Abalone*

Stephen Friedman and **Beltran Ibarra**

Dept of Computer Science and Engineering
University of Washington
Seattle, WA-98195
{sfriedma,bida}@cs.washington.edu

Abstract

In this paper we explore applying the technique of Alpha-Beta pruned MinMax search to the board game Abalone. We find that Alpha-Beta pruning can give you significant processing savings. We also present several heuristics for evaluating non-terminal board positions and examine their effectiveness when used by a depth limited search algorithm to play the game of Abalone.

Motivation

Board games have always created a lot of interested in every human community. Their simplicity in terms of rules but complexity in games possibilities are one of the key features of their success. Moreover, these games have always been associated to strategy and war, which has been up to now (and sometimes still is) one of peoples favorites hobbies.

For half a century, these board games have been the benchmark of computer scientists working in artificial intelligence. Games like Chess, Othello, GO have all been studied and had computational implementations created, but not always with great success. While many of these game playing programs are highly tailored to the individual game, some general purpose techniques have been developed. It is these general purpose algorithms that we are interested in applying to the relatively new board game of Abalone.

In the current state of the world, an Abalone playing program by the name of ABA-PRO is the current world champion. Unfortunately there is little information available on how it actually works (Oswin Aichholzer 2002), or what heuristics it uses. As such, we also looked at the process of tailoring a computer game player to a game by investigating multiple heuristic evaluation functions.

Abalone Rules

Abalone was first created in France some 20 years ago. Like many interesting board games, it is very simple to learn and amazingly complex to play. It is now played all over the world and is considered a to be a classic board game, among the ranks of Chess or GO.

The rules are very basic. We will give a simple overview here, but for the finer points the reader is directed to more thorough online references such as the Wikipedia Abalone game entry (WIKIPEDIA 2004) or the official Abalone web site(AbaloneS.A. 2004). There is an hexagonal board, 5 spaces to a side, with white and black balls. Each player has 14 balls of one colour (black or white).

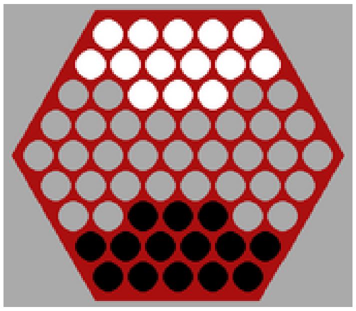


Figure 1: Initial board(WIKIPEDIA 2004)

The aim of the game is to remove 6 of the opponent's balls by pushing them out of the board. At each turn one player can move up to three inline and adjacent balls to the next free space. If moving multiple balls, they must all move in the same direction. When moving multiple balls, we call moves to spaces in line with the set of balls and in-line move, and moves to spaces adjacent to the set of balls broadside moves. If there is an opponent's ball occupying the space a player wishes to move to, they may try to push the opponents balls. In order to push an opponents pieces, the number of pieces in the players line of balls must outnumber the number of pieces in the opponents line. Thus two player pieces may push one opponent piece, and three player pieces may push one or two opponent pieces. Any balls pushed out of the board are lost for the remainder of the game.

---

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

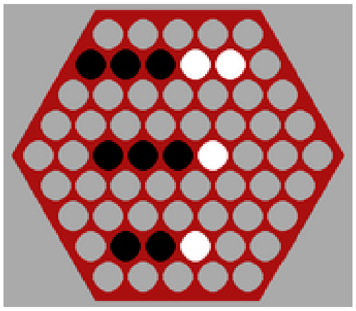


Figure 2: Before pushing(WIKIPEDIA 2004)

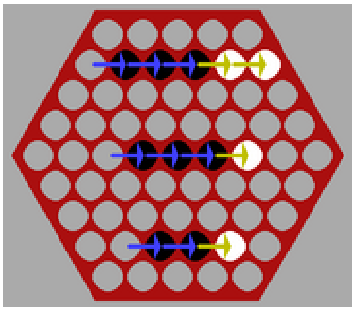


Figure 3: After pushing(WIKIPEDIA 2004)

In our version of the Abalone game, we are going to forbid the broadside moves in order to simplify the game. By doing this, we will reduce the number of possible moves and thus we will reduce the breadth of the search tree.

Solution : Name of our Solution

Choosing a Move

By defining our search space as the space of possible game boards, it is easy to search through that space generating child nodes using the rules for legal moves. Using this search space, we can apply the Min-Max algorithm to choose the best next move. Unfortunately, Abalone has a very large branching factor which prohibits searching the tree all the way to the endgame states. Instead, we terminate the search at a pre-set depth and apply a heuristic evaluation function the board state at that depth. To further reduce the search space, we apply the technique of Alpha-Beta pruning. Below we describe the individual heuristics we implemented.

Implemented Heuristics

Heuristics are functions that help the computer to choose the best moves based on an incomplete set of information. Using these heuristics, the computer can choose the next move without forcing the Min-Max algorithm to search the entire tree through to the end. Instead, we can evaluate the board value at a non-terminal depth and compare these to choose the best. Contrary to Min-Max and Alpha-Beta pruning, heuristics are specific to the particular game they are developed for and so cannot necessarily be directly applied to other games.

As they are so specific to the game, these heuristics can define the way a computer "plays". In our case, when designing the heuristics, it helped us to think in terms of strategies we felt were valuable in playing the game. For example, you can set a heuristic which may play a defensive game by increasing the evaluation score based on grouping, or on the hand have a very aggressive game by increasing the score for pushing off the other players pieces.

For the Abalone game we have developed four main heuristics. Each one of them is based on game experience that we have had and thus try to imitate what a human player would (very often unconsciously) do. Of course these are not the only heuristics that exist and one can always come up with some new function that will completely change the game play.

First Heuristic : Gravity Center The first heuristic is based in the fact that being in the center is safer than being near the borders. There are two reasons for this. The first one is that when the pieces are near the center of the board, they are further away from the borders (obviously) and so they are in less danger of being pushed off. The second reason is that when the pieces are in the center, they are in a pack and so there are more likely to be in rows of three, a position in which they cannot be pushed.

This heuristic was implemented by first assigning a value to each of the board's fields. This value is calculated in function of its distance to the center. The further from the center the lower the score it gets. Then we just have to add all the scores of the positions occupied by one players pieces and subtract all of the opponents. This maps directly to use with the Min-Max algorithm since one player will try to get the highest values and the other one will go for the lowest values. As this was the original, heuristic that came with the codebase we are building upon, it is the one that we pitted our solutions against as a base metric.

Second Heuristic : Three In a Row The second heuristic awards points based on having up to three balls in a row. This reflects the fact that it is beneficial to have up to three in a row, but there is very little strategic benefit to having four or more in a row. When we have three in a row, we can push the maximum number of opponents pieces, but cannot be pushed ourselves.

To encourage groupings of three in a row, the board is scored as follows. For each of the 3 line directions possible on a hexagonal board, the algorithm searches through the corresponding rows. When it sees two balls in a row, it adds or subtracts one point based on whether it is the min players pieces or the max players pieces. Similarly, it adds or subtracts two points when it sees three balls in a row. It is a highly defensive heuristic, as the previous one, but one is never too careful!!

Third Heuristic : Keep Packed The third heuristic is also a defensive one. It is based in the fact that wherever the pieces are, it always better to have them grouped than all scattered among the board. Like in the first heuristic, a big group of balls will certainly be harder to beat than small groups. It is more likely to have rows of three in several directions when the balls are packed than when just a few pieces are together.

We have implemented this heuristic by going all along the board and whenever we find a ball, we count the number of neighbours of the same colour and add that number to a counter. Then we do the same with the other colour only this time we subtract. Thus one has to maximize and the other has to minimize, which is what we want for the Min-Max. This way of counting seems very simplistic since we count the same neighboring pieces several times, but in fact it is very interesting since the scores will rise exponentially with the number of neighboring pieces. The more packed they are, the better score they'll get. This is to emphasize that it is better to have one big pack than two medium packs, for example.

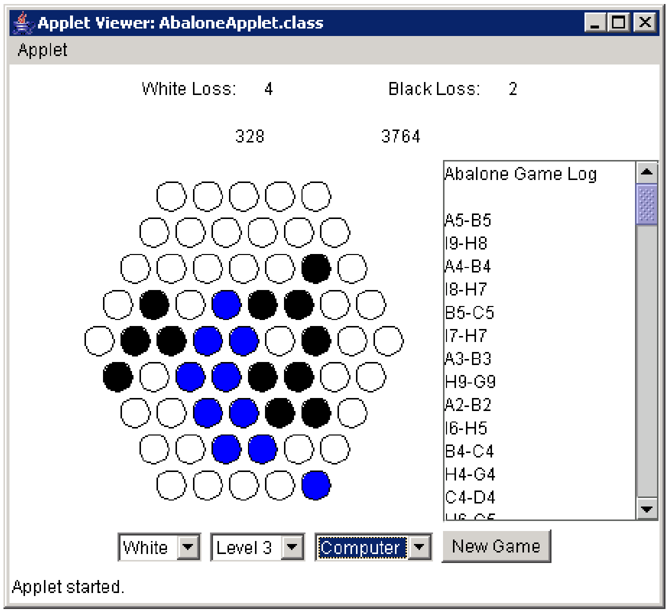
Fourth Heuristic : Let's Kill'em The fourth heuristic is used to counterbalance this rather defensive set of heuristics. It aims to attack whenever it is possible and whenever it will not represent a danger in the next move. It is worthless to push out a ball, if in the next move the opponent can do the same to you. This kind of situation occurs very often in the game. For this heuristic to be effective, it is necessary to explore at least one opponent move further.

We implemented this aggressive heuristic by basing the score on how many pieces have been thrown out. We calculate a score that has an exponential relationship to the number of balls thrown out. Then we calculate the difference between each opponent. Let's look at the situation where two white balls have been pushed out, three black balls have been pushed out, and black has a chance to push out a white ball. If the ball is pushed out, the differencing will make the heuristic score null again, representing an even match. If the ball is not pushed, the large white score will remain and it will severely reduce blacks heuristic score. One should not forget that the aim of the game is to remove six of the opponents balls and not to survive forever, so it may be in one's interest to sacrifice a piece if it means it is possible to win in the next ply.

	H1 Weight	H2 Weight	H3 Weight	H4 Weight
Run1	1	0	0	0
Run1	1	1	0	0
Run2	1	0	1	0
Run3	1	0	0	1
Run4	0	1	0	0
Run5	0	0	1	0
Run6	0	0	0	1
Run7	1	1	1	1
Run8	0	1	1	1
Run9	0	1	0	1
Run10	0	1	1	0
Run11	0	0	1	1
Run12	1	0	0	0

Table 1: Experimental Run Configurations

Experiments



White Loss: 4 Black Loss: 2

328 3764

```
Abalone Game Log

A5-B5
I9-H8
A4-B4
I8-H7
B5-C5
I7-H7
A3-B3
H9-G9
A2-B2
I6-H5
B4-C4
H4-G4
C4-D4
```

Applet started.

Figure 4: Abalone Experiment Applet

We wanted to test two things in this experiment. First, we wanted to show that Alpha-Beta pruning provided significant speedup to move searching, allowing a deeper search. Second, we wanted to show that through careful choosing of board evaluation functions, one can play a better game of abalone without having to search all the way to the endgame condition. To facilitate this, we started with a Abalone playing applet by Frank Bergmann and enhances it with Computer vs. Computer play and scoring, move timing, move counting, and move logging capabilities. The UI for the modified applet can be seen in Figure 4.

To show the speedup benefits of Alpha-Beta pruning, we set up two computer opponents, one using standard Min-Max search, and the other enhances with Alpha-Beta pruning. We then setup time counters around the move search functions for each computer player. These timers are rather coarse (millisecond accuracy) and because it was not strict process accounting, other tasks running simultaneously could affect our measured time of the computer players. In order to compensate for this, we ran the program and accumulated the first 30 moves worth of time for a game for each player, and repeated these games 5 times to get an average of the time spent calculating these moves. While it is possible to prove that an Alpha-Beta pruned searches a tree the same size or smaller than pure Min-Max for a given depth, we wanted to demonstrate that the added overhead associated with the Alpha-Beta pruning was outweighed by the speedup gained from the reduction of the searched tree.

To demonstrate the benefits of careful selection of non-terminal board evaluation heuristics, we pitted computer players with differently weighted evaluation heuristics against each other. We gave the advanced heuristics to the Alpha-Beta search player. We set the search depths to be the same on both the traditional Min-Max and Alpha-Beta players, and varied the weights on the heuristics for the Alpha-Beta player. We then recorded wins, losses, and piece counts for tied games. We tried the experiments with the weights listed in Table 1. HX refers to the heuristic evaluation functions described in the Implemented Heuristics section, and they are numbered according to the order of appearance.

Upon running these experiments on the first 6 runs, we realized we were obtaining inconclusive results. Noticing that the computer players would often get stuck in cycles of moves, we added a bit of randomness to our Alpha-Beta player. We let it randomly pick between boards with equivalent heuristic evaluation scores. We also stacked the cards in favor of the Alpha-Beta player, setting the Min-Max recursion depth to 1 ply and the Alpha-Beta recursion depth to 3 plys.

Experimental Results

Upon running our first experiment, we saw a significant improvement in calculation time when compared to the Min-Max algorithm running at the same recursion depth with the same heuristic evaluation function. The results are given in Table 2 show the evaluation time per move of each algorithm at a recursion depth of 3 plys averaged over 51 moves each.

After running the experiments, we obtained the results given in Table 3. As you can see, the results were not very enlightening when it comes to telling whether or not the heuristics we wrote improved play. In all of these games, the two computer opponents get stuck in an endless loop of moves. This often happens in the early stages of the game so we cannot assume that the average times are meaningful when compared to full games. This is due to the fact that at the beginning, all of the pieces are tightly packed at either side, so they can only move forward. This severely limits the breadth of the tree and decreases the effectiveness of the Alpha-Beta pruning.

Level 3 Min-Max	404ms per move
Level 3 Alpha-Beta	170ms per move

Table 2: Min-Max vs. Alpha-Beta Time Comparison

	Black Score	Average Black Time	White Score
Run 1	1	12ms	2
Run 2	0	8ms	0
Run 3	0	15ms	0
Run 4	0	15ms	0
Run 5	0	5ms	0
Run 6	0	under 1ms	0
Run 7	1	12ms	4

Table 3: Initial Experimental Run Results

Due to the lack of useful results in the fist set of experiments as can be seen in Table 3, we modified and re-ran it, as described in the Experiments Section. Table 4 shows the results of the modified experiment.

Conclusions

It is interesting to note that any set of heuristics that included the first one did quite well. When looking at the last 3 heuristics alone, H2 was the only one able to win by itself, and even then only twice. It also seems that having H1 and H2 together actually strengthen the play, as they win by a greater margin than H1 paired with either H3 or H4. In fact, the performance of H1 combined with H3 or H4 is not noticeably different from the performance of H1 alone. This could be a hint about combining heuristics to build stronger players. It could also mean that the weights are not appropriate. Having all the heuristics together isn't necessarily a good thing, as is shown by runs 7 and 11. We don't always get better results, but we will definitely slow down the computation by combining these heuristics.

It is not that easy, or at least not intuitive, to design good heuristic functions. This is especially apparent when you look at the results of Run 11, where the computer did quite well playing second, but lousy when it played first. Because the other runs didn't show this similar trend, we can be fairly comfortable in assuming that it is due to the combinations of heuristics used, and that starting first doesn't automatically put one at a disadvantage.

Run 1		Run 2		Run 3	
Black	White	Black	White	Black	White
6-0	0-6	6-5	5-6	6-3	5-6
6-0	0-6	6-0	5-6	6-3	2-6
6-2	0-6	6-2	2-6	6-0	5-6
6-3	0-6	6-4	5-6	6-3	4-6
6-0	0-6	6-2	5-6	5-6	4-6
6-0	0-6	6-5	4-6	5-6	5-6
6-0	0-6	6-2	5-6	6-3	5-6
6-2	0-6	6-3	4-6	6-4	5-6

Run 4		Run 5		Run 6	
Black	White	Black	White	Black	White
6-5	5-6	stuck	6-1	3-6	6-1
stuck	6-0	stuck	6-1	1-6	6-2
0-6	6-0	stuck	6-1	2-6	6-1
0-6	6-0	1-6	6-1	1-6	6-1
stuck	6-0	1-6	6-0	1-6	6-1
stuck	6-1	stuck	6-1	0-6	6-1
stuck	6-2	stuck	6-1	0-6	6-0

Run 7		Run 8		Run 9	
Black	White	Black	White	Black	White
6-3	5-6	stuck	stuck	1-6	4-6
stuck	5-6	6-3	5-6	stuck	6-1
stuck	5-6	stuck	stuck	6-4	5-6
6-3	5-6	6-5	5-6	6-5	5-6
6-3	2-6	6-4	5-6	0-6	6-0
6-4	1-6	6-4	4-6	6-5	4-6
6-1	2-6	6-4	stuck	6-4	6-2

Run 10		Run 11		Run 12	
Black	White	Black	White	Black	White
6-3	stuck	6-5	6-5	6-5	4-6
6-5	6-0	6-4	6-0	6-5	2-6
6-5	5-6	6-4	6-0	6-3	3-6
3-6	4-6	stuck	6-1	6-4	2-6
3-6	6-2	stuck	6-1	5-6	4-6
stuck	6-0	0-6	stuck	6-3	4-6
5-6	stuck	stuck	4-6	6-5	6-5

Table 4: Experimental Run Results with Modifications - Scores are given in B-W format. The colour at the head of the column indicates that played by the Alpha-Beta algorithm.

We noticed that for a 2 ply depth search, the Min-Max implementation was sometimes faster than the Alpha-Beta implementation. Because the Alpha-Beta implementation requires more computation at each node than Min-Max, in trees where there is not a lot of benefit to pruning, it is expected that Alpha-Beta may take longer. In short trees, you only get the opportunity to prune short trees and leaf nodes. The real advantage comes in deep trees, where you can prune whole subtrees near the root.

The game length was quite variable, from 60 to 600 moves. In the longer games, many of these were cyclic moves. This is most likely due to the fact that it would take the first good move it saw with a very high probability, and we only included the randomness to get it ”un-stuck” in these cyclic situations.

During the course of our experiments, we noticed that on several occasions with higher depth search trees, the computer player we created would fail to take obvious and immediate moves that would allow it to win the game. We believe this is because of the depth-first search nature. If it finds a winning board position three ply's down before it finds the winning position one ply down, it will simply accept the sequence with the three ply win first. So sometimes it is better to think only one good move in advance as opposed to three moves.

Suggestions for Future Research

The obvious extensions to the research would be to investigate more and varied heuristics and their interacting effects. These should be developed with more precise testing/timing procedures and better movement and strategic effect analysis. Also, more variety in the assignment of weights to the heuristics may reveal better balances than our simple all-or-nothing approach. Of course this method is a lot of tedious work. In order to capture the short-term obvious win moves with large depth search trees, an iterative deepening approach can be attempted. In addition, if leaf nodes at one depth are re-ordered before proceeding to the next depth, it may be possible to get near the optimal ordering for Alpha-Beta pruning. It has been shown that the repeated effort in iterative deepening search doesn't add a prohibitive amount of computing time.(Russell & Norvig 2003) The way we set up the evaluation function, the heuristics may be combined in a variety of ways. One further investigation would be to find closer to optimal weightings of the heuristics using machine learning techniques. Extending this idea, one could experiment with on-line machine learning to allow the program to adapt as it plays. It could, for example, use more defensive strategies whenever the amount of pushed out pieces is similar, but go on the offensive when it is about to win. We have also noticed that it is quite easy for the computer players to get stuck. This is also appears to be a problem for people, and an interesting avenue of research would be to explore more starting positions that prevent defensive stalemates, as suggested by the

Wikipedia entry(WIKIPEDIA 2004).

In our implementation, setting the Level that the computer plays at via the GUI is equivalent to setting the search depth in plys. Something that may be investigated in the future is whether or not it is better to look ahead and end your search on a player ply or an opponent ply. One can imagine that for a heuristic such as the one that awards points for pushing a ball off the board, it may be better to look ahead to an opponent ply, so that we don't greedily push a ball of, only to lose one immediately on the next turn.

Acknowledgements

We would like to thank Frank Bergmann for the use of his Abalone applet as the base for our experimental system. Acknowledgements go to Artem Zhurida for reinforcing the idea that we should try to add randomness to the movements due to Alpha-Beta being highly sensitive to move ordering. We would also like to thank Steve Balensiefer for his constant chiding, prodding, and puzzlement at our bugs.

References

AbaloneS.A. 2004. Abalone official web site. Web Article. http://uk.abalonegames.com/.

Oswin Aichholzer, Franz Aurenhammer, T. W. 2002. Algorithmic fun - abalone. *Special Issue on Foundations of Information Processing of TELEMATIK* 1:4-6. http://www.igi.tugraz.at/oaich/abalone.html.

Ozcan, E., and Hulagu, B. 2004. A simple intelligent agent for playing abalone game: Abla. *TAINN.* http://cse.yeditepe.edu.tr/∼eozcan/research/ papers/ABLA_id136final.pdf.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition.

WIKIPEDIA. 2004. Abalone game. Web Article. "http://en.wikipedia.org/wiki/Abalone_game".

Appendix A - Contributions

We received a basic Abalone Java Applet GUI complete with a computerized MinMax opponent from Frank Bergmann. On top of this we added an Alpha-Beta search capable opponent. We also implemented 3 new heuristic evaluation functions. For facilitating experiments, we modified the code so that computer opponents could play one another, and we added lost pieces, move counter, and time spent statistics to the UI. We also implemented a game logging feature that records each game in side text from in the format described by the Abalone Wikipedia entry(WIKIPEDIA 2004).

Stephen Friedman's contributions to the project included research into other works, implementing Alpha-Beta pruning, implementation of the Gravity Center and creation/implementation of the Three in a Row heuristics, GUI layout, computer vs. computer mode, scoring, timing, and large contributions to writing of the report.

Beltran Ibarra's contributions to the project include research into other works, setting up the initial LaTeX outline, implementation of the Keep Packed and creation/implementation of the Let's Kill'em heuristics, implementation of movement logging, movement counting, executing and recording test runs, and large contributions to the writing of the report.