible to add knowledge so the random players do not really choose their moves randomly, but they prefer more promising moves.

Time Limit

A fundamental question according to Monte-Carlo search is: How many games should be simulated for every possible move?

First of all it is possible to define a fixed number that is the number of simulated games per possible move. That is obvious and easy to implement, but it is not predictable how long the search will take since every game position offers another number of possible moves to the player. Since Abalone's official championship rules contain that each player has a certain amount of time for the whole game there is a need to implement the search in a way where one can define a time range that is not exceeded.

Therefore, the solution is not to define a fixed number of simulated games, but to calculate this number using the number of possible moves for the current situation, the time the player has for the whole game and the average number of plies for a whole Abalone game. The result is the time available for the simulations of one move. When time is up games for the next move are simulated.

$$TimeForCurrentMove = \frac{LeftTimeForWholeGame}{\#AvgPliesOfPlayer \times \#PossibleMoves}$$

By this technique the search for the next move to perform always stops in time. Thus, for every move the maximum number of games are simulated according to the time limitation.

4.3.2 Improvements

Depth-Limited Monte-Carlo Search

In general a Monte-Carlo search simulates several games for every possible move to the end. Thus, every game results in a win or a loss. The disadvantage is that it is not possible to estimate accurately how long a game will last until it is finished. But if there are some games that take a long time it may happen that there is not enough time to simulate some games for every possible move. In that case some moves are not evaluated at all. As a result good moves may be overseen.

To avoid this problem the duration of a simulated game should be limited in order to have an equal number of simulated games for every move. This is done by setting a maximum number of plies for a simulated game. If the game arrives at a terminal position before it reaches the limit it stops. In this case, the outcome is a win or a loss. Otherwise, i.e if the game arrives at the maximum number of plies before reaching a terminal position the game stops as well. In that case the outcome would be a draw.

Evaluation of Simulated Games

The time problem is solved by the approach discussed in section 4.3.2, but another problem arises. Since games are not finished necessarily it can happen that no winner is determined. Thus, the final outcome of the game is a draw. If there are only a few games that end in a draw it is not a real problem, but if the maximum number of plies per simulated game is set too low most of the games become a draw leading to a non-accurate evaluation of the possible moves for the player who is to move.

This problem can be solved by refining the evaluation. The most general and obvious evaluation is, as already described, to determine if the outcome of a game is a win, a draw or a loss. A more precise approach is to count the lost marbles of each player. Since these numbers are the most important to assess the strength of a player's position they give a more precise estimate of the current position's quality than simply calling it a draw.

In the final Abalone implementation the limitation of plies per simulated game is implemented. The simulated games are evaluated accoring to the number of each player's lost marbles.

Chapter 5

Experiments and Results

In this chapter all experiments done to assess the different algorithms and their improvements are discussed. First of all the alpha-beta player is tested. Afterwards, there are some explanations about the Monte Carlo player. Finally both are compared.

5.1 Alpha-Beta Search

This section describes the experiments that were done to test the performance of the alpha-beta search. The different improvements are tested and a sophisticated evaluation function is determined.

To achieve comparable test results for the experiments described in sections 5.1.1 to 5.1.4 the same evaluation function is used. The weights of the features that contribute to the value of a position are listed in table 5.1.

5.1.1 Move Ordering

The most basic improvement for an alpha-beta player implementation is the ordering of possible moves to achieve the best pruning. In Abalone moves with multiple marbles are more promising than moving a single marble since attacking and defending can only be performed with multiple marbles. Therefore, a first basic ordering is to first investigate moves with three marbles, then two marbles and finally single marbles. This is referred to as move ordering 1.

Another approach for ordering is to order the moves according to their type. This second approach orders the moves as follows:

1. 3 marbles, capturing
2. 2 marbles, capturing
3. 3 marbles, attacking
4. 2 marbles, attacking

Feature		Value
Own Lost Marble		$-1,000$
Opposing Lost Marble		$+1,000$
Own Lost Marbles = 6 (Loss)		$-1,000,000$
Opposing Lost Marbles = 6 (Win)		$+1,000,000$
Center Distance of All Own Marbles	< 24	$+400$
	< 30	$+300$
	< 35	$+200$
	< 40	$+100$
Grouping of Own Marbles	> 55	$+320$
	> 50	$+240$
	> 45	$+160$
	> 40	$+80$
Attacking Power on Opponent	per attack	$+10$
Attacking Power on Own Marbles	per attack	-10

Table 5.1: Evaluation function

5. 3 marbles, inline
6. 2 marbles, inline
7. 3 marbles, sideways
8. 2 marbles, sideways
9. 1 marble

As an alternative it is tested to sort the moves in descending order with respect to the number of marbles, but moves with equal number of marbles are ordered by their type:

1. 3 marbles, capturing
2. 3 marbles, attacking
3. 3 marbles, inline
4. 3 marbles, sideways
5. 2 marbles, capturing
6. 2 marbles, attacking
7. 2 marbles, inline
8. 2 marbles, sideways
9. 1 marble

This ordering is referred to as ordering 3.

In the following tables the different approaches of ordering are compared. No ordering means that first the single-marble moves are investigated followed by the two-marbles and three-marbles moves without further ordering. The tables 5.2, 5.3 and 5.4 show the number of nodes that are investigated in the certain game position using the different orderings. The mid-game position is the one illustrated in figure 4.6. For these experiments a plain alpha-beta player is used.

Tree	Investigated Nodes			
Depth	No Ordering	Ordering 1	Ordering 2	Ordering 3
1	45	45	45	45
2	175	132	132	175
3	2,552	2,423	2,552	2,423
4	9,380	6,918	11,543	6,918
5	427,278	227,504	187,286	266,290
6	1,213,384	823,541	824,568	1,074,184
7	51,212,918	31,584,699	35,432,824	33,645,840
8	239,733,948	90,206,750	128,863,524	153,160,920

Table 5.2: Move Ordering: Investigated nodes Standard formation

Tree	Investigated Nodes			
Depth	No Ordering	Ordering 1	Ordering 2	Ordering 3
1	53	53	53	53
2	362	318	342	354
3	13,160	4,140	3,661	4,076
4	52,969	29,062	21,231	26,944
5	4,522,378	450,324	379,854	451,734
6	16,053,963	2,897,725	1,854,156	2,548,939
7	1,932,568,869	53,040,729	30,777,654	38,878,061
8	-	353,342,474	214,643,317	272,975,265

Table 5.3: Move Ordering: Investigated nodes Belgian Daisy

These results show that a basic ordering already has a noticable benefit compared to no ordering. Furthermore, the ordering 2 seems to be the most promising ordering. For the Standard starting position ordering 2 is worse than the basic ordering. The reason is that at the beginning there are only a few moves leading to captures or attacks. Belgian Daisy leads after some moves to captures and attacks as well as the tested mid-game position. When attacking or capturing moves appear the second ordering has the advantage to first investigate these promising moves leading to many prunings.

In the following experiments move ordering number 2 is always used.

Tree Depth	Investigated Nodes			
	No Ordering	Ordering 1	Ordering 2	Ordering 3
1	60	60	60	60
2	813	367	236	240
3	18,929	8,891	6,082	6,707
4	160,486	38,365	17,979	22,662
5	5,553,251	937,310	427,556	474,235
6	65,038,139	2,538,791	826,597	1,094,926
7	2,442,861,481	62,480,559	17,234,990	17,438,139
8	-	194,809,382	45,190,061	52,819,767

Table 5.4: Move Ordering: Investigated nodes mid-game position

Iterative Deepening

Iterative Deepening is on the one hand a technique to enable the player to use a specified amount of time for a move. Additionally, since several searches are performed iteratively, one uses the principal variation from the previous search for the next search first. The assumption is that this leads to more pruning. In the tables 5.5, 5.6 and 5.7 searches for several depths and several game board positions are listed. The mid-game position is the position showed in figure 4.6. The tables show the number of investigated nodes for a basic alpha-beta search and for an alpha-beta search using iterative deepening starting always with the principal variation of the search before.

Tree Depth	Investigated Nodes	
	Without ID	With ID
1	45	45
2	132	176
3	2,552	2,727
4	11,543	14,163
5	187,286	201,469
6	824,568	1,019,719
7	35,432,824	36,337,658
8	128,863,524	164,662,917

Table 5.5: Iterative Deepening Move Ordering: Investigated nodes Standard formation

The results show that the number of investigated nodes increases when iterative deepening is used. Often it is an advantage to use the principal variation of previous searches to decrease the number of nodes. Mostly, this advantage outperforms the disadvantage of re-search nodes multiple times. For Abalone it is not the case. Obviously, the re-search of some nodes is not absorbed by the better move ordering.

Anyway, iterative deepening has to be used in Abalone to afford finishing

Tree Depth	Investigated Nodes	
	Without ID	With ID
1	53	53
2	342	442
3	3,661	4,135
4	21,231	28,028
5	379,854	384,696
6	1,854,156	2,210,887
7	30,777,654	32,921,869
8	214,643,317	243,589,752

Table 5.6: Iterative Deepening Move Ordering: Investigated nodes Belgian Daisy

Tree Depth	Investigated Nodes	
	Without ID	With ID
1	60	60
2	236	247
3	6,082	5,939
4	17,979	23,258
5	427,556	450,390
6	826,597	1,268,097
7	17,234,990	18,232,930
8	45,190,061	61,108,828

Table 5.7: Iterative Deepening Move Ordering: Investigated nodes mid-game position

a search in a certain time. Therefore, all following experiments are done with iterative deepening.

5.1.2 Killer Moves

Using Killer Moves for a better move ordering should lead to more prunings. In the tables 5.8, 5.9 and 5.10 a basic alpha-beta player is compared to one using killer moves. Furthermore, it is tested how many killer moves per depth should be saved.

The tables show that in both start positions it is beneficial to use killer moves. In these two positions using one or two killer moves seems to be the best. At the mid-game position killer moves lead to a higher number of investigated nodes. Still using three killers performs worse than using one or two. Since the loss in the mid-game position is small, but the gain in both other positions is high killer moves should be used. Two killer moves per depth seem to be accurate.

Tree Depth	Investigated Nodes			
	no Killer Move	1 Killer Move	2 Killer Moves	3 Killer Moves
1	45	45	45	45
2	176	176	176	176
3	2,727	2,601	2,601	2,601
4	14,163	9,614	9,614	9,614
5	201,469	146,264	146,310	146,250
6	1,019,719	543,835	546,156	546,346
7	36,337,658	8,789,211	8,803,583	8,774,354
8	164,662,917	32,814,503	32,815,327	39,879,826

Table 5.8: Killer Moves: Investigated nodes Standard formation

Tree Depth	Investigated Nodes			
	no Killer Move	1 Killer Move	2 Killer Moves	3 Killer Moves
1	53	53	53	53
2	442	426	396	398
3	4,135	4,274	4,612	3,945
4	28,028	19,520	18,341	16,193
5	384,696	240,728	262,186	252,903
6	2,210,887	1,164,946	1,029,932	1,306,542
7	32,921,869	13,421,546	12,188,274	15,070,636
8	243,589,752	76,534,085	70,701,891	69,057,904

Table 5.9: Killer Moves: Investigated nodes Belgian Daisy

5.1.3 Transposition Table

The objective of using a transposition table is to minimize the number of nodes that have to be investigated. Since Abalone has only one type of pieces it is expected that Abalone has many transpositions within a search tree. Therefore, the number of investigated nodes should decrease dramatically when using a transposition table. A comparison between an alpha-beta player using a transposition table and one not using it is given in the tables 5.11, 5.12 and 5.13.

The results of these experiments are as expected. Using a transposition table leads to many prunings. Thus, they are useful to improve the results of the alpha-beta search.

5.1.4 Quiescence Search

Quiescence search should improve the results of the alpha-beta search in the way that it solves the horizon problem. Therefore, non-quiescent game positions are searched deeper leading to a more accurate resulting move. The player avoids ejecting moves if he gets under pressure afterwards. Some games between a plain alpha-beta player and an alpha-beta player using quiescence search are played. Both of them use a transposition table. The results of the played games

Tree Depth	Investigated Nodes			
	no Killer Move	1 Killer Move	2 Killer Moves	3 Killer Moves
1	60	60	60	60
2	247	232	234	234
3	5,939	5,922	6,348	5,784
4	23,258	21,803	22,213	21,098
5	450,390	367,276	394,865	394,643
6	1,268,097	1,223,807	1,294,076	1,254,569
7	18,232,930	19,678,793	19,948,542	22,546,049
8	61,108,828	63,795,607	62,488,941	68,719,815

Table 5.10: Killer Moves: Investigated nodes mid-game position

Tree Depth	Investigated Nodes	
	Without TT	With TT
1	45	45
2	176	176
3	2,727	2,728
4	14,163	8,568
5	201,469	120,634
6	1,019,719	283,100
7	36,337,658	8,216,593
8	164,662,917	14,286,296

Table 5.11: Transposition Table: Investigated nodes Standard formation

are listed in table 5.14.

These results show that quiescence search seems to be an advantage. During the quiescence search only capturing moves are investigated leading to very small trees with a branching factor of at most 2 for every capturing position on the board.

5.1.5 Evaluation Function

The quality of a game-playing agent using alpha-beta search strongly depends on the evaluation function. The ideas about which features can be used is already described in section 4.2. These features contribute to a total value which represents the quality of a game board position. The difficulty is to find a good weighting between these features. A first intuitive weighting is given in table 5.1.

To improve the quality of the evaluation function different configurations of features and weightings were defined. Players using these different configurations played several games against each other. By the outcome of those games it was assessed which player performs best.

To get useful results Belgian Daisy was used as start position. The reason is that when using the standard position the players play very defensive and never

Tree Depth	Investigated Nodes	
	Without TT	With TT
1	53	53
2	442	442
3	4,135	4,134
4	28,028	22,693
5	384,696	257,810
6	2,210,887	936,549
7	32,921,869	11,268,180
8	243,589,752	68,810,881

Table 5.12: Transposition Table: Investigated nodes Belgian Daisy

Tree Depth	Investigated Nodes	
	Without TT	With TT
1	60	60
2	247	247
3	5,939	5,754
4	23,258	17,453
5	450,390	363,182
6	1,268,097	843,729
7	18,232,930	9,126,680
8	61,108,828	24,003,413

Table 5.13: Transposition Table: Investigated nodes mid-game position

perform a move that offers the opponent the opportunity to attack. Thus, these games get stuck and always end in a draw. Starting with Belgian Daisy is an effective solution for that problem. Even in tournaments between human players Belgian Daisy is used to avoid defensive play.

The second interesting thing is that the player who begins the game seems to have an advantageous position. Games between similar players are mostly won by the first player.

During the experiments several weighting configurations were tested. One configuration was extended by regarding the grouping of the opposing marbles which is not in the configuration illustrated in table 5.1. It turned out that it is not beneficial. Other ones put more weight on the attacking positions. Another configuration plays more defensive since it gets a reward of $1,000$ if the opponent loses a marble, but it gets a reward of $-3,000$ if it loses an own marble. A last configuration has a higher weight on the grouping than on the center distance feature.

The results of the experiments show that the first intuitive weighting in table 5.1 was already very good. Just the weighting of the attacking points was changed from 10 to 40 in the final evaluation function.

Allowed	As Black			As White			Win
Time	Win	Draw	Loss	Win	Draw	Loss	Rate
600	13	4	3	7	6	7	62.5 %
900	11	4	5	9	6	5	62.5 %
1200	12	5	3	4	6	10	53.8 %

Table 5.14: Played games using quiescence search

5.2 Monte-Carlo Search

This section describes the experiments that were done to test the performance of the Monte-Carlo search.

5.2.1 Simulated Games using Random Move Players

The strength of Monte-Carlo search depends on how games are simulated. To get an idea of how long a game between two random players last 1,000 games starting at the Belgian Daisy position were simulated. For each game the number of played plies to reach a terminal position were counted. These numbers range from 351 to 3,752. On average a game ends after 1370 plies.

Since games should not always be played to the end, but they should be limited to a certain number of plies it was also tested how the game outcome is if a simulated game is limited to 200 plies. It turned out that the losing player, i.e. the player with more lost marbles, has between 0 and 5 lost marbles. On average the number of lost marbles is 1.37. Playing 300 plies leads to an average number of lost marbles of 1.92. Playing 400 plies leads to an average number of lost marbles of 2.54.

It was also tested how many games per possible move are simulated. For this test simulated games were limited to 200 plies and the advanced evaluation, i.e. counting lost marbles, was used. Results are given in table 5.15.

Allowed	Start	# simulated games	
Time	formation	per move	total
900	Standard	294	12,936
900	Belgian Daisy	290	15,080
900	mid-game position	285	16,815
1200	Standard	393	17,292
1200	Belgian Daisy	380	19,760
1200	mid-game position	372	21,948

Table 5.15: Monte-Carlo player: Numbers of simulated games

It shows that if a player has 900 seconds for the whole game, it uses less than 20 seconds for the first move. Within these 20 seconds the player is able to simulate more than about 13,000 games. Thus, for every possible move almost 300 games can be simulated.

5.2.2 Evaluation of Simulated Games

Since the games are not always simulated to a terminal position, but they stop if a limit of plies is reached, it can be an advantage to not only evaluate an outcome by determining whether it is a win, a draw or a loss, but to use the difference between the lost marbles as a measure. Table 5.16 shows the outcomes of played games of a Monte-Carlo player counting lost marbles versus a Monte-Carlo player using the basic approach. Each player simulates games with a maximum of 200 plies. Again, games start at the Belgian Daisy position. Since the players were not able to finish a game within 200 plies it is not possible to count wins and losses. Instead the lost marbles of each player are summed up and listed in the table. For each configuration 15 games were played.

Allowed Time	basic vs. improved lost marbles		improved vs. basic lost marbles	
	basic	improved	improved	basic
900	17	1	5	25
1200	12	2	3	22

Table 5.16: Comparison of basic and improved Monte-Carlo player

From these numbers we can conclude that counting the lost marbles is more accurate than just assessing wins, draws or losses. Thus, the Monte-Carlo player should use the improved evaluation.

5.3 Comparison of Alpha-Beta Search and Monte-Carlo Search

In this section both algorithms are compared. Some Belgian Daisy games are played between an alpha-beta player and a Monte-Carlo player. Table 5.17 lists the game results.

Allowed Time	Winner		Winner	
	AB	MC	MC	AB
900	15	0	0	15
1200	15	0	0	15

Table 5.17: Alpha-Beta player vs. Monte-Carlo player

It is obvious that alpha-beta search is much better. The Monte-Carlo player is not able to win a game. During all 60 games the Monte-Carlo player was able to eject three opposing marbles together. All these games took between 15 and 40 plies. Thus, alpha-beta achieved fast wins since the average number of plies for one game is 87.

The conclusion of this experiment is that alpha-beta search is a good approach to use for Abalone. Using Monte-Carlo search does not lead to a good

reasoning about which move is best.

5.4 Comparison of Alpha-Beta Search and Another Program

This section gives some results of games between AIBA and another Abalone program that is available on the internet.

The ABA-PRO agent implemented by Tino Werner et. al mentioned already in the introduction in section 1.3 is available for free. Therefore, it is a good test candidate. Today it is probably the best Abalone program. It has even beaten the 3 times world champion of Abalone – Gert Schnider [3]. AIBA is tested to several strength levels of ABA-PRO. ABA-PRO does not support defining a time limit to the computer player. The player with strength 9, i.e. it searches the game tree to a depth of 9, needs between 15 and 30 seconds to generate most of the moves. Sometimes it needs even several minutes, but that is seldom. All in all the player with strength 9 seems to be a fair comparison to AIBA using a time limit of 1200 seconds for the whole game. Thus, the comparison between these two players is the most accurate. Several games between these two players are played. All of them start at Belgian Daisy formation. If a game is not finished after 200 plies it is called a draw. That is fair, because the time AIBA uses for a move keeps decreasing. Thus, at a certain point it does not longer search to the usual average depth of 6, but at a shallower depth which would lead to a distorted test result. The results are listed in table 5.18. AIBA 1200 indicates that AIBA uses 1200 seconds for the whole game. The number behind ABA-PRO gives the strength level of the player.

Player 1	Player 2	Lost Marbles	Winner
AIBA 1200	ABA-PRO 3	1 : 6	AIBA 1200
ABA-PRO 3	AIBA 1200	4 : 5	Draw
AIBA 1200	ABA-PRO 4	3 : 4	Draw
ABA-PRO 4	AIBA 1200	6 : 3	AIBA 1200
AIBA 1200	ABA-PRO 9	6 : 5	ABA-PRO 9
ABA-PRO 9	AIBA 1200	4 : 6	ABA-PRO 9

Table 5.18: Played games between AIBA and ABA-PRO

It turned out that AIBA can keep up with ABA-PRO 3 and 4. It achieves two draws and two wins.

Strength 9 is stronger than AIBA 1200. It achieved 2 wins. However, the AIBA did not perform too bad since it was able to capture 4 or 5 opposing marbles. This is because it is aggressive right from the beginning on while ABA-PRO tries to occupy the center of the game board. Thus, at the beginning the AIBA is strong and captures some marbles. In the following rounds ABA-PRO has a stronger position since it has a single group of marbles at the center of the board.

Chapter 6

Conclusions and Future Research

This chapter concludes the research that was done. It gives answers to the research questions and the problem statement mentioned in section 1.4. At the end some ideas for future research are given.

6.1 Answering the Research Questions

In section 1.4 some research questions are given. This chapter gives answers to them according to the experimental results of chapter 5.

> *RQ1: Can an Abalone player be implemented using the alpha-beta search?*

During the research one of the investigated algorithms to implement an Abalone playing agent was alpha-beta search. It turned out that it is a good approach to use since Abalone is a two-player strategic board game. The performance of the algorithm strongly depends on the quality of the evaluation function. If the evaluation function gives reliable results, alpha-beta search is a good approach to use.

> *RQ2: Can an Abalone player be implemented using the Monte-Carlo search?*

Monte-Carlo search is an approach that can be used. It works, but the resulting decisions of the player are not always reasonable. This is probably due to the fact that during the simulation games the player take absolutely randomly moves. Since the majority of possible moves in a situation consists of bad ones the players often choose bad moves. Thus, the simulation games do not represent typical games. Maybe it performs better if it is possible to increase the number of simulated games per move.

RQ3: Is it possible to implement some improvements to make both approaches making better decisions?

There are some opportunities to improve both the alpha-beta and the Monte-Carlo search. For alpha-beta search move ordering is the most important one. If a good move ordering is used the number of investigated nodes per search can decrease dramatically leading to a deeper search in the same time range. Additionally, a transposition table can further improve the search, because subtrees of transpositions do not need to be re-searched completely. The killer-move heuristic turned out to be another valuable improvement to speed up search. Quiescence search is a good way to solve the horizon problem. The extra effort is minimal since only the capturing moves are investigated to a greater depth.

For Monte-Carlo search two improvements were implemented and tested. The first one is limited Monte-Carlo search where the simulated games are limited to a certain number of plies. This is needed, because a simulation game can easily last for 4,000 plies. An average game only last for 87 plies. Limiting the number of plies in a simulation game leads to a higher number of simulated games per move since it avoids very long games.

The second improvement is not only to evaluate the outcomes of simulation games, but to count the lost marbles of each player. Since games are limited they often end up in a draw. In that case it can only contribute to the final evaluation if the lost marbles are counted. Otherwise, it would be a draw that does not change anything for the result.

RQ4: Which one of the two approaches performs better?

The comparison of both approaches clearly shows that alpha-beta search performs much better than Monte-Carlo search. Every game was won by alpha-beta search. Even in 60 games Monte Carlo search was able to eject only 3 opposing marbles. Thus, alpha-beta search is much more accurate. It should be absolutely preferred.

6.2 Answering the Problem Statement

Now that the research questions are answered this section gives an answer to the problem statement given in section 1.4.

Is it possible to implement a strong computer player for Abalone?

Yes, it is possible to build a strong computer player for Abalone. The approach that should be used to make the player strong is alpha-beta search. It leads to reasonable behaviour early in the research progress and can be further improved by some techniques as well as by enhancing the evaluation function. During research move ordering, transposition table, killer moves and quiescence search were investigated. They all contributed to a better performance of the Abalone playing agent. But there is still research to do regarding the evaluation function.

6.3 Future Research

This research gives detailed experimental results of a computer player built on alpha-beta search together with some basic techniques to improve it.

There are other approaches to further improve the search which are not investigated yet. Examples are an opening book, history heuristic [20] and aspiration search [8]. Furthermore, forward-pruning techniques like null move [24] or multi-cut [24] can also be investigated.

Another important field where further research has to be done is the evaluation function. The features currently used are valuable, but maybe there are others that can contribute to a better evaluation.

Furthermore, the time that is used for a move can be improved. Currently only an alarm timer is implemented. Therefore, the time for a move decreases with every move. It would be good to distribute the time more intelligent, e.g. to keep some time for the moves in the endgame.

As the sample game described in appendix A shows it would be an improvement to have dynamic weights on the features of the evaluation function. That would enable the player to concentrate on reaching the center at the start game. When the player has reached a strong position it then starts to attack and capture opposing marbles.

Furthermore, much more experiments can be done. It is interesting to let the agent play against other Abalone computer programs. A comparison between the agent and professional human players would also be interesting.

There are also Monte-Carlo search techniques that may perform better than the basic Monte-Carlo approach currently implemented. The so-called Monte-Carlo tree-search [9] could be implemented, which is expected to be much better than the basic Monte-Carlo search. However, seeing the enormous overwhelming power of alpha-beta search over Monte-Carlo search, it is not to be expected that Monte-Carlo tree-search outperforms alpha-beta search.

Bibliography

[1] Abalone S.A., *Abalone User's Manual*. 1997.

[2] O. Aichholzer, Abstract Strategy Games Abalone and Pyraos. *Research Stay FU Berlin*, Berlin, Germany, May 2004.

[3] O. Aichholzer, F. Aurenhammer and T. Werner, *Algorithmic Fun – Abalone*. Institute for Theoretical Computer Science, Graz University of Technology, Austria, 2002.

[4] S.G. Akl and M.M. Newborn, The principal continuation and the killer heuristic. *ACM Annual Conference Proceedings*, pp. 466-473, 1977.

[5] L.V. Allis, *Searching for Solutions in Games and Artificial Intelligence*. Maastricht University Press, Maastricht, 1994.

[6] D.F. Beal, A Generalised Quiescence Search Algorithm. *Artificial Intelligence*, 43(1):85-98, 1900.

[7] D.M. Breuker, *Memory versus Search in Games*. Ph.D. thesis, Department of Computer Science, University of Limburg, 1998.

[8] M.S. Campbell and T.A. Marsland, A Comparison of Minimax Tree Search Algorithms. *Artificial Intelligence*, 43(1):85-98, 1900.

[9] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-carlo tree search: A new framework for game ai. In Michael Mateas and Chris Darken, editors, *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 216-217. AAAI Press, Menlo Park, CA., 2008.

[10] J.M. Hammersley and D.C. Handscomb, *Monte Carlo Methods*. Methuen & Co., 1964.

[11] H.J. van den Herik, J.W.H.M. Uiterwijk and J. van Rijswijck, Games solved: Now and in the future. *Artificial Intelligence*, 134:277-311, 2002.

[12] H. Kastner, *Backgammon. Geschichte. Regeln. Strategien.* Humboldt / Schluetersche, August 2008.

[13] D.E. Knuth and R.W. Moore, An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, 6:293-326, 1975.

[14] R.E. Korf, Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, 27:97-109, 1975.

[15] N. Lemmens, *Constructing an Abalone Game-Playing Agent*. B.Sc. thesis, Maastricht University, 2005.

[16] H.J.R. Murray, *A History of Board Games Other than Chess*. Oxford University Press, 1952.

[17] N.J. Nilsson, *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill Book Company, New York, NY, USA, 1971.

[18] E. Ozcan and B. Hugalu, A Simple Intelligent Agent for Playing Abalone Game: ABLA. *Proc. of the 13th Turkish Symposium on Artificial Intelligence and Neural Networks*, pp. 281-290, 2004.

[19] S. Russel and P. Norvig, *Artificial Intelligence – A Modern Approach*. Prentice-Hall Inc., Englewood Cliffs, NY, USA, 1995.

[20] J. Schaeffer, The history heuristic and the performance of alpha-beta enhancements. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, pp. 1203-1212, 1989.

[21] C. Shannon, Programming a Computer for Playing Chess. *Philosophical Magazine*, Ser. 7, Vol. 41, No. 314, pp. 256-275, March 1950.

[22] A.M. Turing, Digital Computers Applied to Games. In B.V. Bowden, editor, *Faster than Thought*, Sir Isaac Pitman, London, 1953.

[23] T. Werner and P. Sommerlund, *The 1st ICGA Abalone Tournament*. Graz, Austria / Copenhagen, Denmark, 2003

[24] M.H.M. Winands, H.J. van den Herik, J.W.H.M. Uiterwijk and E.C.D. van der Werf, *Enhanced Forward Pruning*. Department of Computer Science, Institute for Knowledge and Agent Technology, Maastricht University, 2005

[25] A.L. Zobrist, A New Hashing Method with Application for Game Playing. *Technical Report 88*, The University of Wisconsin, 1970.

[26] The history of checkers,
`http://www.essortment.com/hobbies/historycheckers_smap.htm`,
March, 2009.

[27] Wikipedia.org - Abalone (board game),
`http://en.wikipedia.org/wiki/Abalone_game`, March, 2009.

[28] Wikipedia.org - Board game,
http://en.wikipedia.org/wiki/Board_games, March, 2009.

[29] Wikipedia.org - Game Complexity,
http://en.wikipedia.org/wiki/Game_complexity, May, 2009.

Appendix A

Belgian Daisy Sample Game

This appendix gives an example of an Abalone game between AIBA 1200 and ABA-PRO level 9. It starts from the Belgian Daisy position. AIBA 1200 plays the black marbles and starts the game. Table A.1 lists all moves that were made during the game.

Black opens the game with A1B2. This is a move ABA-PRO also prefers when it plays first. It leads to a position where no marble is in danger. ABA-PRO answers with the similar move A5B5. By moving A2B3 Black is not attacked by White, but puts pressure on the opponent since it is able to attack at two positions. White is focussed on occupying the center of the board. By moving B5C5 it moves the three marbles nearer the center. But after this move Black takes the move B1B2 leading to a position with two opportunities to capture a white marble. Therefore, White will lose at least one marble for sure. The main difference between both players in the start game and mid game is that AIBA plays aggressively right from the start. On the other side ABA-PRO concentrates on reaching the center to have a strong position for the remaining game. As a result AIBA is able to capture some marbles in this phase of the game.

After ply 36 (position in figure A.1) White has already a good position since it has all marbles in a single group at the center of the game board. The black marbles are divided into two groups one at the top and one at the bottom.

At ply 67 (position in figure A.2) Black moves B6B5. This looks stupid at first glance, but it is not. Before that move the marble at B6 is attacked and could be captured. By moving B6B5 the marble on B6 is moved away. The marble at A3 is now attacked. However, this position is better, because if White ejects the black marble on A3 it also loses a marble, since Black attacks A3 as well. Before the move White has lost 4 marbles, Black has lost 2. Thus, by these two moves both players lose a marble which is better for Black, because then it has already ejected five opposing marbles and is very close to a win. Unfortunately, although Black is leading the game White is able to win the game at the end since it has a much better position. After ply 95 AIBA (Black) realizes that it has lost the game for sure. White moves C3C4 leading to a

Ply	Move	Ply	Move	Ply	Move	Ply	Move
1	A1B2	26	F6E5	51	A2B3	76	E6D5
2	A5B5	27	B2B3	52	F5E5	77	B5B4
3	A2B3	28	B6C6	53	B3C4	78	F6E6
4	B5C5	29	H9H8	54	G6F6	79	B3B4
5	B1B2	30	C6D6	55	C4D5	80	E5E6
6	B6C6	31	A1A2	56	F4G5E4	81	A3A4
7	C2C3	32	D6E6	57	B6C7	82	B2C3
8	C6D6	33	H8H7	58	D6C5	83	B5B6
9	D4C4	34	F4F5	59	C7C6	84	E7E6
10	I5H5	35	H7H6	60	D3D4	85	A4A3
11	H7G7G6	36	D5E5	61	C6C5	86	E4D4
12	G4G5F3	37	H5H6I6	62	E3E4	87	A4A3
13	I8H7	38	C3D3	63	C4C5	88	D6C5
14	F3F4	39	I5I6H5	64	E4F5	89	I8I9H7
15	F7G7	40	E6F6	65	F2E2	90	A3B3
16	C7D7	41	I6H5	66	G6F6	91	A1A2B1
17	H7G6	42	F3F4	67	B6B5	92	E5D5
18	D7D6	43	H4G4G3	68	C3B3	93	A5B6
19	G8G7	44	G5H6	69	C5B4	94	B5B4
20	I6H6I7	45	G3F3F2	70	F5E5	95	B1C2
21	F5G6	46	I7H6	71	B5B4	96	C3C4
22	I7I8	47	A4B5	72	B2C3	97	C7D7
23	G6H7	48	H6G5	73	A5A4	98	B2B3
24	H4G4	49	B4C5	74	F7E6	99	C2D2
25	G7H8	50	E4D4	75	C6B5	100	B3B4

Table A.1: Belgian Daisy sample game

position where Black can save the marble on C7, but the marble on B6 cannot be saved (see figure A.3).

This game shows that AIBA is able to find good moves and puts pressure on the opponent. It does not play much worse since it ejects 5 opposing marbles. AIBA's biggest problem is that it tries to capture opposing marbles right from the start. This leads to early captures. On the other hand, ABA-PRO does not attack in the beginning of the game, but it tries to reach a good position for the remaining game by occupying the game board's center. As a result it loses some marbles at the beginning, but in the endgame it is very strong and is still able to win the game.

Starting a game at the Standard position does not suffer from this problem since there are no attacking and no capturing moves at the start of the game. To reach a position where the opponent can be attacked the player has to move his marbles to the center anyway (see appendix B).

From this sample game one can conclude that AIBA can be improved by concentrating on reaching a good position at the beginning. It should attack

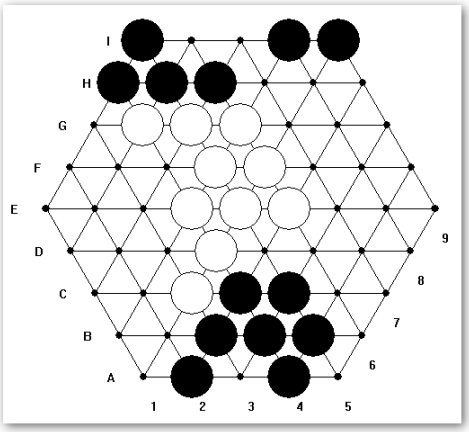

Figure A.1: Position after ply 36

not before it has a strong position at the center with most of the marbles in one single group. A possible approach would be to divide the game into different phases. In the start phase the main goal is to occupy the center leading to an evaluation function that has the highest weight on the features center distance and grouping. Once a good position is reached the weighting changes in the way that ejecting opposing marbles is the most important goal.

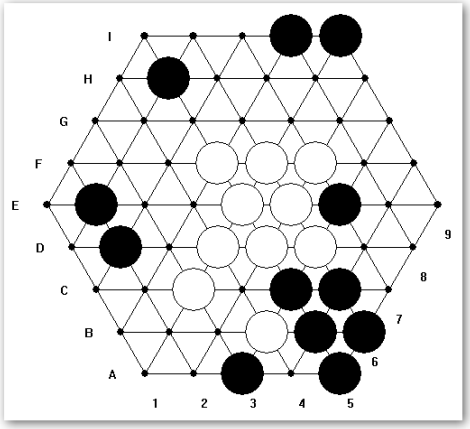

Figure A.2: Position after ply 66

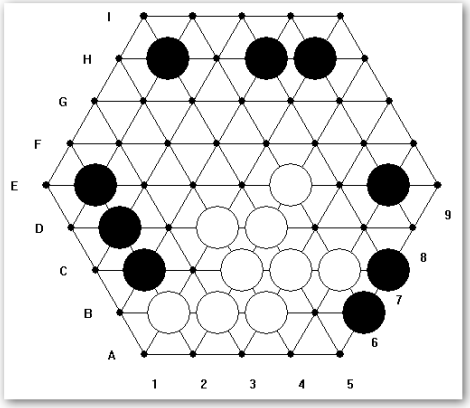

Figure A.3: Position lost for Black after ply 96

Appendix B

Standard Position Sample Game

This appendix gives an example of an Abalone game between AIBA 1200 and ABA-PRO level 7. It starts from the Standard position. In that position ABA-PRO searches much longer to find a move. Therefore, it plays only with a search depth of 7. AIBA 1200 plays the black marbles and starts the game. Table B.1 lists all moves that were made during the game.

Ply	Move	Ply	Move	Ply	Move	Ply	Move
1	A1B2	16	G8F7	31	B3B4	46	H4H5
2	I5H5	17	A4B4	32	I7I6	47	B3B4
3	A2B3	18	F7E6	33	B4B3	48	I8H8
4	I6H6	19	B3C4	34	H9I9	49	B4B3
5	A5A4	20	I8I7	35	B3B4	50	I7I6
6	H8G7	21	B4B3	36	I8I7	51	B3B4
7	B6C6	22	I7I8	37	B4B3	52	G8H8
8	I9H9H8	23	B3B4	38	I9I8	53	B4B3
9	A3B4	24	I6I7	39	B3B4	54	I8I7
10	H4I5	25	B4B3	40	H5H6	55	B3B4
11	A2B2	26	I8I9	41	B4B3	56	I6I5
12	I5I6	27	B3B4	42	I6I5	57	B4B3
13	B1C2	28	H8I8	43	B3B4	58	H5I6
14	I7H6	29	B4B3	44	I5H4	59	B3B4
15	B4C4	30	I9H9	45	B4B3	60	I5H4

Table B.1: Standard position sample game

After ply 60 the game is stopped as a draw, because both players repeat their moves. They keep their marbles in groups so that the opponent cannot attack. This behaviour is as expected. The final position is given in figure B.1. As a result the Belgian Daisy should be preferred for games between the players

to keep the game in progress.

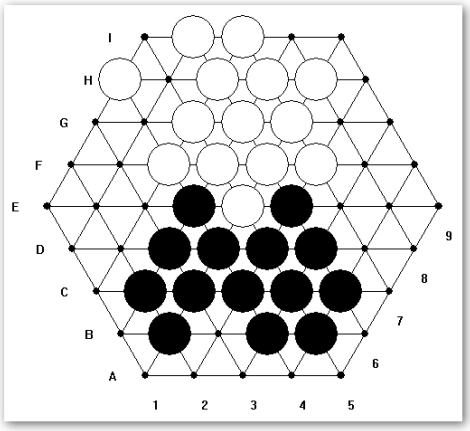

Figure B.1: Position after ply 60