Constructing an Abalone Game-Playing Agent

N.P.P.M. Lemmens

18th June 2005

Abstract

This paper will deal with the complexity of the game Abalone[1] and depending on this complexity, will explore techniques that are useful for constructing an Abalone game-playing agent. It turns out that the complexity of Abalone is comparable to the complexity of Xiangqi and Shogi. Further, some basic heuristics and possible extensions to these basic heuristics are described. With these extensions the agent did not even loose once in the test-games played against third-party implementations although it only played at a search depth of 2. So these heuristic extensions turn out to be a valuable tool with which the Abalone agent will play more advanced.

1 Introduction

With the rise of computers and techniques to automate processes, mankind has taken on the challenge to construct machines that act like humans do. The reasons for this is the lack of human presence or the inability for humans to be present to perform tasks. Mankind has thus started to examine how to make machines intelligent. One of the major problems in this area is how to make machines choose from an enormous amount of possibilities and decisions. In what way should a machine act in a certain situation?

In computer game playing, specifically board games, the same problems arise. The computer has to decide what action in a given situation has to be made. The games can be divided into four categories. These categories are characterized by two features, namely the state-space complexity and the game-tree complexity. These two features will be explained in section 4. In figure 1 these four categories are graphically presented. The figure also shows the meaning of being in a specific category. Especially the fourth category (large state-space complexity and high game-tree complexity) is a typical example of a hard to tackle problem for computers. Games like Go and Chess are games belonging to this category.

[1] Abalone is a registered trademark of Abalone S.A. - France.

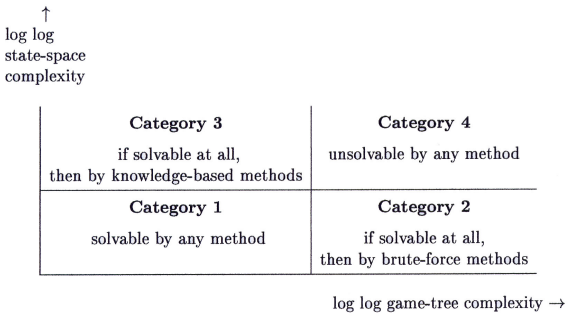

Figure 1: Game Categories [11].

In 1988 Abalone was introduced to the world. This perfect information game, which is based on the Japanese Sumo wrestling, is expected to reside in the same category as Go and Chess. Our goal is to make an Abalone game-playing agent. The following problem statement is formulated for this paper:

What algorithms and heuristics are valuable for the construction of an Abalone game-playing agent?

In this paper the following research questions will be answered:

- *What is Abalone's state-space complexity and game-tree complexity?*
- *Depending on these complexities, which algorithms are the most promising to develop a computer agent that is capable of playing Abalone?*
- *What adjustments and extensions of these algorithms make the agent more advanced?*

In the next section some basic approaches for playing computer games will be illustrated. In section 3 the Abalone game will be described. Section 4 will present the complexity for the game Abalone. The Abalone agent and its approaches for playing the game are explained in section 5. In section 6 the performance of the various approaches are evaluated and results of playing against other Abalone implementations are given. The conclusions will be provided in section 7. Finally, in section 8, future research will be presented.

2 Basic Approaches

This section gives a short overview of the basic algorithms used to develop an Abalone-playing agent. In section 2.1 the basic Minimax algorithm will be described. In section 2.2 an extension of this Minimax algorithm, namely the Alpha-Beta algorithm, will be explained.

2.1 Minimax

As Abalone is a two-player, perfect-information game, no randomness (like chance) is involved. When playing, the two players are in fact trying to respectively maximize and minimize the score function from the first player's view. Thus the 'maximizing player' (i.e., the first player) will try to determine a move which will give him/her the maximum score while giving the other player the minimum score. Given a search tree (for an impression of how a Minimax search tree looks like, see figure 2), the search proceeds as follows: the branches at the root of the tree represent all possible moves for the MAX player, each leading to a new game position. The branches at each node of one of these branches represent again all possible moves, but now for the MIN player, and so on. Supposing that both players are playing optimally then the best move for MAX will be one where the lowest score reachable by MIN is as high as possible. At the end of the tree (the leaves) the score function is calculated for the corresponding game positions. The algorithm uses a simple recursive computation of the minimax values of each successor state, directly implementing the defining equations. The recursion proceeds all the way down to the leaves of the tree, and then the minimax values are backed up through the tree as the recursion unwinds. The minimax algorithm performs a complete depth-first exploration of the search tree [9]. In figure 2 Minimax search leads to a best move with value 3. The algorithm's complexity is b^d, where b stands for the average branching factor and d stands for the depth of the search tree.

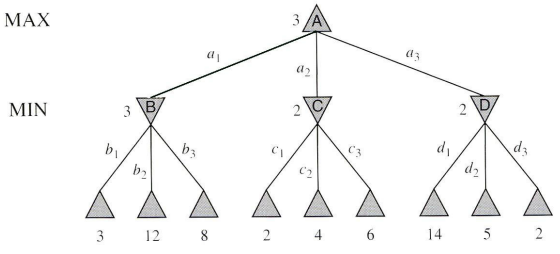

Figure 2: Minimax search tree [9].

2.2 Alpha-Beta

The main problem with Minimax search is that the number of game states it has to examine is exponential in the number of moves. As is noted in section 2.1 Minimax's complexity is b^d. By using the Alpha-Beta algorithm (provided it uses optimal move ordering) it is possible to reduce the complexity effectively to $b^{\frac{d}{2}}$. Alpha-Beta search is an extension of the Minimax algorithm. The algorithm computes the correct Minimax decision without looking at every node in the game tree. The algorithm uses pruning in order to eliminate large parts of the tree from consideration. Essentially, it detects nodes which can be skipped without loss of any information. For the

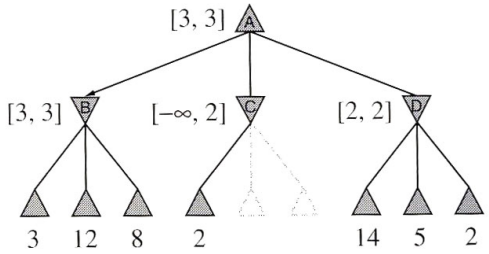

Figure 3: Alpha-Beta search tree [9].

example search tree given in figure 3, the algorithm proceeds as follows: After examining the first move of MAX it is known that the player can do a move with a value of at least 3. When the MAX player tries its second move it detects that the MIN player has a move leading to a score of 2 on its first move. Now it can be concluded that there is no need to explore the rest of the subtree because if there are moves exceeding 2, the MIN player is expected not to make them. So the best the MAX player can get out of this subtree is the move with score 2. This move will eventually be ignored since the MAX player already has a move with a value of 3.

Generally speaking the algorithm proceeds as follows: consider a node n somewhere in the tree, such that the player has a choice of moving to that node. If the player has a better choice m either at the parent node of n or at any choice point further up, then n will never be reached in actual play. So once we have found sufficient information about n (by examining some of its decendants) to reach this conclusion, we can prune it [9].

Alpha-beta search gets its name from the following two parameters that describe bounds on the backed-up values that appear anywhere along the path. The first one is α, which is the best score that can be forced. Anything worth less than this is of no use, because there is a strategy that is known to result in a score of α. Anything less than or equal to α is no improvement. The second is β. β is the worst-case scenario for the opponent. It is the worst thing that the opponent has to endure, because it is known that there is a way for the opponent to force a situation no worse than β, from the

opponent's point of view. If the search finds something that returns a score of β or better, it is too good, so the side to move is not going to get a chance to use this strategy [6].

3 The Game Abalone

Abalone is a strategy board-game comparable to Chess and Go. Since its introduction in 1988 it has grown in popularity. It has been sold over 4 million times in 30 countries and has over 12 million players. In 1998 the game was ranked 'Game of the Decade' at the International Game Festival [1].

Abalone is a game on a hexagonal board on which two groups of 14 marbles oppose each other. The game rules are simple. The player who first pushes off 6 of his/her opponents marbles wins. Players move in turn. After tossing for colours, black plays first. In section 3.1 the move rules will be given. Section 3.2 will give some insight in possible dead-end situations.

3.1 Moves

Figure 4 shows the game's start position. At a player's turn, one, two or three marbles (of the player's own color) together may be pushed in any of the six possible directions. That is, provided there is either an adjacent free space behind the group or a 'sumito' situation (see below). When two or three marbles of the same colour

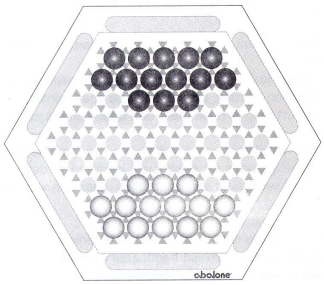

Figure 4: Abalone's start position [4].

are pushed together, they all must be moved in the same direction. A move can be either broadside or inline. See figures 5 and 6.

Moving more than three marbles of the same colour in one turn is not allowed. One, two, or three marbles of the same colour, which are part of a larger row, may be separated from the row played.

To push the opponent's marbles the player has to construct a so called 'sumito' situation. A 'sumito' situation is one of the three superiority positions. A 'sumito' situation occurs when the player's marbles outnumber the opponents marbles, i.e., 3-to-2, 3-to-1, 2-to-1. See figure 7. The opponent's marbles may only be pushed

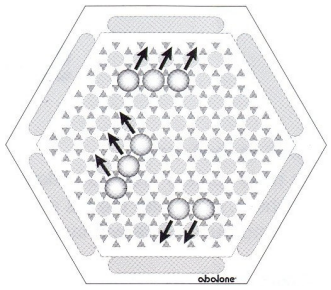

Figure 5: Broadside moves [4].

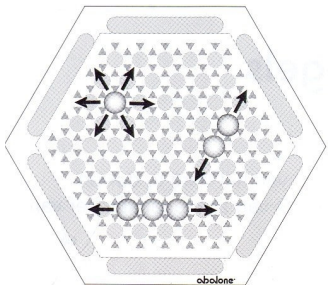

Figure 6: Inline moves [4].

'inline', when in contact and only provided there is a free space behind the attacked marble or group of two marbles.

In order to score it is necessary to push the opponent's marbles over the edge of the board.

Figure 7: Sumito moves [4].

In case of an even-balanced situation no pushing of opponent's marbles is allowed. These situations are called 'pac' situations. These occur when the situation is a 3-to-3, 2-to-2 or 1-to-1 situation. See figure 8. The 'pac' situation can be broken along a different line of action.

Figure 9 shows moves that are not allowed. The upper situation, where Black wants to push White to the right, is not allowed because of the single black marble to the right of the white group. The middle one, where Black wishes to push the white marbles, is not allowed since there is a free space between the (black) pushing group and the (white) to-be-pushed group. In the lower situation Black wants to push White around the corner. As only inline pushes are valid this push is not allowed. Suicide moves (pushing your own marble off the board) are not allowed either (not shown in the figure).

Figure 8: Pac situations [4].

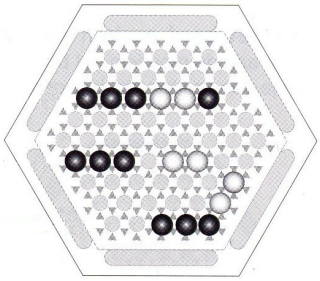

Figure 9: Unallowed moves [4].

3.2 Dead-end positions

Dead-end positions are positions that are reachable but in which no move is possible. See figure 10. To reach

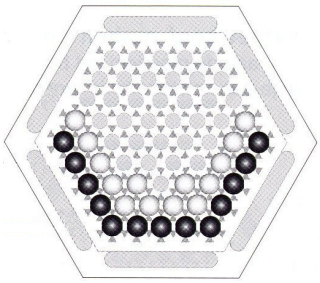

Figure 10: Example of a dead-end position.

these positions the players have to co-operate or play randomly. It is unlikely that these positions are reached by normal play. Abalone rules do not mention what happens when such a dead-end position is reached (i.e., does that player lose?). Our agent will gracefully mention that no move can be found and terminate the game.

4 Abalone's Complexity

The property complexity in relation to games is used to denote two different measures, which are named the state-space complexity and the game-tree complexity. In figure 11 a rough distribution of games based on these two measures can be seen. In section 4.1 the state-space complexity definition will be given and the value for Abalone determined. Section 4.2 will handle the game-tree complexity definition and the value for Abalone will be determined.

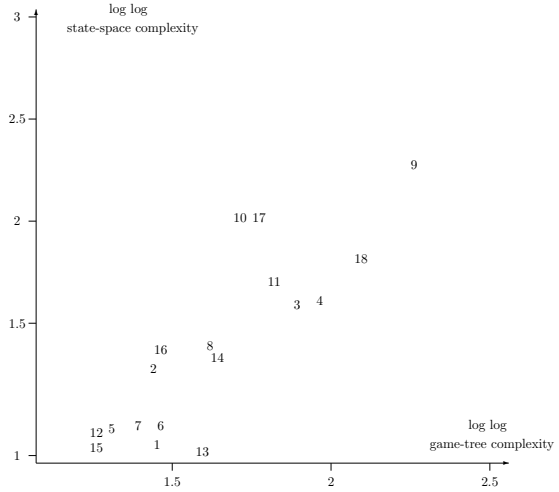

Figure 11: Game distribution based on complexity [11]. Awari (1), Checkers (2), Chess (3), Chinese Chess (4), Connect-Four (5), Dakon-6 (6), Domineering (8×8) (7), Draughts (10×10) (8), Go (19×19) (9), Go-Moku (15×15) (10), Hex (11×11) (11), Kalah(6,4) (12), Nine Men's Morris (13), Othello (14), Pentominoes (15), Qubic (16), Renju (15×15) (17), Shogi (18).

4.1 State-space complexity

The state-space complexity of a game is defined as the number of legal game positions reachable from the initial position of the game [3]. Because calculating the exact state-space complexity is hardly feasible it is necessary to make an approximation of it. For the upper bound of Abalone's state-space complexity one has to make all possible combinations in each of the legal situations. All situations where one player has nine marbles and the other less than eight are illegal (since the game ends when a player looses the sixth marble). The state-space complexity can therefore be approximated by the following formula:

$$\sum_{k=8}^{14} \sum_{m=9}^{14} \frac{61!}{k!(61-k)!} \times \frac{(61-k)!}{m!((61-k)-m)!} \qquad (1)$$

This approximation has to be corrected for symmetry. Symmetrical situations can be mirrored and rotated and will therefore occur more than once. The Abalone game has 6 possible 'mirrors' and 6 possible 'rotations'. So in order to correct the state-space complexity we divide the found state-space complexity by 12. The state-space complexity will then result in approximately: 6.5×10^{23}.

4.2 Game-tree complexity

A game tree is a tree whose nodes are positions in a game and whose branches (edges) are moves. The complete game tree for a game is the game tree starting at the

initial position and containing all possible moves from each position.

The number of leaf nodes in the complete game tree is called the game-tree complexity of the game. It is the number of possible different ways the game can be played. The game tree is typically vastly larger than the state space in many games. For most games it is usually impossible to work out the size of the game tree exactly, but a reasonable estimate can be made. In order to calculate an estimate, two factors have to be known. These factors are the average branching factor of the tree and the number of ply (half-moves) of the game.

The branching factor is the number of children of each node. If this value is not uniform, an average branching factor can be calculated. In Abalone, if we consider a 'node' to be a legal position, the average branching factor is about 60. This means that at each move, on average, a player has about 60 legal moves, and so, for each legal position (or 'node') there are, on average, 60 positions that can follow (when a move is made). An exhaustive brute-force search of the tree (i.e., by following every branch at every node) usually becomes computationally more expensive the higher the branching factor, due to the exponentially increasing number of nodes. For example, if the branching factor is 10, then there will be 10 nodes one level from the current position, 100 nodes two levels down, 1000 three levels down, and so on. The higher the branching factor, the faster this 'explosion' occurs.

A *ply* refers to a half-move: one turn of one of the players. Thus, after 20 moves of an abalone game, 40 ply have been completed, 20 by White and 20 by Black. One ply corresponds to one level of the game tree.

The game-tree complexity can be estimated by raising the game's average branching factor to the power of the number of ply in an average game.

The average game-length of Abalone is 87 ply. This is determined with the help of the PBEM-archive which can be found on the internet [8]. As the average branching factor is 60 we can calculate the resulting game-tree complexity: 5.0×10^{154}. This game-tree complexity lies between the game-tree complexity of 'Xiangqi' and 'Shogi' as can be seen in Table 1.

Game	Log(State-Space)	Log(Game-Tree)
Tic-Tac-Toe	3	5
Nine Men's Morris	10	50
Awari	12	32
Pentominoes	12	18
Connect Four	14	21
Backgammon	20	144
Checkers	21	31
Lines of Action	24	56
Othello	28	58
Chess	46	123
Xiangqi	75	150
Shogi	71	226
Go	172	360

Table 1: Complexities of well-known games [5].

5 The Abalone Agent

This section describes the techniques and algorithms used to make the basic agent more advanced. The basic agent resembles the more advanced one with exception of the heuristic extensions. Both agents use Alpha-Beta as their main search algorithm. Section 5.1 handles the algorithm extension 'Move Ordering'. Section 5.2 handles the algorithm extension 'Transposition Table'. Finally section 5.3 presents the 'Evaluation functions' or heuristics.

5.1 Move Ordering

In order to optimize search speed and efficiency of the Alpha-Beta search it is practical to order the possible moves in such a way that the most promising ones are evaluated first. Knowing that pushing moves are more worthwhile above, for example, a move of one single marble, these moves are presented first. Furthermore they are ordered from large groups to small groups. Another extension to the order algorithm is evaluating the board position of the group of marbles. If for example a group of marbles resides in the start position at the top of the board it is more efficient to present the moves, which get the marbles as quick as possible to the center, first.

5.2 Transposition Table

The search cost is one of the big disadvantages of the Minimax algorithm. Although the search cost is greatly reduced by the Alpha-Beta extension and the move ordering it is still worthwhile to extend the algorithm even more. Repeated states occur frequently because of transpositions. Transpositions are different move sequences that end up in the same position. It is worthwhile to store the evaluation of this position in a hash table the first time it is encountered, so that on a subsequent occurence it does not have to be re-evaluated.

5.3 Evaluation Functions

Game-tree search assumes that the used evaluation function (or heuristic) will give a good interpretation of the current position. When it is possible to look deep enough in the search tree the evaluation function does not have to be very advanced. For example, if it is possible to search in the tree (within a reasonable amount of time) to the end of the game, the evaluation function would be just a check of who won. However in practice it is often not possible to search to the end of the game in the search tree. Thus the evaluation function has to interpret the situation at a certain (non-terminal) depth

of the game tree. The more advanced the evaluation function is, the less deep the search has to go.

The basic constructed agent uses a simple evaluation function in that it:

- keeps the marbles around the middle of the board and forces the opponent to move towards the edges;
- keeps the marbles together as much as possible, to increase both offensive and defensive power.

As stated in Ozcan and Hulagu [7] these heuristics perform quite well. In order to further reduce the search depth these strategies are extended with:

- try to break strong groups of the opponent by pushing out the center marble, thus both dividing the opponent and creating a good defence since the opponent cannot push when its own marbles are in the way;
- try to push off the opponent's marbles (and keep your own on the board) because it weakens the opponent and therefore strengthens your own position;
- strengthen a group when it is in contact with the opponent.

Depending on the situation on the board the weights for these strategies are adapted. A rough weight-setting has been found by trial and error.

At first the constructed agent tries to get to the center with an as large as possible cohesion. Once the center has been reached and the opponent is pushed far enough out of the center, the agent weakens the will to get to the center and strengthens the 'agressive' strategies (i.e., break strong groups and pushing off opponent's marbles) while trying to preserve its own cohesion. When the opponent's cohesion breaks up the 'agressive' strategies are further strenghtened. In the endgame the extended agent will almost not care about the center but will try (as tactically as possible, i.e., without losing own marbles) to push off the opponent's marbles, giving the opponent no chance to recover.

The evaluation function looks like this:

$$eval(s) = \sum_{i=1}^{5} w_i \times f_i(s) - w_6 \times f_6(s) \qquad (2)$$

Here $f_1(s)$ stands for the distance to the center which is calculated by taking the difference between the Manhattan distances (of each player's marbles) to the center of the board (i.e., position *e5* on the board) depending on the state s of the game.

$f_2(s)$ is the cohesion strategy. This strategy determines the number of neighboring teammates of each marble for each player in state s of the game. After this the difference between them is calculated.

$f_3(s)$ is the 'break-strong-group' strategy. This strategy determines how many strong groups are broken by the player's marbles in the state s of the game. In order to determine this value for a player each marble (of that player) is checked for an opponent marble at one adjacent side of the marble and an opponent marble at the opposing adjacent side. Again the difference between the values for both players is calculated.

The 'strengthen-group' strategy is denoted by $f_4(s)$. This strategy calculates the number of contact positions by looking at one adjacent side of the player's marble for a teammate and on the opposing adjacent side of the marble for an opponent marble.

$f_5(s)$ stands for the 'number-of-marbles' strategy. This strategy calculates the difference between the number of opponent marbles before the search began and the number of opponent marbles on the board in gamestate s.

Finally $f_6(s)$ is equal to $f_5(s)$ but deals with the player's own marbles.

5.4 Evaluation Function Weights

In order to make the strategies work they have to be played with certain strength. Furthermore it is important not to play in the same way all the game. The extended agent plays therefore in nine different modi. These modi are delimited by the values of the center strategy and the cohesion strategy. In Table 2 the conditions and the corresponding strategy's weight values can be seen. Table 2 shows the agent plays for the center at first. As the agent's cohesion grows with respect to the opponent's cohesion it strengthens the 'agressive' strategies and weakens the will to own the center while preserving cautiousness for loosing own marbles.

Modus	Center	Cohesion	w_1	w_2	w_3	w_4	w_5	w_6
1	< 0	NA	3	2	6	1.8	0	$50 \times w_5$
2	< 5	NA	3.3	2	6	1.8	35	$50 \times w_5$
3	≥ 5	$0 \leq x < 4$	2.9	2	15	3	4	$50 \times w_5$
4	≥ 5	$4 \leq x < 10$	2.9	2	15	3	15	$50 \times w_5$
5	≥ 5	$10 \leq x < 16$	2.8	2.3	25	3	15	$50 \times w_5$
6	≥ 5	$16 \leq x < 22$	2.8	2.1	25	3	25	$50 \times w_5$
7	≥ 5	$22 \leq x < 28$	2.7	2.3	25	3	30	$50 \times w_5$
8	≥ 5	$28 \leq x < 34$	2.4	2.3	25	3	35	$50 \times w_5$
9	≥ 5	≥ 34	2.2	2.3	25	3	40	$50 \times w_5$

Table 2: Modus conditions and corresponding strategy weights. NA means 'not applicable'.

6 Performance

In this section the performance of the Abalone agent will be outlined. In order to test the performance of the agent it had to play against itself (with the basic heuristic and the extended heuristic) and against third party (commercial) implementations. In this section the results of these games will be given. In all games the black player plays first.

First the results of the encounters between the agents with basic and extended heuristics will be given. See Table 3. Note that the agents both play with a search depth of 2. A reason for this shallow depth will be given in the 'Further Research' section. The column 'Winner' indicates the winning agent, or (by mentioning 'Draw') that the game ended since none of the two agents is able to generate a new move and the game therefore enters a loop. These results show that the extended heuristics

Game	Black Player	White Player	Score	Winner
1	Basic	Extended	0-1	Draw
2	Extended	Basic	6-0	Extended

Table 3: Results of games between constructed agents with basic and extended heuristics. 'Score' means the number of captured opponent marbles (an agent needs six marbles to win the game).

are valuable. The extended player plays more agressively than the basic one. It sometimes also plays more risky than the basic player in order to force a breakthrough.

The results against the (commercial) implementations will be given next. The implementations the agent played against are:

- Random Soft Abalone (RandomAba)
- Ali Amadi and Ihsan Abalone (AliAba)
- NetAbalone

It should be noted that the 'Random Soft' implementation does not support broadside moves and thus these are also not available for the extended agent. RandomAba plays at different difficulty levels. Both medium and high difficulty are tested. AliAba has just one difficulty level. NetAbalone has ten different difficulty levels but only one is available in the freeware version of the game. It should be stressed that the extended agent plays only at search depth 2. The results in table 4 show that the

Game	Black Player	Difficulty	White Player	Difficulty	Score	Winner
1	RandomAba	M	Extended	depth 2	0-1	Draw
2	Extended	depth 2	RandomAba	M	0-2	Draw
3	RandomAba	H	Extended	depth 2	2-1	Draw
4	Extended	depth 2	RandomAba	H	0-2	Draw
5	AliAba	NA	Extended	depth 2	2-2	Draw
6	Extended	depth 2	AliAba	NA	6-1	Extended
7	NetAbalone	1	Extended	depth 2	0-0	Draw
8	Extended	depth 2	NetAbalone	1	0-0	Draw

Table 4: Results of games between extended heuristics agent and third-party (commercial) implementations. Again 'Score' means the number of captured opponent marbles (an agent needs six marbles to win the game).

extended agent never loses. It either wins or draws (because of the inability of both players to come up with new moves). As can be seen in figures 12, 13 and 14, where some game positions from drawn games are presented, the extended player has managed to split up the opponent and resides itself in the middle. The score is in favour of the opponent because of the inability of the extended agent to recognize dangerous situations early on. It therefore waits too long with taking counteractions. The same occurs in the Extended - NetAbalone game but the other way around. The moment the game enters the 'repeated-move' loop the score is 0-0. In this game NetAbalone accomplished to break up the extended agent but lacked the ability to push off any marbles. In the NetAbalone - Extended game both players were equally strong.

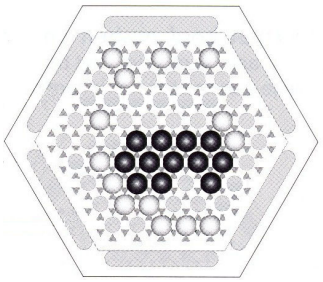

Figure 12: End situation in Extended - RandomSoft M. Black is Extended, White is RandomAba M.

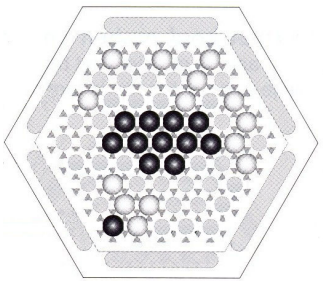

Figure 13: End situation in Extended - RandomSoft H. Black is Extended, White is RandomAba H.

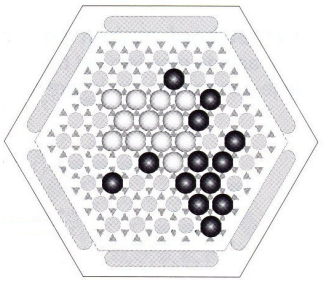

Figure 14: End situation in RandomSoft H - Extended. Black is RandomSoft H, White is Extended.

Table 5 shows the moves of a test game between

NetAbalone and the extended agent. The game ended in a 0-0 draw. This explains the small number of moves that are made.

Move Nr.	Move	Move Nr.	Move
1	a5b5	25	b1b2
2	i9h8	26	f4e4
3	b5c5	27	b3c3
4	i8h7	28	f3f4
5	a4b4	29	a4b5
6	h9g8	30	g8f7
7	b4c4	31	b4c5
8	i7h7	32	h8g8
9	b3c4	33	a1b1
10	i6h5	34	g8h8
11	a2a3b3	35	b1b2
12	h4g4	36	h8g8
13	b6b5	37	b5c6
14	i5h4	38	g8h8
15	b4c5	39	b2c2
16	h7g6	40	h8g8
17	e6e5	41	b3b2
18	h4h5	42	g8h8
19	b3c4	43	b2c2
20	h7g6	44	h8g8
21	c4d5	45	e2d2
22	h9h8	46	g8h8
23	f7e6	47	b2c2
24	g4f4	48	h8g8

Table 5: Moves of a test game between NetAbalone and the extended agent. NetAbalone plays the first move.

7 Conclusions

As can be seen in the previous section the extended agent performs quite good. In this paper no implementation is able to win from the extended agent. As the game lengthens it shows that the extended agent nicely breaks up its opponent. The opponent is scattered alongside the edges of the board. The agent just lacks the ability to foresee very bad and good situations, where its own marbles are in danger near the edge of the board and where the opponent's marbles lie helplessly at the edge of the board, respectively.

NetAbalone plays quite the same as the extended agent. It also tries to split the strong (i.e., three marbles) groups by pushing out the center but is a little bit more aggressive in that it tries harder to get and keep the center. Both NetAbalone and RandomSoft Abalone probably search deeper in the tree than the constructed agent does (the extended agent only searches at depth 2). NetAbalone further has probably more fine-tuned weights for its strategies and therefore performs better than the other implementations.

Summarizing the results: For an agent playing with only a search depth of 2, the agent performs quite good. Due to the search depth it does not always choose the best possible move as it does not always detect that marbles are in danger. It just evaluates the situation and picks the best move for that situation at that moment without looking very far in the future. Even though the agent has this disadvantage it wins/draws, in this paper, against every other computer agent found.

An interesting point to note is the game length. While humans play reasonably 'fast' (e.g., on average 87 moves per game) the games between computer players typically last longer (e.g., on average 130 moves per game). A reason for this can be found in the fact that humans are able to play more to the point than computer players do. Computer players tend to be more conservative while human players are more progressive.

8 Future Research

Due to the generation of valid moves and the time this takes, the extended Abalone agent can get no further than two deep in the search tree. Searching deeper in the tree results in a severe time penalty. By searching deeper in the tree the agent will probably detect better moves because it can more exactly predict the outcome of a certain move. As it is to be expected that searching deeper in the tree will result in better results it is therefore recommended to optimize the generation of valid moves.

Implementing more optimalizations of the Alpha-Beta algorithm could also further reduce search time and deepen the search. 'Iterative deepening' could be one of these optimalizations.

The move ordering could be further extended. This could be done by applying some additional heuristics, like the killer-move heuristic [2] or the history heuristic [10].

The strategy weights for the agent are very rough. It could increase performance if these weights could be fine-tuned. Machine-learning techniques could help in the process.

Pattern-recognition techniques could improve the 'human' view of the game, e.g. trapeziums, diamonds and daisy-forms (pattern of 6 own marbles and in the middle an opponent marble) are strong groups. It could further speed up optimal move searching as interrupting strong groups (i.e., pushing out the middle marble of a strong group) is a strong strategy.

As Abalone theory matures it should be possible to construct an opening book. The first moves of the game are important for the conquering (or first reaching) of the center. As these moves do not directly involve the opponent it saves time to have them already in the opening book.

References

[1] Abalone S.A. (1989-2002). Abalone official site. http://uk.abalonegames.com/.

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

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

[4] International, Jumbo (1989). *Abalone*. Jumbo International, P.O. Box 1729, 1000 BS, Amsterdam.

[5] Lockergnome LLC. (2004). Game tree complexity. http://encyclopedia.lockergnome.com/Game_tree_complexity-sb.html.

[6] Moreland, B. (2001). Alpha-beta search. http://www.seanet.com/~brucemo/topics/alphabeta.htm.

[7] Ozcan, E. and Hulagu, B. (2004). A simple intelligent agent for playing abalone game: Abla. *Proc. of the 13th Turkish Symposium on Artificial Intelligence and Neural Networks*, pp. 281–290.

[8] Rognlie, R. (1996). Richard's play-by-email server. http://www.gamerz.net/pbmserv/.

[9] Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ.

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

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