A Knowledge-based Approach of the Game of Amazons

P.P.L.M. Hensgens

Master's Thesis CS 01-09

Thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science of Knowledge Engineering
in the Faculty of General Sciences
of the Universiteit Maastricht

Thesis committee:
Prof.dr. H.J. van den Herik
Dr.ir. J.W.H.M. Uiterwijk
Dr. E.O. Postma
Ir. E.C.D. van der Werf

Universiteit Maastricht
Institute for Knowledge and Agent Technology
Department of Computer Science
Maastricht, The Netherlands
September 2001

Abstract

The game of Amazons is a game with a high complexity. To write an Amazons program that plays well, we have adjusted known game-playing techniques, such as the history heuristic, and investigated many new ideas. Out of these ideas only a few proved to be fruitful. For instance the relative territory evaluator introduced in this thesis, is a new idea, which builds upon the territory function introduced by Kotani [12]. The relative territory evaluation extends the territory evaluation in that it does not only optimise the position of amazons, but of barricades as well. This idea proved to be a valuable asset.

Other techniques such as the history heuristic, the killer moves and the principal variation search did not prove to be beneficial. The critical factor here is the large branching factor, which forces us to use a narrow selective search. Because of the selective search many of these techniques perform badly.

In the end we found a powerful combination of selective search and sorting by the evaluator, which made it possible to play Amazons at a high level. The combination is implemented in the program ANKY, which participated twice in the Computer Olympiad (London 2000, Maastricht 2001). Here ANKY established itself among the worlds best Amazon programs.

A further success of the combination of selective search and sorting by evaluator is that it led us to the idea with which ANKY solved Amazons on the 4×4 and 5×5 boards. It was even possible to compute all the game-theoretical scores for each combination of the first two ply on the 4×4 board.

Finally, it is concluded that the complexity of the game is a decisive factor in finding an optimal combination of techniques to play the game at a high level. The importance of the relative territory evaluator is determined and a few suggestions for future research, such as a game-specific quiescence search, are made.

Samenvatting

Amazons is een spel met een hoge complexiteit. Om een programma te maken dat ondanks deze barriere goed speelt zijn er in deze thesis bekende game-playing technieken, zoals de history heuristic, en aanvullingen hierop onderzocht. Het bleek echter dat maar enkele technieken goed werken, bijvoorbeeld de relative territory evaluator die in deze thesis geïntroduceerd wordt, is een nieuw idée dat voortbouwt op de door Kotani [12] ontworpen territory evaluator. De uitbreiding die de relative territory evaluator vormt op de territory evaluator is dat niet alleen de positie van de amazon geoptimaliseerd wordt, maar ook die van de barricade. Deze uitbreiding bleek erg succesvol te zijn.

Andere technieken zoals de history heuristiek, killer moves en de principal variation search waren minder succesvol. De verklaring hiervoor kan gezocht worden in de hoge vertakkingsgraad van Amazons, die dwingt om sterk selectief te zoeken. Dit is dan ook de oorzaak voor het feit dat deze technieken in Amazons niet het gewenste resultaat hebben.

Uiteindelijk vonden we echter een sterke combinatie van selectief zoeken en ordenen met behulp van de evaluatie funcie die het mogelijk maakte om Amazons op hoog niveau te spelen. Deze combinatie is geïmplementeerd in het programma ANKY, dat al twee keer heeft meegedaan aan de Computer Olympiade (London 2000, Maastricht 2001). Hier werd duidelijk dat ANKY behoort tot ‘s werelds beste Amazon programmas.

Een ander success van de combinatie van selectief zoeken en ordenen met behulp van de evaluator is dat het ons op het idee bracht waarmee Anky uiteindelijk Amazons oploste op een 4×4 en 5×5 speelbord. Het was zelfs mogelijk om alle spel-theoretische oplossingen voor alle combinaties van de eerste twee zetten te berekenen op het 4×4 speelbord.

In de conclusies lichten we de verhouding tussen de complexiteit en de onderzochte technieken toe en bespreken we het belang van de relative territory evaluatie. Vervolgens doen we een aantal sugesties voor vervolg onderzoek, zoals een spel-specifieke quiescence search.

Preface

This report is my M.Sc. thesis in Knowledge Engineering at the Universiteit Maastricht. The subject of the thesis is a two-player zero-sum game with perfect information. The research has been done under supervision of the research institute IKAT (Institute for Knowledge and Agent Technology).

Of course there are many people who have helped me considerably in realising this thesis. I would especially like to thank Prof.dr. Jaap van den Herik as my supervisor for sharing his extensive knowledge with me and dr.ir. Jos Uiterwijk as my daily advisor for giving expert opinions and ideas on the issues involved in this thesis.

Furthermore, I would like to thank both of them for giving me the opportunity to compete in two Computer Olympiads (London 2000, Maastricht 2001). This event has made my fascination for the world of AI games considerably greater than it already was. These Olympiads have inspired me to come up with new ideas for improving my program, which is named after a famous Dutch amazone Anky van Grunsven. The Olympic year 2000 has been a very fruitful year for both Anky's. Anky van Grunsven won a Gold medal at the Olympics in Sydney, while my ANKY won a bronze medal at the Computer Olympiad in London, see [5]. This year at the 6th CMG Computer Olympiad in Maastricht, ANKY participated again and showed that it was still highly competitive.

I would also like to thank dr. David Sturgill, assistant professor at Baylor University (Texas) for allowing me to use his interface for board games and giving me my first 'hands-on' experience with programming AI games.

Then, last but not least, I want to thank Mark Winands M.Sc. who also performed his M.Sc. research in the domain of AI games. Together we always managed to overcome the difficulties in writing the programs and our theses.

Patrick Hensgens
Maastricht, September 2001

Table of Contents

List of Figures

List of Tables

1 Introduction

Games have always been one of humans' favourite ways to spend time, not only because they are fun to play, but also because of their competitive element. This competitive element is the most important impetus for people to excel in the games they play. In the past few centuries, certain games have gained a substantial amount of status, which resulted in even more people endeavouring to become the best in the game. Many games have evolved from just a way to spend some free time to a way of making a reputation. For some people games have even become their profession. This is an evolution, which will probably not stop in the foreseeable future. One of the trends which we have seen in the last few years is that people start to use computers to solve small game problems so that they can understand the problems and learn how to handle them. Computers are thus being used as a tool for people to help them in their quest for excellence.

One of the consequences is that in the long run computers will be able to play a game superior to any human. This has already happened in a number of games, with the most illustrious example of all being chess where DEEP BLUE won against Kasparov in 1997 [16].

In relatively young games such as Amazons, however, both humans and computers are still in a stage of learning from each other. In this stage the humans usually learn faster than computers by the development of implicit strategic knowledge perceived from the games they played. Even though computers can be equally efficient in learning to grasp the tactics of a game, in the case of Amazons they are not and therefore have to fall back on other conventional ways of playing the game. So until the programmers are able to convert the implicit human knowledge into rules which can be used by a computer the humans are expected to be the better players.

1.1 The game of Amazons

The game of Amazons was created by Walter Zamkauskas and later edited as a board game by Ediciones De Mente (Argentina) [2].

Amazons is a two-player zero-sum game with perfect information. The game is played on a 10×10 board and each side has 4 pieces (called Amazons) in the initial position, see figure 1. The white pieces are situated at a4, d1, g1, and j4, and the black pieces are at a7, d10, g10, and j7.

1.1.1 Rules of Amazons

There are only a few simple rules in this game.

1. Although there is no consensus about who starts the game, in this thesis we will assume that White is the player to move first.

2. A move is defined as moving one of the player's pieces to a square reachable by a Queen's move. Then from that square an arrow is shot to another square also reachable by a Queen's move. This square will be a barricade during the rest of the game.

3. An Amazon or arrow cannot land on or move over a square occupied by an amazon or a barricade.

4. The last player who is able to move wins the game.

An additional rule for computing the scores used in tournaments is the following.

5. The score of the winner is calculated as the total number of moves the winner could still make under optimal play after termination of the game. The score is used in case of a tie in the final ranking of a tournament. In such a case the tie is broken in favour of the player with the highest sum of the scores in his won games.

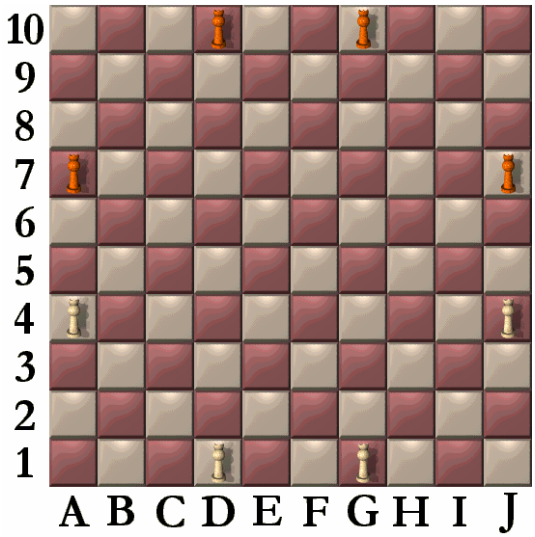


Figure 1. The initial position of Amazons.

The game of Amazons contains certain aspects which can be found in several other games as well. The notion of mobility of pieces for instance is an important factor in this game, as can be deduced from rule 4. The idea of using mobility is one that has been used in several other games like Chess and Checkers. In contrast, Amazons is also a territory game with the subgoal of having control over a territory. This again bears a resemblance to the strategies used in other games like Go.

1.1.2 Notation

For consistency purposes we will adhere to Chess notations when pointing out a position on the playing board. In this notation a1 is the lower left corner, see Figure 1. The notation of a move is slightly different to that of Chess. We will use the notation d1-d7(g7) to illustrate an amazon move from d1 to d7 and a barricade on g7.

To improve readability we also want to distinguish between the game and the pieces. When we use Amazons with a capital letter we refer to the game of Amazons, alternatively we refer to the pieces when we use amazons with a lowercase letter.

1.2 Problem statements

There are two main problems under investigation in this thesis. The first problem statement is:

How can we use knowledge about the game of Amazons in an efficient and effective implementation of an Amazons program?

We will gather some specific knowledge about the game, and use it to implement an evaluation function for playing the game. This evaluator will be compared with other conventional evaluation methods.

The second problem statement is:

How can we apply and adapt certain interesting techniques, developed for computer-game playing, to the game of Amazons?

We will investigate how we can adapt several existing game-playing techniques to fit the game of Amazons. These techniques will be tested for their effectiveness. We will also explore new techniques and ideas, which could be beneficial in playing the game of Amazons.

1.3 Thesis Outline

The contents of this thesis is as follows. Section 1 is an introduction to this thesis containing an overview of the matters under discussion including a presentation of the game of Amazons. This is also the chapter in which the problem statements of this thesis are formulated. In section 2 we will go into detail on the complexity of Amazons. Section 3 covers the techniques investigated and implemented in ANKY, our Amazons program. This section is closely related to section 4 where we will discuss the evaluation function and the knowledge behind it. In section 5 we will show some experimental results considering the techniques and evaluation functions described in previous sections. Its purpose is not only to find the best combination of techniques to play Amazons but also to determine the influence of the individual techniques on the playing strength. Section 6 will present the conclusions and will formulate answers to the problem statements. Moreover some ideas about future research on the subject will be presented.

2 Complexity analysis

The complexity of a game is measured by two different factors, the state-space complexity and the game-tree complexity. Together they provide some information on how difficult the game at hand is. The state-space complexity roughly indicates the number of different board positions that are possible in principle.

The game-tree complexity gives more insight into the decision complexity of the game. It is an indication of how many nodes there are in the solution tree. The number depends on the specific search techniques used. Below we discuss in more details these two measures of complexity.

2.1 State-space complexity

The state-space complexity is an approximation of the number of different positions possible in Amazons. To compute this number we use the following formula:

$$\mathrm{O}(StateSpace) = \binom{P}{W} \times \binom{P-W}{B} \times 2^{P-W-B} \tag{1}$$

In this formula P is the number of squares on the board, W is the number of white pieces and B is the number of black pieces. The formula has three components. The first part denotes the number of ways the white pieces can be put on the board. The second part denotes the analogous number for the black pieces, given there are already W white pieces on the board. Third, remaining squares can be empty or have a barricade on it.

We do not have to sum over different combinations of pieces on the board, because the number of pieces is constant and all pieces of one colour are indistinguishable from each other. The complete formula is as follows.

$$\binom{100}{4} \times \binom{96}{4} \times 2^{92} = 6.45 \times 10^{40} \tag{2}$$

This number has not been corrected for positions that are impossible to occur in the game. Sample positions are given in the figures 2-4. In general, all these positions are derived from the initial position. Each full move the number of barricades will increase by two. This means that by checking the number of amazon moves needed to reach the position and verifying that against the number of barricades, we can detect some illegal positions.

As an example, in figure 2 there are only 2 barricades whereas at least 3 amazons have moved. This means that this position could never occur in Amazons.

It turns out that there are two classes of illegal positions that can easily be characterised.

1. The class of positions where the number of amazon moves cannot be the same as the number of barricades on the board. (For an example, see figure 2.)

2. The class of positions where there are amazons or barricades in positions which they cannot have reached through legal moves. For instance, in the position of figure 3, the implied move g1-g7(e6) is illegal.

A subclass of class 1 is the following.

1a. The class of positions that can only be reached if Black moved first, see figure 4.

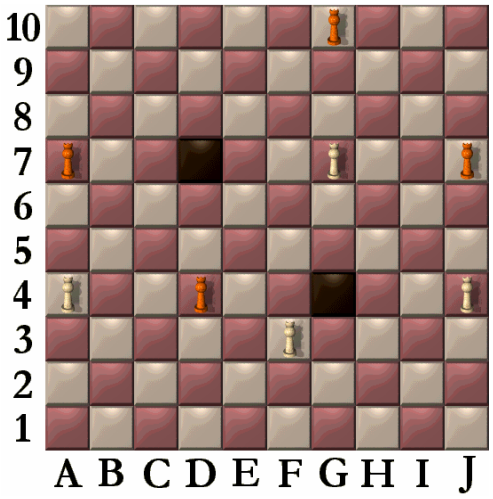


Figure 2. An example of an illegal position. (class 1)

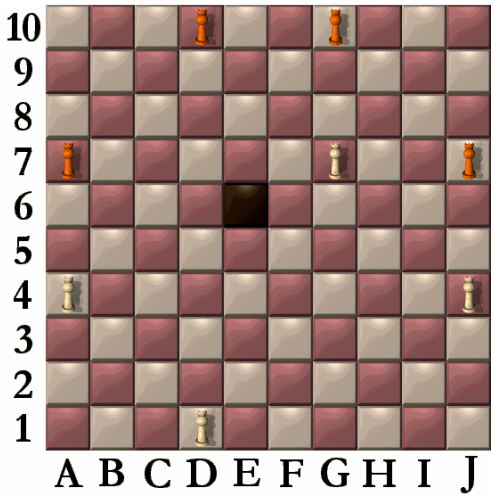


Figure 3. An example of an illegal position. (class 2)

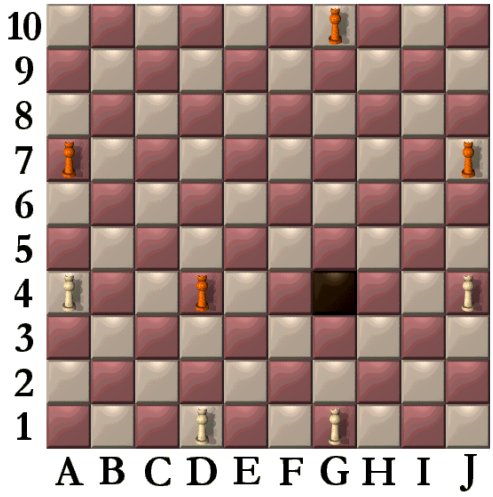


Figure 4. An example of an illegal position. (class 1a)

To see why class 1a is a subclass of class 1 we will just take a different view on the position in figure 4. If we assume that White has made 2 moves (back and forth) and Black only made one move, then we see that this illegal position is an instance of the class 1. Notice that in this assumption we use the notion of the minimum number of amazon moves necessary to reach a specific position.

While it is not hard to find an example of each class, it is much more difficult to give an approximation of the number of illegal positions involved. There is one peculiarity which suggests that most of the instances of these classes only contain a low number of barricades. This is because after a certain number of moves virtually all positions with the same number of barricades are possible. It is only in positions with a low number of barricades where we find illegal positions frequently. Because of this characteristic, we can safely assume that the number of illegal positions is negligible compared with the total number of positions possible. Thus we believe that $O(10^{40})$ is still a good estimate for the state-space complexity of Amazons.

2.2 Game-tree complexity

The game-tree complexity is computed in a way entirely different from the state-space complexity. Here we rely on estimated numbers for the average branching factor and length of the game.

In the initial position the first player has 2176 possible moves. This is a huge number, especially when compared to other AI games, where most games have a branching factor below 50 in the initial position (e.g., Chess 20, Lines of Action 36, Checkers 7, Draughts 9). Fortunately the branching factor in the game of Amazons decreases rapidly as the game progresses. In the endgame the branching factor is usually below 50.

Investigating the average branching factor of Amazons, we encounter a strange phenomenon. From the experiments we performed for finding the average branching factor we observed that this number is quite different for both players. The first player has an average branching factor of 374 while the second player has an average branching factor of 299, (see below). This means that to compute the game-tree complexity we need a formula that takes into account that both players have a different branching factor.

$$O(GameTree) = b_1^{\left\lceil \frac{d}{2} \right\rceil} \times b_2^{\left\lfloor \frac{d}{2} \right\rfloor} \tag{3}$$

b_1 = average branching factor for White
b_2 = average branching factor for Black
d = average number of plies

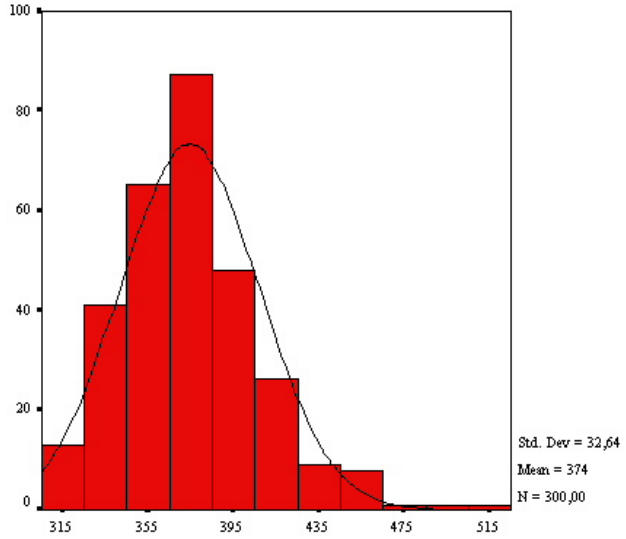
Furthermore, we observed that the average branching factor for both players increases with decreasing playing strength of both players. We suspect that this phenomenon is caused by the fact that when both players are of equal strength and rather strong, the winner does not win by a large score. On top of that, both players use the available space better and will be able to avoid losing squares pointlessly. This has the effect that games usually last longer and thus have a larger game length. Inherently to the larger game length the average branching factor

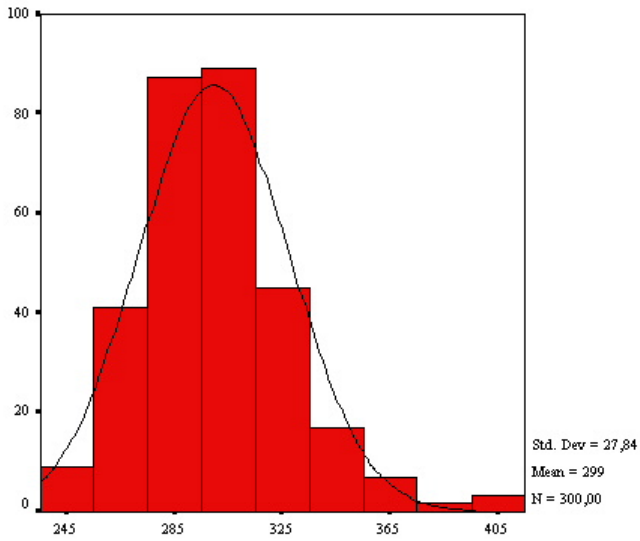
will be lower. For more information on the experiments performed and their results, we refer to section 5.

The average game length we measured during the simulations was 84 plies. When we use these numbers in formula (3) we get:

$$374^{\frac{84}{2}} \times 299^{\frac{84}{2}} = 1.09 \times 10^{212}$$

As is evident from the figures 5 and 6, the average branching factors for both players differ significantly. The numbers in these graphs are based on 300 games where ANKY played against itself with a small random part in the evaluation function and each player using 10 seconds for each move.





There are two factors involved in explaining this difference in average branching factors between both players. First, the rules of the game state that there will be a new barricade each move. This inherently means that the number of possible moves decreases for both players. Since White is the first player he usually has a higher branching factor than Black has in the next move. Second, there is the balance between the strength of the players, which influences their branching factor. If Black would be significantly stronger, early in the game he would be able gain control over larger territories than White. This would mean Black's branching factor would be higher than White's branching factor. We tested this by checking if there is a correlation between the winning player and the average branching factors, and it turns out that the player who wins the game does have a branching factor higher than normal. It is however even then very rare for the second player to have a higher branching factor than the first player. In this test there where only two occurrences of a game where Black had a higher average branching factor than White.

In [11] it was stated that the average branching factor was about 479 and the average game length was 79. We believe that this average branching factor is much too high to represent a usual game, since it is based on experiments using two random players not using any reasonable playing strategy. In particular, the players will not try to restrict their opponent's

freedom of moves efficiently. The lower average game length is explained by the fact that the players do not use their territories optimally, resulting in games terminating prematurely.

To test whether the correlation between branching factor and strength of the players holds, we also performed a similar experiment in which each player only had 500 msec. per move. The results are given in figures 7 and 8. We observe that the average branching factors of both White and Black increased significantly, confirming our conjecture.

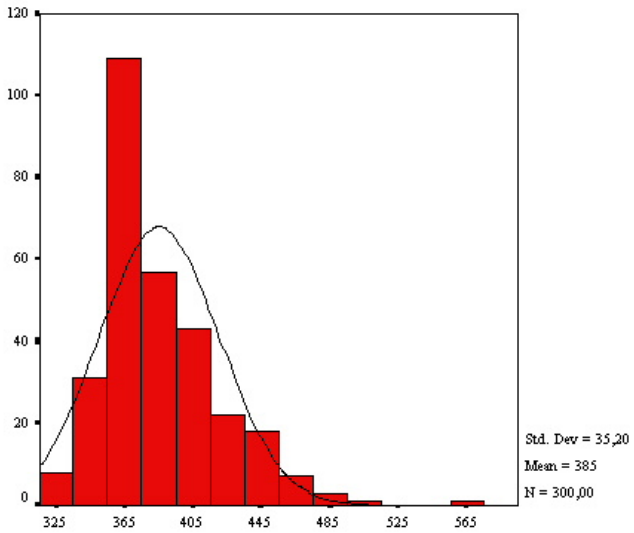


Figure 7. Average branching factor for White. (500 msec. per move)

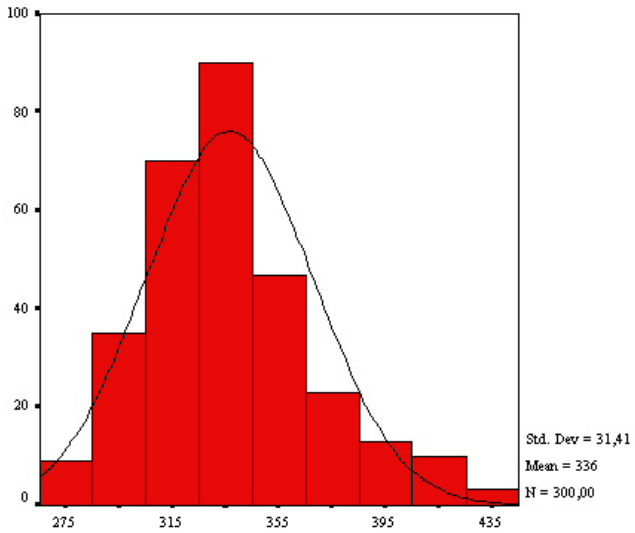


Figure 8. Average branching factor for Black. (500 msec. per move)

2.3 Chapter conclusions

We can safely conclude that Amazons is computationally an extremely complex game, especially since the game tree is huge. In several simulations, it was shown that the rate of play is very important to establish the game-tree complexity reliably. Games between equal players tend to be longer than games between two unequal ones. Furthermore, games between strong opponents also last longer than games between weaker opponents. We also found that the branching factor of each player heavily depends on their strengths. The better the players, the lower their average branching factor. However, in the case where there is one strong and one weak player this is not entirely true. The strong player has a higher branching factor because he is more successful in expanding his territory than his opponent. This means that on average the strong player has more moves to choose from.

A second conclusion we can draw from the results in this chapter is that because of the huge game-tree complexity, a conventional brute-force approach will not work. In the next chapter we will therefore discuss a few search enhancements needed to make search a feasible approach in this game.

3 Searching the tree

In this chapter we introduce the search algorithms and heuristics tested, aiming to answer the second problem statement. The search implementation and the evaluation are the main fields of research for game playing. These two parts of the program can be compared to the heart and the brain in the human body. No human can do without either one of these, just like no game-playing program can do without them. This is also the reason why chapters 3 and 4 are tied together very closely and therefore cannot be read separately. However, before we dive into details concerning the search, we will describe how we build the tree which we have to search.

3.1 Building the tree

To search a game tree efficiently, our principal algorithm is the well-known alpha-beta algorithm, to be discussed below. Before we can find the best move using alpha-beta, we have to generate all the moves. This is done by the move-generation code in ANKY. It is a straightforward procedure of checking which moves are valid for the side to move. All the possible moves are enumerated in a list. This takes a bit of time, but still the amount of time that is used to generate moves during search is less than 5% of the total search time. We therefore believe there is not much to gain in trying more efficient techniques like incremental move-generation or move-generation using pre-filled databases.

3.1.1 Alpha-beta

In the game of Amazons building the search tree is fairly straightforward. The root node corresponds to the current position, while every possible move in this position leads to a child node. The tree is constructed by recursively generating all the children of each position. Of course, we need a stopping criterion. The straightforward one is when we encounter a terminal position, which means that one of the players has won the game. If such a position is reached, we will not generate children but backtrack and generate the remainder of the tree. A second stopping criterion is a maximum depth, set before the search started. If we reach this depth we will not generate any children but instead go on generating the remainder of the tree to this depth.

As can be deduced from the former paragraph we use a depth-first algorithm to traverse the game tree, a depth-first iterative-deepening algorithm to be more exact. The algorithm of our choice is alpha-beta. It has been conceived by McCarthy (1956), but the first person to use it in an actual program was Samuel [14]. When first proposed it was denoted a heuristic, since it was not clear whether it guarantees the best possible result. A few years later, when Knuth and Moore (1975) [11] proved the correctness of alpha-beta, it was no longer seen as a heuristic, but promoted to the status of an algorithm.

By using the alpha-beta algorithm, we implicitly assume that both players play optimally according to the evaluation function in use. The idea behind this is that if the opponent does not play the best move, the current player will be able to achieve more profit in terms of evaluation scores. Because of this, it is of the utmost importance that a position is evaluated as accurately as possible.

The success of alpha-beta in game playing is mainly achieved by the cut-offs it enables. It effectively prunes away large uninteresting chunks of the game tree. These cut-offs are realised by the alpha and beta bounds, where alpha is a lower and beta is an upper bound on

the evaluation score. The reduction of the game tree can be as high as 99% when the moves are perfectly ordered. For a detailed description of the alpha-beta algorithm we refer to [13].

3.1.2 Iterative deepening

Iterative deepening is a method of consecutively searching to increasing depths 1...*N*. The evaluations in the previous search depth can be used to sort the moves of the next search. The overhead when using this method is minimal since the game-tree complexity is $O(b^d)$, where *b* is the branching factor and *d* is the depth. This means that in each search there are always more leaf nodes than there are internal ones. So the majority of the nodes encountered in each search are leaf nodes. Therefore the number of nodes examined during a search process is only a fraction of the nodes to be examined in the next iteration. Moreover, using results from previous iterations can enhance the move-ordering and thus the efficiency. The killer move, history heuristic and transposition tables are examples of techniques which use the concept of iterative deepening to optimise the search. They do this by sorting the moves from best to worst using the evaluations found in the previous search. As an effect, node counts with iterative deepening can be even less than searching to the same depth without iterative deepening.

3.1.3 Selective search

Selective search reduces the effort put in seemingly unpromising moves. It selects only the best moves available for further examination in the alpha-beta algorithm. The problem here is to have a decent selection criterion. Since it is possible that we cut off the best move with this selectiveness, we want to be sure that the chance of this happening is quite small.

3.2 Sorting the moves

By sorting the moves generated in the alpha-beta search we can maximize the number of cut-offs in the algorithm. The idea behind this is that if the best move is evaluated first, then the alpha and beta bounds will leave only a small window in which moves are not pruned. In effect every move which violates these bounds will be pruned. The problem is that we do not know which move is the best at this point because that is why we are searching in the first place. Several heuristics have been developed to solve this problem. We will discus a few of these in the next sections.

3.2.1 Killer moves

The killer-move heuristic is one of the most efficient ordering heuristics. It was introduced by Huberman in 1968 and has proved to be quite successful and reliable in all sorts of games. The logic behind this heuristic is based on the assumption that the best move for a given position could well be a very good or even the best move in another position encountered at the same depth during the search process. For an example we look at figure 9 and see that h6-e6(c4) is the best move because it takes control of the upper left area. In figure 10 we see a similar situation, with the exception that White played its last move, say h3-g3(g4) differently. In picture 10 White played h3-g3(h3). Still move h6-e6(c4) is the best move here since we still have the same advantage which caused us to choose this move in figure 9. For more information on the killer-move heuristic we refer to [1].

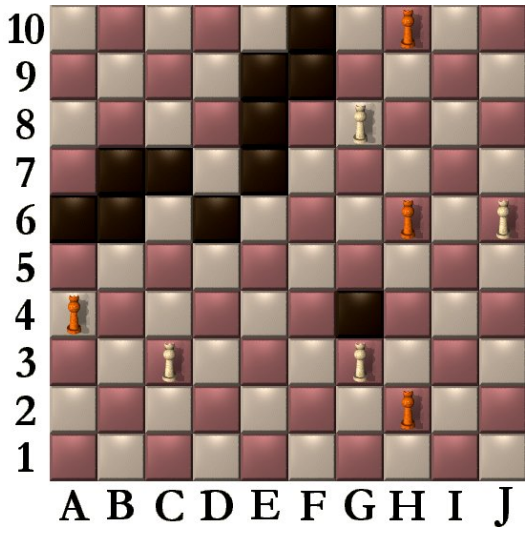


Figure 9. An example position (Black to move).

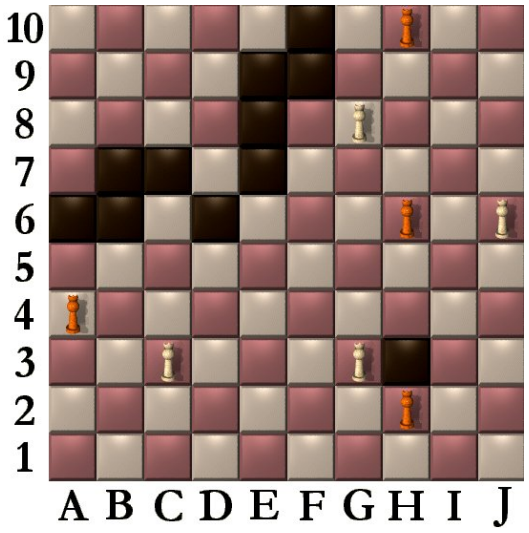


Figure 10. A similar position (Black to move).

3.2.2 History heuristic

Despite its simplicity, this heuristic is still capable of ordering moves quite well. The working of this heuristic is in some way similar to that of the killer moves. Every time a move is selected as best move it gets awarded a score of 1. We keep track of all the scores of every possible amazon move on the board. This is done by making a 10×10×10×10 array, thus having a counter representing each possible amazon move. Each counter is a 4-byte integer, which means that in total the memory needed for the history heuristic is 40 kilobytes per player.

To order the moves we have to look up the counter representing this move in the table. This is easy because all the counters are indexed according to the 'from' and the 'to' position which is known at that point.

The logic behind this method is that a move, often preferred over other moves, has a high likelihood of causing a large number of cut-offs. Secondly, a move that is good at this depth might very well also be good at other depths. A well-known problem that we stumble upon here is that the history table keeps building up and what appear to be good moves now can still be overshadowed by the good moves earlier in the game. To counterbalance this we divide all the counters by 4 at the start of a new search process.

When we want to use the history heuristic in the game of Amazons we need to make some adjustments because of the structure of a move. The queen move and the barricade move can be seen as two separate consecutive moves. These two parts are not entirely dependent on each other because the same barricade could be put by another amazon move. In order to capture this independence we used an extra array for the barricades. Because origin is not relevant for a barricade we only need 10x10 counters (0.4 kilobyte) to implement them in the history heuristic. When ordering the moves we sum the counters for the amazon move and the barricade move and sort the moves on these values. Another reason to do this is memory efficiency. A 10×10×10×10×10×10 array of 4 byte integers would amount to a 4 megabyte history table for each player. For more information about the history heuristic we refer to [17].

3.2.3 Transposition tables

Transposition tables hold knowledge about already encountered positions. These positions are saved together with valuable information like the value of the position and a flag, which states if the value is exact or an upper or lower bound. When we encounter one of these positions again somewhere else in our search, we can use this information to narrow down the rest of the search. This heuristic has proven to be very successful, for example in Chess where the search tree can be reduced by 50%. The transposition table is mostly implemented as a hash table using the well-known Zobrist hashing method [21] with a replacement scheme to handle collisions [5].

Although we did not include transposition tables in our experiments, there are some interesting thoughts about them related to their importance in playing the game of Amazons and especially in the program ANKY. In Amazons, the occurrence of transpositions is far less than in Chess since a position with a specific number of barricades can only occur at that specific depth in the game. This effectively means that a transposition can only occur at the same depth. Furthermore, transposition tables start to pay off when the search reaches depths of at least 4 ply. In a brute-force Amazons program this is virtually impossible due to the huge complexity. Most of the performance gain when using a transposition table is a result of the ordering of the moves according to the table. Since the use of transposition tables in ordering the moves is limited in that it can only order moves which have been encountered before, it will be useless at the beginning of the search. We therefore decided not to implement transposition tables for the moment, but gave priority to other enhancements.

3.2.4 Static board scores

Another way of ordering the moves is to compute scores for each square on the board. This should give an indication of the value of an amazon or barricade at that position. The advantage of this method is that it is reasonably fast since the board scores only have to be computed once, before the search starts. Every move encountered from there on will be ordered according to these scores. To do this we take the score of the destination square and subtract the score for the source square of the amazon. Second we add to this the score for the barricade made by this move.

However, there is also a drawback of this heuristic. The accuracy of the scores on the board deteriorates the deeper we search, since all the scores are computed for the position at the root. The deeper we search the more discrepancies we will have compared to the position at the root of our search tree. The effect of this is countered by the fact that the deeper we search the less important the exact scores are, since the largest benefits of move ordering occur high in the search tree.

3.3 Other enhancements

In the next three sections a few other enhancements, which we investigated briefly, are discussed.

3.3.1 Principal variation search

PVS, shorthand for Principal Variation Search, is a windowing technique with the goal of minimizing the effort involved in classifying good and bad moves. To achieve this the initial alpha-beta window will be limited to a minimal window with beta = alpha+1. These bounds only hold for the principal variation. In the case where we find a value which appears to be better than the current value and thus outside the window we will have to widen the window

and search again in order to find the exact value of this move. The reason for this re-search is that values outside of the window are inaccurate due to cut-offs based on the current window.

The initial value of alpha is found by searching the first, assumed to be the best, move in the list and initialising the alpha value according to that search. If at some point we find that the value of another move is below the alpha bound we return a fail low. In the similar case where we find a value higher than the beta bound we consequently return a fail high.

The profit here is that, in general we examine considerably fewer nodes less when using PVS than when using plain alpha-beta with an open window. For more information on this technique, see [13].

3.3.2 Opening book

The use of an opening book to store the best moves for several opening sequences, is a technique which is commonly used in game playing. The advantage here is twofold. First one of time: for a position in the opening book no search has to be performed, but the best move is instantaneously played from the book. Second, the moves from a book are mostly based on large (human) experience and their quality therefore often exceeds that of the moves determined by search. In Amazons however there is not yet sufficient knowledge about openings to get a definite advantage from using an opening book. As an alternative we could construct an opening book by searching positions using a large time limit. However, since this method is very time-consuming and we do not expect a large benefit from it, we have refrained from using an opening book altogether.

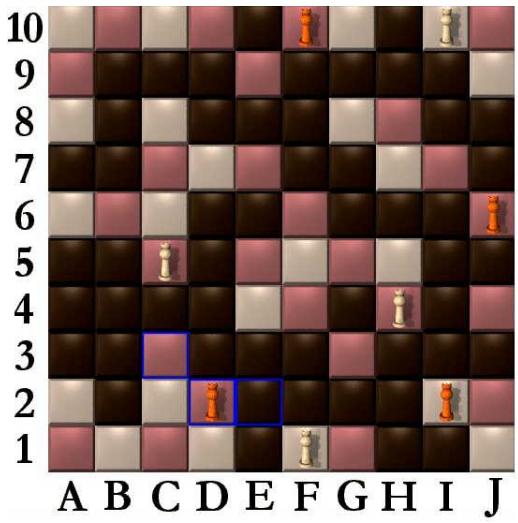


Figure 11. Example of an endgame position.

3.3.3 Search in the endgame

We define the endgame in Amazons as the part of the game in which there are no areas, which contain amazons of both sides. We call these areas absolute since the value of these areas cannot change anymore; they either belong to a side or belong to none. For a position where this is the case, see Figure 11. This stage of the game is irreversible in that there are no legal moves, which cause the game to go back to the middle game. Even though we still have

two players and use the same rules, the basics of searching change in this stage. We no longer need minimax since every move by a player is independent of any move by the opponent. Alpha-beta is in this stage an inefficient algorithm to use; a single-player search algorithm would be much better. Although we can increase our search depth if we only search for one side, it is still not trivial to find the right move to fill an area optimally. Buro [6] discovered that the problem of finding the optimal moves to fill an area is equivalent to finding a Hamilton circuit in a graph, which is an NP-complete problem.

Endgame databases can help in minimizing the chance of creating an area which cannot be filled optimally. Thegos from the University of Alberta suggested an endgame database for dealing with this problem [18]. The criterion for the database is not the usual number of pieces on the board, but the size and shape of the area. He has constructed databases for certain areas with one or more amazons in it, including some areas where both players are present. This database is used in his program ANTIOPE.

4 Evaluation of positions

Even the fastest search will not find anything in a complex game like Amazons if it is not guided into the right direction. The ultimate goal of the search is to find a forced win for the player to move. However, since this is usually impossible to achieve, we will have to rely on an estimation on what is the most promising move in the current position. This estimation is based on an evaluator. The evaluator maps all possible positions to a value. The move which leads to a position with value *X*, will be given the same value *X*. The alpha-beta algorithm will then take care of propagating the values from the leaf nodes up to the root.

Positions good for White are inherently bad for Black since Black does not want White to gain any profit. This is represented by the fact that higher values are good for White and lower values are good for Black.

4.1 Evaluation functions

There are several evaluators that are capable of guiding the search, see [12], [7] and [19]. The most elementary one is the random evaluator. This evaluator will be dealt with in subsection 4.2.

As explained in section 1 at the introduction of the game, there are two major properties of the game of Amazons. The first one is the mobility component in the game. The mobility is represented by the number of possible moves a player can make in the position at hand. The reason for its usefulness stems from the rules of the game, especially the winning rule. The player who made the last legal move wins. The more moves a player has, the less likely he is to get one of his amazons locked up in a small area. A second property is the notion of territory. A player can consider all the squares, which he can occupy with less moves than the opponent, to his territory. The idea behind this is that if a player controls more squares than his opponent, he will eventually have more space to move and thus win the game. A mobility and two territory evaluators will be considered in sections 4.3 – 4.5. An example position is considered in section 4.6. Section 4.7 gives some preliminary conclusions.

4.2 Random Evaluation

Although random evaluation is far from optimal, it does allow us to investigate some properties of the game. As reported by Beal and Smith [4], random evaluations yield some results characterising alpha-beta search. Furthermore, it enables us to abstract some information on how important mobility is in the game of Amazons.

The random evaluator is quite easy to implement. We only have to generate a random number each time a position is evaluated. The consequences of this are at first sight surprising. We note that though the evaluator is completely random, the chosen move is not! It follows that this evaluator prefers moves that lead to a position with larger mobility. This can be explained by using probabilities. Let us say the White player wants to maximize the evaluation function. He will therefore prefer the move with the highest evaluation score. Suppose he has two choices to make a move and one leads to a board where he has 90 moves and one leads to a board where he has only 10 moves. All these moves will get a random score. However there is a 90% chance that the highest random number will be assigned to one of the 90 moves and only 10% to one of the 10 moves. The alpha-beta algorithm will propagate this value up through the tree. This causes the program to choose the move leading

to the higher mobility with a larger probability, which indirectly introduces a bias to subtrees where the player has more legal moves to choose from.

4.3 Mobility Evaluation

The mobility evaluator [19] computes the number of moves for both sides and then subtracts the number of Black's moves from the number of White's moves and returns this value. The computation of these values is easy and fairly fast compared to territory evaluation.

There might be other enhancements to the mobility evaluator. However the main advantage of this evaluator is its speed. It can evaluate more nodes per second than the territory evaluators, to be discussed below, and still give a good estimate. In order to preserve this advantage, it is not advisable to add too many enhancements.

4.4 Territory Evaluation

If we want to add more knowledge to the evaluator, we need to look at more properties of the position at hand to discover how well each player is doing and who is winning the game. The territory evaluator first introduced under the name Minumum Stone Ply Function in 1999 by Kotani [12] does just that. The score for a position is based on the territories which belong to either side. The exact territory is determined by finding the least number of moves necessary for both players to occupy a square on the board with one of their amazons. All the squares White can reach faster will be in White's territory. The ones Black can reach faster will be in Black's territory. Of course, there are also squares which can be reached in the same number of moves by both players. These squares do not contribute to the score of either player.

The score for each player is defined by the territory he controls. Each square in this territory adds a certain credit to the players score. This credit can be a constant as suggested in [12], which would mean that the score is effectively defined by the number of squares. A more informative value would be to take into account how many moves it would take for this player to actually get to this square. This measure adds a sense of certainty to the area. Areas which can be reached fast, are easier to defend and thus should be rewarded extra. The latter approach has been adopted by us.

4.4.1 Implementation

The implementation can be described as a bi-directional search for the owner of a square. There are a number of steps which are taken to evaluate each position.

Step 1: Mark all squares which can be reached directly by the original amazons of each side by *Y* which denotes the number of moves necessary to reach this square, *Y* is initialised to 1. A square that is marked for both sides will be denoted neutral.

Step 2: For each unmarked square, search if there are marked squares to be reached in one move. If so mark this square by the marker found and add one to it (*Y+1*). Again if markers for both sides are found the square under consideration will be denoted neutral.

Step 3: Use all newly-marked squares found in step 2 except the neutral ones and mark all the squares they can reach in the same way as described in step 1. Keep repeating step 2 and 3 until either of them cannot put any more markers on the board.

The advantage we have when we mark for both sides each step is that we only have to mark each square once, or twice in the case of a neutral square.

4.5 Relative Territory Evaluation

The idea behind relative territory evaluation is essentially the same as the former territory evaluation function. The difference here is that the value of a square is not only dependent on how fast the owner can reach it, but also how fast the opponent can reach it. The value of all the squares in a territory is in this evaluator completely dependent on the difference of number of moves necessary for both players to put an amazon on the square. The higher the difference, the easier it is for the owner to assure that this square will be his. In essence this means that the evaluator will not only optimise the number of moves necessary to occupy a square, but will also try to maximise the number of moves necessary for the opponent to reach this square. In order to achieve this the barricades are used. The relative territory evaluator has thus more information about the barricade position than the territory evaluator has.

4.5.1 Implementation

There is one big difference in the implementation of this evaluator compared to the former one. In this evaluator we first complete all the steps described in section 4.4.1 for White and thus mark all the empty squares on the board, after which we mark the board for Black. Each square will have two markers that refer to the number of moves necessary for each side to reach it. The value of a square is determined by the difference in these two markers and the sign of the value is determined by checking which marker is higher.

Computationally this procedure is more time consuming than the one described in section 4.4.1, because here we need to know how fast each player can reach a square, while the territory evaluation only computes how fast the fastest player can get there and says nothing about the opponent.

4.6 An Example

As an example to show the difference between the territory and relative territory evaluation we reuse the position of figure 9 in figure 12.

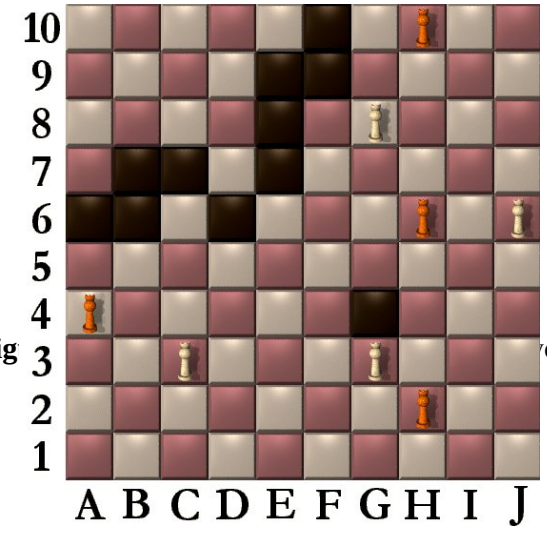


Fig e).

In this position Black is to move and will choose h6-e6(c4) when using the relative territory evaluator. The reasons for this move are simple. Moving to e6 will provide Black with access to the upper left area while a barricade on c4 will secure this area so that Black does not only occupy the area first but also optimizes the difference between the number of moves needed for Black and White to get there.

The territory evaluator looks at this position in a slightly different way and prefers h6-e6(e1). Even though this is an inferior move, the territory evaluator is not equipped to determine this. From its point of view both moves provide access to the upper left area. The move with the barricade on c4 does not look as appealing as it should to the territory evaluator. If Black puts the barricade on c4, White controls two additional squares d4 and f4 (since these squares are blocked for Black by the c4 barricade), while it only loses c5. Putting a barricade on e1 will not provide White with any advantage like that. Only in a deeper search will this evaluator cause the search to find that the barricade on c4 is superior to the one on e1.

4.7 Chapter Conclusions

In this chapter, we described 4 evaluators. The random evaluator is only used for benchmarking purposes since its playing strength is far worse than the other evaluators.

The idea of using mobility and territory evaluators is not new. They have been used and described before, see [7] and [19]. The relative territory evaluator adds something new to an old idea. The effect of this enhancement will be investigated in the next chapter.

5 Experiments and Results

This section provides more information on the experiments performed and their results. We mainly focus on the 10×10 board, which is also the size most commonly used in human play. Although we also investigate different sizes, we do not treat the 14x14 board size. This may seem weird since this size is also known for its use in human play, but for computer play this size is not very interesting in that it does not add something new to the game, but just increases the branching factors.

Sections 5.1 to 5.3 describe the experiments on the standard board, investigating game characteristics, search enhancements and evaluation functions, respectively. In section 5.4 we investigate non-standard board sizes. Finally, section 5.5 provides some conclusions.

5.1 Game Characteristics

In this section we detail some of the experiments determining game characteristics, of which the main results have already been mentioned in sections 2 and 4. The characteristics are, branching factor (5.1.1) and game length (5.1.2). In section 5.1.3 we discuss random evaluations

5.1.1 Branching factors

Several experiments regarding the average branching factors have been performed. For the experiments in Table 1 we used the relative territory evaluator in a plain alpha-beta setting with a selective search of 14 moves. The table shows the relation between the strength of the opponents and their branching factors. For these experiments we used a fast (Athlon 700Mhz) and a slower (PII 233 Mhz) computer. The difference in strength of both players is manipulated by the amount of time they can use to select their move.

PC	time per move		no. of wins		avg. branching factor	
	White	Black	White	Black	White	Black
P II 233Mhz	500 msec	10000 msec	23	127	375.9	345.5
	500 msec	500 msec	78	72	380.9	327.3
	10000 msec	10000 msec	89	61	373.2	299.1
Athlon 700Mhz	500 msec	10000 msec	15	135	398.4	350.1
	500 msec	500 msec	165	135	385.0	336.3
	10000 msec	10000 msec	160	140	374.5	299.4

Table 1. Average branching factor for both players at different playing strengths.

We can clearly see significant differences in the branching factors of the players. The first player, White, appears to have a significantly higher average branching factor than Black. A second observation which can be made is that there is a correlation between the strength of the players and their average branching factors. The exact same players both had a higher branching factor when we decreased their search time by a factor of 20 and thus decreasing their strength. This might be because both players will choose their moves according to evaluations found at low search depths, which decreases their ability to find deficiencies in their territories and cause them to stay in local maxima of the search space. Especially in the

endgame this can have grave consequences since the players will lose squares as a result of this drawback. It is here where they lose significant parts of the board, thus limiting their number of plies with low branching factors. Because of this short endgame their average branching factor will increase.

In the case where one of the players is stronger than the other we note that the strong player will take control over the board faster and thus increasing his average branching factor.

5.1.2 Game length

The game length of Amazons is closely related to the playing strength of both players. We did several tests to get more insight into this relation. Table 2 shows the average game length for different playing strengths. To change the playing strength we only adjusted the search time and left all other values unchanged. For this test we used the relative territory evaluator.

PC	Time White	Time Black	Avg. length
P II 233Mhz	10000 msec	10000 msec	83.8
	500 msec	500 msec	82.3
	500 msec	10000 msec	79.8
Athlon 700Mhz	10000 msec	10000 msec	83.9
	500 msec	500 msec	83.8
	500 msec	10000 msec	80.7

Table 2. Average game length at different playing strengths.

As can be derived from table 2 the game length is higher when both players are equally strong (independent of their absolute strength) as opposed to one player being somewhat stronger. This is easy to explain, since if one of both players outperforms the other one, the stronger player will in general have his opponent locked up earlier.

5.1.3 Random Evaluations

In order to assess how important mobility is in Amazons we did some tests with the random evaluator. For more information about these tests we refer to [4]. For people unfamiliar with the article we briefly describe the tests.

We used two different evaluators, one random evaluator as described in 4.2 and a lookahead-zero evaluator. The lookahead-zero evaluator evaluated every position to 0 except for terminal positions which were evaluated to the appropriate value exactly the same as the Random evaluator did. The lookahead-zero evaluator chose a move completely at random unless terminal positions where found during the search, while the random evaluator chose according to the random values propagated back up the tree by the alpha-beta algorithm.

Table 3 shows how both evaluators perform when playing against each other at different constant search depths.

R vs LZ	No. Games	R wins	LZ wins	R %	Av. Length
Depth 1	400	217	183	54.25%	70.7
Depth 2	400	290	110	72.5%	67.8
Depth 3	30	30	0	100%	65.7

Table 3. Random (R) against lookahead zero (LZ) at different search depths.

Here we see that the deeper both players search the bigger the advantage is for the random evaluation. The results also present some evidence for the relation found between game length and playing strength. This results in lower game lengths when the difference in strength between both players increases. The main conclusion we can draw looking at these results is that mobility is definitely an important part of evaluating positions in Amazons.

5.2 Search Enhancements

The search enhancements we tested have already proven their value in the world of game playing. However, it is clear that they do not always improve the playing strength. In the following four sections we look at the Killer-move heuristic, the history heuristic, the static board scores and principal variation search and investigate which enhancements are beneficial and which are not.

5.2.1 Killer-move heuristic

The killer-move heuristic as described in section 3.2.2. was one of the first enhancements we tested. Even though the test results were surprising at first, they became more logical when we took a closer look at all the test results.

In this experiment we use the territory evaluator with a maximal branching factor of 14 and a thinking time of 10 seconds per move. These 10 seconds are used as a guideline in deciding if the search should start another iteration in the iterative-deepening process. This is done if the square root of the branching factor times the time already used is lower or equal to the remaining search time. If this is true then it is likely that searching one ply deeper would still be feasible within the remaining time.

Killers		Wins		Nodes evaluated		Time used	
White	Black	White	Black	White	Black	White	Black
yes	no	56	94	59580701	60321954	41378300	38855730
no	yes	73	77	60524718	62439799	40547700	39013020
no	no	72	78	60680274	61041507	41052435	39012165

Table 4. The performance of killer moves in Amazons.

Looking at this table we see that the players who use killer moves lose more games than the players who do not use them. From these results we would even argue that using killer moves decreases the playing strength. If we look a bit further we can see that both players use approximately the same time and examine the same number of nodes. We can deduce from this that the killer-move heuristic causes the search to examine less fruitful lines of play. This is exactly what we want to prevent when using killer moves. An explanation for this surprising result might be that killer moves do not work in combination with a drastic selective search, especially if the ordering on killer move is done before the selective search, so the killer move is always selected, even if it did not seem good enough to be selected according to the evaluation.

The reason why killed moves do not perform good enough to be used can be explained as follows. Due to widespread influence of an amazon even the smallest change on the board often influences the entire board and the best move is very likely to be changed accordingly. We therefore take a big risk when basing our decision for taking a move as a killer move on

the narrow basis, which we have when using a selective search of the magnitude we are using here. One small change might for instance cause hundreds of moves to be better than the killer move.

Even though alpha-beta is not an efficient way of searching in the endgame, we tested our hypothesis by checking if the killer move heuristic would perform better in the endgame where branching factors are usually low. We found that in the endgame, killer moves become more and more profitable. This can be explained by our hypothesis because, the risk of using a killer move that is inferior to the other selected moves is drastically diminished in the endgame. If for instance we examine 14 possible moves for each position with about 20 legal moves, we have only a low risk of not examining the best move and thus failing to minimize the search window effectively. Furthermore, even if the killer move is not the best move in this position it is still likely that it is better than some of the 14 moves selected.

5.2.2 History Heuristic

The history heuristic is a very efficient way of sorting nodes according to their history of success [17]. In ANKY however, the heuristic did not nearly perform as good as sorting according to the evaluator. The cause of this bad performance can again be brought back to the use of a severe selective search. First of all we want to be sure that the few selected moves contain the best move. If we are not sure that we have a very big chance of selecting the best move than we should increase the number of selected moves. This is where the history heuristic comes into play. The history heuristic is a very efficient, but crude way of sorting nodes. This means that it is very fast and provides a good ordering. However, it is not good enough to ensure that the best move is among the selected five percent of the total moves. Second if we use history heuristic and selective search we introduce an unwanted dependency. This dependency causes a downward tendency to the performance of the history heuristic. It is obvious that only the nodes, which are selected for that depth, can receive history-heuristic points. Because we do not even look at all the other possible moves they will rarely be able to get enough points so that they have a good history of success. This might cause these moves to be underrated in the next history-heuristic sorting which is the cause for the fact that these nodes have a bad chance of being selected by the selective search. Here the circle is complete because, if they are not selected, they will not receive history-heuristic bonuses. If we would not use a selective search, each node would have an equal chance of receiving history-heuristic points according to their applicability in the situation at hand. This would be better since there is no unwanted bias to moves that proved to be good in the former search.

In table 5 we display results of experiments done with history heuristic and killer moves, these experiments where all done using the territory evaluator. If no history heuristic is used, the moves are sorted using the evaluator. The table shows that the history heuristic does not work when using selective search as it lost all 100 games played in that setting. Even at full width search the heuristic lost 60% of its games.

Selective search		History heuristic		Killer move		Wins	
White	Black	White	Black	White	Black	White	Black
14	full	No	Yes	No	Yes	61	39
full	14	Yes	No	Yes	No	41	59
14	14	No	Yes	No	No	50	0
14	14	Yes	No	No	No	0	50

Table 5. Experiments with history heuristic and killer move.

5.2.3 Static Board Scores

The use of static board scores has a big advantage: it only has to be created once each move. This is especially interesting when we compare it to sorting by evaluator. If we sort by static board scores we only need to put a score on each square except the squares which are already barricaded. However, since we have two players we will need to do this for both players. Additionally we need to compute the value of a barricade on each square. Since a barricade is the same for both players we will not have to do it separately for each of them. This all adds up to a maximum of 300 evaluations.

The drawback is that already at the second depth of search most of these scores are bad approximations of the real values of positions. The structure of a move already hints at the fact that a single move can change the entire board. A move can go from one side of the board to another side and will in most cases have a certain impact on all 8 amazons. Therefore the influence and thus the value of each amazon changes each time a move is made. This is exactly the weak point of the static board scores.

Its efficiency in fast evaluation is no compensation for the low quality of the evaluation and therefore we refrained from its use.

5.2.4 PVS

The principal variation search is especially useful in games with bushy trees. Amazons is such a game, with its branching factor ranging up to 2200. The PVS technique can be used to determine that subtrees are inferior without knowing their exact value. This means that we will have more time to spend in more fruitful subtrees.

In ANKY however we found that the presumably best move on which the PVS window is based, is often not the best move. Therefore the window has to be adjusted quite a few times and this means we need to re-search parts of the tree which takes away the benefit of using PVS. In positions with an obvious expected line of play, the PVS can still cause a drastic reduction of the search tree. Unfortunately these positions are only exceptions in Amazons. The lack of understanding of the game and therefore the slightly rudimentary evaluations do not provide us with the quality of evaluation we would need to use PVS efficiently considering the large branching factor. Another reason for the disappointing results is that we are using a selective search, which greatly reduces the effective branching factor for the PVS. This also means that there are usually no nodes significantly inferior to the rest which causes the PVS to re-search too many times.

PVS		wins		avg. game	nodes		time used	
		White	Black	length	White	Black	White	Black
Yes	No	23	27	84	22407155	19299247	14138090	12943940
No	Yes	20	30	85.5	20060323	24398888	13590950	13310250

Table 6. Experiments using the territory evaluator with PVS.

In table 6 we see that the PVS-enhanced player wins 53 out of 100. This is not a convincing score. Moreover we can see that this player evaluated more nodes than his opponent, something which should not happen in a successful use of the PVS technique. Therefore we are not convinced that PVS works in Amazons. Further research should be done as to the use of PVS in the endgame since we believe that there will be fewer re-searches in this phase because of the fewer possibilities and the better estimates of the value of a move.

5.3 Evaluator tests

There are two interesting evaluators outlined in this thesis. We will evaluate their performance compared to each other. However, since each evaluator has its own strong and weak points we needed to adjust the settings for some other parameters so as to make a fair comparison between the various evaluators.

5.3.1 Selective Search

The selective search is one of the main features we adjusted to make a fair comparison possible between the evaluators. The mobility evaluator for instance, is somewhat faster than the territory evaluators, but is a worse estimator than the other evaluators. For best results we used a less narrow, selective search and thus increased the number of nodes examined for each position.

The main goal was to have the strongest configuration of each evaluator play against the others. Only in this way we could determine the true strength of the evaluators.

5.3.2 Territory versus Relative Territory

We tested the territory and relative territory evaluators against each other. In order to find the best settings for each evaluator we had them play themselves. This might of course not be the best setting to play against the other evaluator but it does provide us with a simple solution to the problem of finding the best evaluator. Since we already said that killer moves, history heuristic and PVS do not provide a noticeable improvement we did not use those techniques in this test. We only changed the selectivity factor to find the best playing evaluator using a 10-seconds search time for both evaluators.

Evaluators		Selectivity factor		Wins	
White	Black	White	Black	White	Black
Territory	Rel. territory	14	22	94	56
Rel. territory	Territory	22	14	62	88
Territory	Rel. territory	14	14	43	57
Rel. territory	Territory	14	14	24	76

Table 7. Territory against Relative Territory.

As we can see in table 7, the territory evaluator beats the relative territory evaluator by 182 against 118, when using a selectivity of 14 for the territory evaluator and 22 for the relative evaluator. When they both use a selectivity of 14 the territory wins 119 out of 200 games. This is surprising since we argued that the relative territory is better at estimating the value of a position. The difference in strength here can be explained by the difference in speed. The relative territory evaluator is inherently slower than his opponent, the fact that it has a less narrow selective search in the first two rows of table 7 even enhances that difference.

If however, we would have found optimal selectivity settings for both evaluators to play against each other we might have seen a different picture. The selective search for the relative territory evaluator would be narrower since it would have to use its strategic difference in evaluating positions better. The optimum selection would be closer to, or even smaller than, the selectivity of the territory evaluator. In the above games the relative territory evaluator got outclassed on speed which translated into a difference in search depth. This difference in

search depth would be minimised while still trying to maintain a selection big enough to outclass the opponent on the level of valuing positions. We can still see this difference in the experiments where both evaluators had a selectivity of 14.

In order to clarify the previous statement we did some additional tests where we limited the search depth for both players to 3 and gave them the same selective search of 18. The results are given in table 8. If our conjecture that the relative territory evaluator got outclassed on speed holds we should find that this evaluator is superior to the territory evaluator if we remove the speed difference by enabling them to always search to a constant depth of three ply.

Evaluators		Wins		Avg. branching factor		Nodes evaluated	
White	Black	White	Black	White	Black	White	Black
Rel. ter	Ter	56	44	353.13	321.82	13024507	15145902
Ter	Rel. ter	33	67	340.47	328.23	21042455	17023384

Table 8. Relative territory against territory with a 3 ply search depth limit.

What we note in this simulation is that, though the average branching factors for the relative territory evaluator are higher than his opponent in both experiments, it still manages to search to the same depth evaluating less nodes than his opponent. The fact that the evaluations of the relative territory evaluator are qualitatively better than the territory evaluations enables the relative territory evaluator to set the alpha and beta values in the alpha-beta search more effectively and thus causing it to prune more efficiently.

5.3.3 Mobility and random evaluator tests

We did not incorporate the mobility or random evaluators in our evaluator tests explicitly, since preliminary tests convincingly showed their inferiority to the territory evaluators.

5.4 Properties of the game on different board sizes.

In order to get more insight into the game of Amazons we tried to simplify the game without changing any rules. The best solution to do this is to make the board smaller. On a smaller board it is easier for humans to evaluate positions, to overview the possibilities and to see what the consequences of a move are. This insight might inspire us with new ideas to find a better evaluator. To this end we started investigating smaller board sizes. Because we still want to preserve the game rules as much as possible including the initial positions of each player, we only tried 4 different sizes. In the next sectios we will describe these different board sizes and show how we solved the game on the 4×4 and 5×5 boards.

One of the questions we had was: How important is the size of the board to the advantage of the beginning player? It is stated that the advantage of the first player, partly depends on the size of the board [20]. If the board is too small, the advantage of starting the game is not enough to bring the advantage to fruition.

5.4.1 4×4 Board

The smallest board investigated is the 4×4 board. On this board the white amazons are at the start positioned at a2, b1, c1, d2 and the black amazons at a3, b4, c4, d3 (see figure 13). The board has only 8 free spaces so a game has a maximum length of 4 full moves. Because of this the complexity is reduced drastically, which enables us to solve the game for this size.

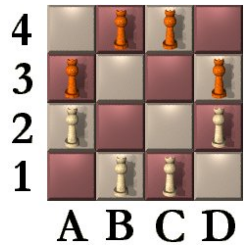


Figure 13. The initial position of Amazons on a 4×4 board.

If we look at the size of the board from a purely computational point of view, we are tempted to say that the second player wins because there are 8 free squares so the second player will be the last one to complete a move. We found that in this case the size of the board indeed is too small for the first player to take advantage of his initiative and win the game.
A principal variation is:

	White	Black
Move 1	b1-b3 (c2)	a3-b2 (c3)
Move 2	a2-a3 (a4)	b2-b1 (b2)
Move 3	a3-a2 (a3)	b1-a1 (b1)
Move 4	c1-d1 (c1)	d3-d4 (d3)

In an attempt to gain more insight in which moves are good and which are bad we also made a database of all the possible variations of the first two ply and computed their game-theoretical value. For more information on this database we refer to appendix B.

5.4.2 5×5 Board

The 5x5 board is somewhat more difficult because of its higher complexity. The amazons on this board are situated at a2, b1, d1 and e2 for White and a4, b5, d5 and e4 for Black (see figure 14).

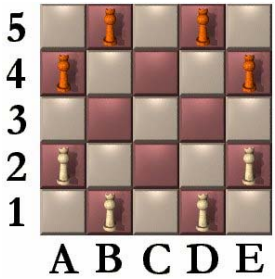


Figure 14. The initial position of Amazons on a 5×5 board.

When we tried to solve Amazons on this board we found that we had to change our way of searching. On the 4×4 board we used a full-width depth-first search and found the answer within a few seconds. On the 5×5 board this approach would have taken weeks. We therefore changed our strategy. We tried to prove that the first player has a win on this board, because from a computational point of view the first player has not only the starting advantage but also the fact that the board has an odd number of empty squares in the initial position. This means that if the board is entirely filled White would have completed the last move in the game. As a result a selective search for the first player is used, the second player however, is allowed to use the full range of possible moves. If we find in the end that White won the game then we can conclude that he has a forced win. However, since we did not know how many nodes we should select in our selective search we applied an iterative-widening approach. In this approach we start with an extremely-narrow selective search and widen it if it does not provide us with the information we are searching. In this way we started with a selective search of 2 moves in the alpha-beta search, which expectedly did not prove a forced win for White. It was not until we widened the selective search to 7 moves that we proved that White has a forced win on the 5×5 board. We cannot be sure that this is an optimal win, because we did not allow White to check all his possible moves. The win we found took all possible space on the board and was thus found at a search depth of 17 ply. It took 2 days of computing time on an Athlon 700Mhz to find these results investigating approximately 4.1 billion nodes. The principal variation found is shown below.

	White	Black
Move 1	b1-b4 (d4)	d5-b3 (b1)
Move 2	d1-c2 (d3)	e4-e3 (c1)
Move 3	c2-c5 (c2)	b5-c4 (d5)
Move 4	e2-d2(e2)	b3-b2(b3)
Move 5	a2-a3(a1)	a4-b5(a4)
Move 6	a3-a2(a3)	b2-c3(b2)
Move 7	b4-a5(b4)	e3-e4(e3)
Move 8	d2-d1(d2)	e4-e5(e4)
Move 9	d1-e1(d1)	

5.4.3 6×6 Board

The 6×6 board has two possible initial positions, one with the white amazons at a2, b1, e1 and f2 and the black amazons at a5, b6, e6, and f5, see figure 15. The second possible initial position is with the white amazons at a3, c1, d1 and f3 and the black amazons at a4, c6, d6 and f4, see figure 16.

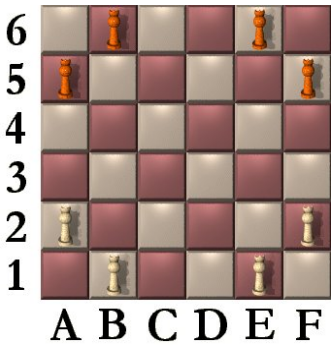


Figure 15. A 6×6 initial position.

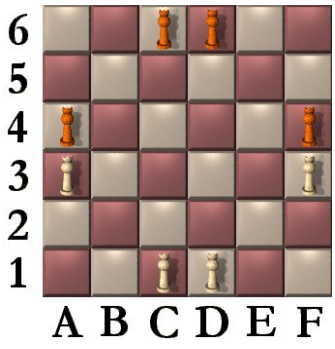


Figure 16. Another 6×6 initial position.

The interesting point about this board-size is that the two advantages discussed above in the previous sections are divided between the two players. White has the advantage of having the first move, while Black has the advantage of the board size. If the entire board is used then Black has the last move and thus wins the game. It would be interesting to see which of the two players has the decisive advantage. Unfortunately this size of the amazons board is already too big to solve.

5.4.4 7×7 Board

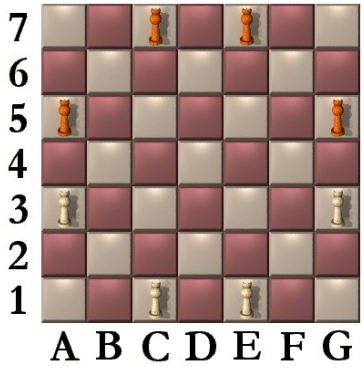


Figure 17. The initial position of Amazons on a 7×7 board.

On the 7×7 board the white amazons are located at a3, c1, e1, g3 and the black ones at a5, c7, e7, g5 (see figure 17). The maximum length of a game on this board is 41 plies. Unfortunately, the complexity of Amazons on this size is too high to solve the game. With the program used for the 5×5 board this would approximately take several years depending on how efficiently moves are selected in the iterative-widening process.

This size is nonetheless very good for gaining better insight in the strategic concepts of Amazons because on this board we can discern the same stages as Amazons on a 10×10 board. This is a subject of future research.

5.5 Chapter Conclusions

Throughout this chapter we have seen one compelling reason for not using several of the techniques we have investigated here. The selective search is the mayor culprit to the bad performance of killer moves, history heuristic and all other sorting techniques. The question arises if the selective search's performance warrants the decision to use it instead of using one or more of the sorting techniques effectively.

If we translate this question, we see that it comes down to the choice between a very narrow, well-guided search which will take us deeper or a wider, more reliable search which will not be able to look as deep in the same time.

The drawback of our choice is that we only search a very narrow tree and thus run a higher risk of not including the best move. We are however rather certain to find a good move, even if it is not the best one. The second strategy of searching will provide us with a move which is more likely to be the best move at this search depth. However, because our search effort is dispersed among more possible moves in this strategy, we will not be able to look at the consequences of this presumably best move in moves to come. It is this uncertainty which

made us choose the selective-search approach. We will take good moves and concentrate all our search efforts to these moves without looking at too many alternatives.

The second conclusion we can make here is that at this point there seems to be a trade-off between the territory and the relative-territory evaluators. The territory evaluator marginally beats the more sophisticated version on the grounds of its speed. If we look at the quality of assessment of the evaluators we see that the relative-territory evaluator has a better performance. This means that we have two measures with which we can compare these two evaluators. The first one is one of speed and the second one is quality. As is known from other game domains the initial increase in performance heavily depends on the search depth, which means that an increase in computing speed is important to enhance the playing strength. However, for a further increase in playing strength, the depth of search, and hence the speed, become less important and the evaluation function becomes the dominating factor. This means that we expect that the relative-territory function will become more important than the normal territory evaluation with increasing speed of the machine.

At this moment the territory evaluator still has the higher playing strength, but it is only a matter of time until the speed of the machines used will render the difference in speed between the two evaluators no longer a determining factor. It is then that we expect to see that the relative-territory evaluator will outperform the territory evaluator.

6 Conclusions

In this final chapter we present the main conclusions of this thesis. First, we will go back to the questions we posed in the first chapter and we will formulate answers to these questions according to the results of the research done. Second, we will give some general conclusions and we will also express our ideas about possible future research.

6.1 Problem statements revisited

After investigating the complexity, the search and the evaluation in the game of Amazons we have come to the point where we can answer the questions that have driven our research.

How can we use knowledge about Amazons in an efficient and effective implementation of the evaluator?

The first step in understanding the game was achieved by looking at the rules of the game. We noticed that mobility and territory control are essential notions to understand the game. An evaluator should be able to value both these characteristics efficiently (see [7]).

Previous research has led to the development of different evaluators. Mobility, territory and hybrid evaluators were suggested. We built upon the idea of the territory evaluator because it provides a better possibility to incorporate game-specific knowledge. We used the rules of the game and an investigation into its complexity to extract this knowledge.

A closer look at the relative-territory evaluator shows that it essentially combines knowledge from a territory evaluator with a mobility evaluator. The way we compute the territory is inherently bound to the mobility of the players. However, the territory evaluator goes beyond the notion of mobility alone. When the evaluator computes the territory it actually uses the mobility to determine which squares are available now and will be available in the future. This is true because apart from the fact that the territory evaluator computes all the spaces reachable in one move, it also uses this result to compute which squares will be available in two moves and so on. Computing continues until a depth is reached where each square can be occupied by at least one player. Here the similarities between the territory and relative-territory evaluators end because the relative territory evaluator adds extra information based on mobility. The relative-territory evaluator computes at this point how strong each player's territory is by checking how much faster the player can get to the squares in its territory compared to the opponent. Because of this extra information the relative-territory evaluator will not only cause the amazon to be moved to the right square but also computes more reliably where the barricade should be to increase the value of its territory.

By using the structure of what is defined as a 'move' we were able to incorporate knowledge efficiently into the evaluator so it can reliably value the worth of positions.

How can we apply and adapt certain interesting techniques, developed for computer-game playing, to the game of Amazons?

As stated in chapters 4 and 5 we investigated several techniques to enhance either the alpha-beta search, especially more efficient ways of ordering the nodes. Some techniques had to be adapted to the game. Especially the definition of a move was a reason for adaptation. It can be seen as two consecutive moves. For this reason we had to adapt the history heuristic where we introduced a second table in which the barricades were stored. Something similar

had to be done for the killer moves. Unfortunately nearly all our efforts on finding effective enhancements to the alpha-beta search failed.

The reason for this bad performance of already-established techniques can be carried back to the exceptionally-high average branching factor and the subsequent use of a strict selective search. We can conclude that in order to justify this strict selective search we need the best move ordering possible. This leaves only one option, ordering by evaluator, which by far outperforms the other ordering heuristics.

The idea of searching for only one player in the endgame as described in section 3.3.3, certainly has its merits and drastically reduces the search.

6.2 General conclusions

We have developed a program, which plays the game of Amazons at a reasonably high level. Even though we were not able to use the established techniques investigated in this thesis we still achieved our goal. We have come up with two very similar evaluators. The territory and the relative-territory evaluator. Both these evaluators perform at a high and competitive level. There is only a small difference in the way they work. At the moment the difference in performance is also small. In the near future, however, we predict that the relative-territory approach will become increasingly stronger as computing power increases. It is no advantage to search twice as fast as your opponent while this opponent only has to search 1/3 of the number of nodes you have to search to achieve the same depth. This is exactly the difference between the territory and the relative-territory evaluator. The first one is faster, but the second one evaluates better and can thus enable the alpha-beta search to prune away more nodes. The deeper the search the bigger the difference between these evaluators. At this moment the territory evaluator still prevails. We predict that within 2 years the quality of the evaluation will be more fruitful than the speed of the evaluation and the relative-territory evaluator will outperform the territory evaluator.

A second conclusion we can present here is that we found that in a game with a very high branching factor like Amazons, the established game-playing techniques do not work well and that other solutions have to be found. In this research we sought that solution in a heavy selective search. Other games might use the same solution at the cost of not being able to use many of the established techniques. To counter this loss we will have to come up with new techniques especially made for games with high branching factors, aiming at efficiently selecting a few moves out of many.

6.3 Future research

In this thesis, we described some techniques and ideas, which should be investigated in more detail. We address these issues briefly below.

6.3.1 Enhanced selective search

The selective search used in the program ANKY is in spite of numerous explorations, still quite crude. One idea to improve this is to design a selective search that takes the distribution of the scores for all the moves into account. This would mean that in positions where there is an obvious choice and thus a move with significantly better results in the move ordering, we can use a stricter selective search which would narrow down the alpha-beta search significantly. Alternatively, in positions where there is a significant number of moves eligible to be the best move, we decrease the selectivity and enhance the quality of the search.

6.3.2 Tuning the relative-territory-evaluation function

The relative-territory evaluator can still be significantly improved by finding optimal values for the rewards given when a player can reach a square faster than the opponent by 1, 2, 3 etc. moves. These parameters are extremely important, but unfortunately also hard to optimise. It is for instance clear that if we can reach a square 2 moves faster than the opponent, this is preferred over reaching it 1 move earlier. However, whether it is more important to reach a single square 3 moves earlier or to reach 2 squares 1 move earlier, is still very vague. More tests on this subject will optimise these factors. We believe that a machine-learning technique such as TDLeaf(λ) [3], might be a good approach to this problem since the search space is immense.

6.3.3 Designing an opening book for Amazons

Designing an opening book for Amazons is not of interest from the viewpoint of research, though it might enhance the playing strength of the program. In a certain aspect more knowledge about the importance of different factors in the opening phase of the game would not only be beneficial in designing an opening book, but could also be used in the evaluation function. In short, there is still a great deal to learn about the game of Amazons. The growing interest in this game from artificial-intelligence researchers and competitive game players alike will ensure us that we will steadily learn more and more about this game.

6.3.4 Quiescence search

Although we did not use any kind of quiescence search in ANKY, we still think that in some specific positions a modified quiescence search could be very beneficial.

The positions referred to are characterised by the fact that they always increase the number of absolute squares on the board. In these positions there is an increased risk of making territories that cannot be filled optimally. To remedy this, quiescence search for these moves would find any deficiencies in the new absolute territory very quickly. In this search only moves for the amazon which caused the absolute territories would need to be investigated. Because we use a selective search these deficiencies in territories will be harder to find in the normal alpha-beta search. This can be considered as a special case of the horizon effect. Further research is necessary to investigate the usefulness of such a quiescence search.

7 References

1. Akl, S.G. and Newborn, M.M. (1977). The Principal Continuation and the Killer Heuristic. *1977 ACM Annual Conference Proceedings*, pp. 466-473. ACM, Seattle.
2. Arbiser, A. Amazon Unofficial Homepage. http://www.dc.uba.ar/people/materias/pf/amazons/.
3. Baxter, J. Tridgell, A. and Weaver, L. (1998). Experiments in Parameter Learning Using Temporal Differences. *ICCA Journal*, Vol. 21, No. 2, pp. 84-99.
4. Beal, D.F. and Smith, M.C. (1994). Random Evaluations in Chess. *ICCA Journal*, Vol. 17, No. 1, pp. 3-9.
5. Breuker, D.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (1994). Replacement Schemes for Transposition Tables. *ICCA Journal*, Vol. 17, No. 4, pp. 183-193.
6. Buro, M. (2000). Simple Amazons endgames and their connection to Hamilton Circuits in cubic subgrid graphs. *NECI Technical Report #71* (2000). NEC Research Institute.
7. Hashimoto, T., Kajihara, Y., Sasaki, N. and Iida, H. (2001). An Evaluation Function for Amazons. *Advances in Compuer Games*, Vol. 9, pp. 191-202.
8. Hensgens, P.P.L.M. and Uiterwijk, J.W.H.M. (2000). 8QP wins Amazons tournament. *ICGA Journal*, Vol. 23, No. 3, pp. 179-180.
9. Iida, H. and Müller, M. (2000). Report on the Second Open Computer-Amazons Championship. *ICGA Journal*, Vol. 23, No. 1, pp. 51-54.
10. Junghans, A. (1998). Are there Practical Alternatives to Alpha-Beta? *ICCA Journal*, Vol. 21, No. 1, pp. 14-32.
11. Knuth, D.E. and Moore, R.W. (1975). An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, Vol. 6, No. 4, pp. 293-326.
12. Kotani, Y. (1999). An Evaluation Function for Amazons. *Proceedings of the 40th Programming Symposium*, Information Processing Society of Japan, Tokyo, Japan. P. 103 (in Japanese).
13. Marsland, T.A. (1986). A Review of Game-Tree Pruning. *ICCA Journal*. Vol. 9, No. 1, pp. 3-19.
14. Samuel, A.L. (1967). Some Studies in Machine Learning using the Game of Checkers. II – Recent Progress. *IBM Journal of Research and Development*, Vol. 2, No. 6, pp. 601-617.
15. Sasaki, N. and Iida, H. (1999). Report on the First Open Computer-Amazon Championship. *ICCA Journal*, Vol. 22, No. 1, pp. 41-44.
16. Schaeffer, J. and Plaat, A. (1997). Kasparov versus Deep Blue: The Rematch. *ICCA Journal*, Vol. 20, No. 2, pp. 95-101.
17. Schaeffer, J. (1983). The History Heuristic. *ICCA Journal*, Vol. 6, No. 3, pp. 16-19.
18. Tegos, T. The Amazons Board Game, http://www.cs.ualberta.ca/~tegos/amazons/index.html.
19. Hashimoto, T., Kajihara, Y., Sasaki, N., Iida, H. and Jin, Y. (1999). An Evaluation Function for AMAZON. Shizuoka University, Japan.
20. Uiterwijk, J.W.H.M. and Herik, H.J. van den (2000). The advantage of the initiative. *Information Sciences*. No. 122, pp. 43-58.
21. Zobrist, A.L. (1970). A New Hashing Method with Application for Game Playing. *Technical Report 88*, Computer Science Department, The University of Wisconsin, Madison, WI, USA.

Appendix A. Amazons at the 5th Computer Olympiad

The following contribution was published in the *ICGA Journal* [8] as a report on the Amazons tournament of the 5th Computer Olympiad.

The 5th Computer Olympiad, held during the MSO in London from 21st to 25th of August, for the first time included an Amazons tournament. There have been tournaments in this game in the past (see the reports on the first and second Amazons World Championships in the reference list), but it was the first time for the Computer Olympiad to host a tournament of this game.

The tournament turned out to be very exciting with some of the strongest Amazons programs[1] competing in it. Even though entrants had to subscribe their programs on a somewhat short notice there were still six competitors. We hope this trend will continue throughout the next Computer Olympiads. Table 1 provides an overview of the participants. Table 2 shows the pairings.

No.	Program	Programmer
1	YAMAZON	Hiroshi Yamashita
2	ANTIOPE	Theo Tegos (University of Alberta)
3	ASKA	Yoichiro Kajihara (Shizuoka University)
4	8QP	Johan de Koning
5	ANKY	Patrick Hensgens (Universiteit Maastricht)
6	OTRERE	Paul Utgoff

Table 1: The programs and their programmers.

Pairing			
Round 1	1-6	2-5	3-4
Round 2	1-2	6-4	5-3
Round 3	3-1	2-6	4-5
Round 4	1-4	3-2	6-5
Round 5	5-1	2-4	3-6

Table 2: The pairing.

[1] Unfortunately Michael Buro's AMSBOT, considered one of the top Amazons programs, did not enter this tournament.

Results

In Table 3 the final standings are given. The notation *x* / *y* stands for *x* points when playing White (first), and *y* points when playing Black (second). For a win, a full point is given; for a loss zero points. Moreover, for a win the net score is given between brackets.

	8QP	**YAMAZON**	**ANKY**	**ANTIOPE**	**ASKA**	**OTRERE**	**Total Pts(score)**	**Rank**
8QP	X / X	1(7) / 1(15)	1(13) / 1(7)	1(6) / 1(26)	1(18) / 1(23)	1(19) / 1(6)	10(140)	1
YAMAZON	0 / 0	X / X	0 / 1(9)	1(9) / 1 (14)	1(17) / 1(5)	1(9) / 1(5)	7(68)	2
ANKY	0 / 0	0 / 1(1)	X / X	1(6) / 1 (2)	1(13) / 1(4)	1(11) / 1(24)	7(61)	3
ANTIOPE	0 / 0	0 / 0	0 / 0	X / X	1(7) / 1 (2)	1(24) / 1(0)	4(33)	4
ASKA	0 / 0	0 / 0	0 / 0	0 / 0	X / X	1(9) / 1(8)	2(17)	5
OTRERE	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0	X / X	0(0)	6

Table 3: The final standings of the Amazons tournament.

Table 3 shows that Johan de Koning with his 8QP won all his games rather convincingly. As also can be seen, there was quite a struggle for the silver medal, and the pairings (Table 2) show that this struggle had to be settled in the last round, in a direct confrontation between YAMAZON and ANKY. The authors might be biased, but they believe that the first encounter between YAMAZON and ANKY is most interesting. Therefore we reproduce this game and provide some comments.

YAMAZON vs. ANKY (Round 5)

1. d1-d7 (g7) a7-d4 (a7) 2. g1-e3 (e10) g10-e8 (e6) 3. d7-d8 (c9) d10-b10 (b3) 4. a4-b4 (b9) d4-f4 (b8) 5. j4-i4 (f7) e8-c6 (d6) 6. b4-e4 (c4) b10-d10 (f8)

At this point YAMAZON had a slight advantage according to ANKY's evaluation. In the next few moves the advantage gradually disappears. Then the game slowly evolves into an advantage for ANKY at move 14. Thereafter the advantage steadily increases to approximately 3 squares.

7. i4-j5 (f9) d10-e9 (d9) 8. e4-h7 (i6) f4-f5 (a5) 9. e3-f3 (g4) f5-g5 (g6) 10. h7-h8 (i8) g5-i5 (h6) 11. h8-h10 (j8) j7-h7 (h9) 12. f3-f2 (f3) h7-h8 (g9) 13. h10-i10 (i9) h8-j6 (j7) 14. d8-b6 (d8) c6-e4 (b7) 15. b6-b5 (e8) e4-d4 (d5) 16. f2-d2 (d3) i5-i2 (i4) 17. d2-h2 (b2) i2-h1 (g2) 18. j5-i5 (e1) e9-f10 (h10) (Diagram 1)

ANKY faces a small problem in the upper-left area. As is obvious from Diagram 1 this area cannot be filled entirely and so ANKY will "lose" one square which made the ending a nerve wrecking.

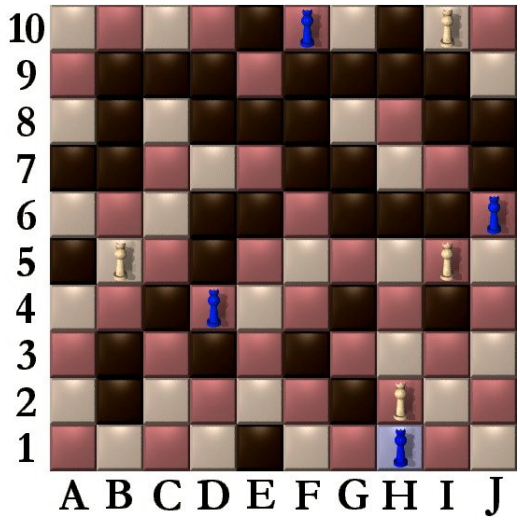


Diagram 1: Position after Black's 18th move.

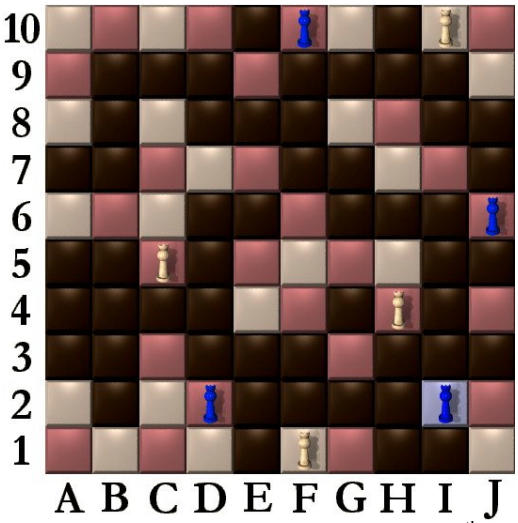


Diagram 2: Position after Black's 26th move.

19. i5-g5 (i5) h1-g1 (e3) 20. g5-j2 (j5) d4-c5 (a3) 21. j2-f6 (d4) c5-b4 (a4) 22. b5-c5 (b5) g1-i1 (i3) 23. f6-h4 (h3) b4-d2 (f2) 24. h2-i2 (h2) d2-c3 (b4) 25. i2-h1 (j3) i1-i2 (i1) 26. h1-f1 (h1) c3-d2 (e2) (Diagram 2).
At this point the territory of both players is definite. The rest of the game is only played for convenience and as a test for both programs to see whether the programs were able to fill their territories optimally.
27. f1-g1 (f1) i2-j2 (j1) 28. i10-j10 (j9) j2-i2 (j2) 29. j10-i10 (j10) f10-e9 (f10) 30. h4-g5 (h5) e9-d10 (a10)??
Because of this blunder ANKY lost another square which meant that the advantage was down to one square. Since no other mistakes were made, ANKY won the game by one square.
31. g5-h4 (g3) d10-b10 (a9) 32. c5-b6 (a6) b10-c10 (b10) 33. b6-c5 (c8) c10-d10 (c10) 34. h4-e7 (h4) d10-e9 (d10) 35. e7-c7 (b6) j6-i7 (h8) 36. c7-c6 (c7) i7-h7 (g8) 37. c6-d7 (c6) h7-i7 (j6) 38. d7-e7 (d7) i7-h7 (i7) 39. e7-f6 (e7) d2-c3 (d2) 40. f6-g5 (f6) c3-c2 (d1) 41. g5-f5 (g5) c2-b1 (a2) 42. f5-f4 (e4) b1-c2 (c3) 43. f4-f5 (f4) c2-b1 (a1) 44. f5-e5 (f5) b1-c2 (b1).

References

Sasaki, N. and Iida, H. (1999). Report on the First Open Computer-Amazons Championship, *ICCA Journal*, Vol. 22, No. 1, pp. 41-44.

Iida, H. and Müller, M. (2000). Report on the Second Open Computer-Amazons Championship, *ICGA Journal*, Vol. 23, No. 1, pp. 51-54.

Appendix B. Amazons at the 6th Computer Olympiad

Despite the fact that ANKY ended at the 4th place in this tournament, it did again prove to be highly competitive. This can also be seen if we count the number of times ANKY lost with 2 or less squares left for the opponent, out of twelve games this happened five times.

scoretable

id	day	time	table	player1	player2	minutes	moves	score1	score2	points1	points2
1	Su	16-18	1	**Anky**	**Aska**	0	0	1	0	8	0
2	Su	16-18	2	**8QP**	**Invader**	0	0	1	0	20	0
3	Su	18-20	1	**8QP**	**Anky**	0	0	1	0	2	0
4	Su	18-20	2	**Invader**	**Aska**	0	0	0	1	0	4
5	Mo	16-18	1	**Anky**	**Invader**	0	0	0	1	0	0
6	Mo	16-18	2	**Aska**	**8QP**	0	0	0	1	0	13
7	Mo	18-20	1	**Aska**	**Anky**	0	0	1	0	21	0
8	Mo	18-20	2	**Invader**	**8QP**	0	0	0	1	0	19
9	Tu	16-18	1	**Anky**	**8QP**	0	0	0	1	0	9
10	Tu	16-18	2	**Aska**	**Invader**	0	0	1	0	5	0
11	Tu	18-20	1	**Invader**	**Anky**	0	0	1	0	1	0
12	Tu	18-20	2	**8QP**	**Aska**	0	0	1	0	28	0
13	We	16-18	1	**Anky**	**Aska**	0	0	0	1	0	1
14	We	16-18	2	**8QP**	**Invader**	0	0	1	0	16	0
15	We	18-20	1	**8QP**	**Anky**	0	0	1	0	18	0
16	We	18-20	2	**Invader**	**Aska**	0	0	1	0	2	0
17	Th	10-12	1	**Anky**	**Invader**	0	0	0	1	0	8
18	Th	10-12	2	**Aska**	**8QP**	0	0	0	1	0	11
19	Th	12-14	1	**Aska**	**Anky**	0	0	0	1	0	3
20	Th	12-14	2	**Invader**	**8QP**	0	0	0	1	0	28
21	Th	14-16	1	**Anky**	**8QP**	0	0	0	1	0	6
22	Th	14-16	2	**Aska**	**Invader**	0	0	1	0	2	0
23	Th	16-18	1	**Invader**	**Anky**	0	0	1	0	1	0
24	Th	16-18	2	**8QP**	**Aska**	0	0	1	0	7	0

The final ranking was:

	score	Points
8QP (de Koning)	12	177
Aska (Iida)	5	33
Invader (Avetisyan)	5	12
Anky (Hensgens)	2	11

Appendix C. A database for 4×4 Amazons.

As described in section 5.4.1 we constructed a database for the 4×4 board (see diagram 1) of game-theoretic values for all the combinations of the first two ply. Among this information, the database also contains information about the depth of the principal variations and the number of nodes investigated to find it.

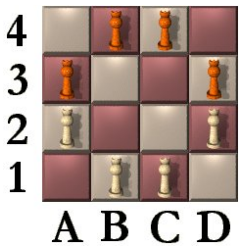


Diagram 1. The initial position of Amazons on a 4×4 board.

With the help of this database we can for instance see which moves are 'bad' and which moves are 'good' for White. According to the database the worst move White could make, is d2-d1 (b3). This leaves 52 possible moves for Black, 45 of these moves have a winning value for Black. If we compare this to move b1-b3(c2), chosen by White in the principal variation in section 5.4.1, we see that White did not make it easy for Black since out of 27 possible counter moves, there was only one move which wins for Black.

In total the database contains all 4332 combinations of the first two ply of which only 1566 are winning for Black.

The complete database can be inspected at http://www.fdaw.unimaas.nl/ktum/hensgens/