A Simple Intelligent Agent for Playing Abalone Game: ABLA

Ender Ozcan and Berk Hulagu

Department of Computer Engineering
Yeditepe University
Istanbul, 34755, Turkey

Abstract. Forming winning strategies for board games requires good heuristics and fast search algorithms on game trees. High branching factors and the need for looking deeper in game trees are overwhelming, even for today's high performance PC's. Therefore, better game-plays are only available with better algorithms and heuristics for an ordinary player, not with faster machines. Abalone is a recent two-person strategy game. Initial evaluations point out that the branching factor is larger than the chess. In this paper, a new heuristic used by a simple intelligent agent for playing Abalone game, named as ABLA is introduced. ABLA's performance is promising as compared to existing computerized Abalone players.

1 Introduction

Board games have always been a popular research area for a wide range of communities. It is itself a challenge to write computer programs for game playing that can challenge human players. Considering the advances in the game playing techniques, there are still interesting domains in board games for research, such as, Chess and Go. There are also some scientists studying Xiang qi (Chinese chess) and Shogi (Japanese chess). Common property of all these games is that each one requires an enormous space to be searched with a high decision complexity.

One of the first attempts to mechanize chess turned out to be one of the greatest illusions in the history. The automaton for chess was first constructed in 1769 in Vienna by Baron Wolfgang von Kempelen and named as *The Turk*. This automaton was nothing more than a hidden chess expert in a cabinet. The interesting history of The Turk can be found in [11] by Standage. Studies led to one of the most popular competitions between a chess master Gary Kasparov, and "Deep Blue". Deep Blue has defeated Gary Kasparov after a game of six series in 1997 ([14]).

The games Xiang qi and Shogi have their differences and similarities with chess (Matsubara *et. al.* in [5]). Presumably, the strategies used in Chess can be fine tuned and used in these games as well, leaving Go to be the main focus of game playing researchers. In [6] Müller and in [12] Takizawa *et. al.* provide the most recent approaches for playing Go and provide some future directions. One of the latest two-person, zero-sum board games is Abalone. Since 1990, the game is gaining popularity progressively more. Abalone is a perfect information game similar to Chess and Go.

In this paper, we introduce a new heuristic embedded into an intelligent agent for playing Abalone game. In the next section, traditional approaches to solve zero-sum games are overviewed. In section 3, Abalone game is described. In section 4, components of the intelligent agent are explained and its performance is evaluated. Finally, conclusions are provided in section 5.

2 Playing a Zero-Sum Game

Board games are generally *zero-sum* type of games. Zero-sum indicates that a participant can only gain at the expense of other participant. There are many examples of such *situations* other than games, referred as zero-sum "games". The main reason even such situations are referred as "games" is that this concept was first introduced in the game theory.

A two player, turn-taking, zero-sum board game with perfect information can be formulized as a search problem, requiring the best move (decision) to be determined at a certain state. At any moment during such a game, one should consider all possible outcomes of the moves that can be made. To evaluate all possible outcomes, one should also be able to think for the *opponent* and attempt to figure out the possible outcomes due to the opponent's moves as well. One players gain is others loss.

Most of the state of the art intelligent agents for playing a zero-sum games perform a search on a *tree* for the best move. Hence, an intelligent agent for playing such a game requires a representation scheme for identifying a state of the game, corresponding to the turn and the arrangement of the board. *Game tree* in this case is an explicit representation of all possible plays of the game. Game tree of tic-tac-toe is illustrated in Figure 1. The *root node* is the current position of the game and its successors are the positions that the first player reach in a single move, their successors are the positions resulting from the opponent's replies. A *successor function*, denoted as *S*, is used to generate a list of *valid* moves, given a turn and an arrangement of the board. One of these valid moves should be selected based on a *utility function*. Each generated state might be the end of the game which should also be tested. *Terminal* (or *leaf*) *nodes* are those representing WIN, LOSS or DRAW. A simple utility function evaluates a WIN by +1, LOSS by -1 and, DRAW by 0.

Given a zero sum board game with perfect information, starting from the initial state whole game tree can be built using the successor function. Analyzing the terminal nodes, a decision can be made among possibilities leading to WIN situation. This process can be repeated after each move of the opponent, building up towards a winning strategy. However, in most of the games, constructing whole the game tree is infeasible, due to the fact that the average number of possible moves for a game state referred as *branching factor*, denoted as *b* and the depth of the terminal nodes are at very deep levels.

A complete game of checkers has about 10^{40} non-terminal nodes as provided by Samuel *et. al*. in [8]. Based on an average branching factor and average game length (depth of the terminal nodes), chess produces a game tree complexity of 10^{123}, and similarly, Go produces 10^{360} as reported by Allis in [2]. Game tree complexity and decision complexity of Abalone is still open for research. An intelligent agent has to make a choice and decide on a move in a reasonable amount of time, limiting the depth of the game tree, managing both time and memory issues as required.

After limiting the depth of the game tree and having a finite search space, most of the times it is impossible to claim which nodes are WIN, LOSS and DRAW nodes, but somehow, the leaf nodes must be evaluated to decide whether they are advantageous or disadvantageous for a particular player. For each game, there are different types of elements affecting the games' state, such as positions of marbles, gaining extra rights during the game, pieces thrown away by opponent etc. Combining these elements, an assumption can be made determining "how good the position is for a player". In other words, utility function becomes a *heuristic function*, denoted as *eval*. Then after evaluating the leaf nodes, the move leading to the most advantageous state, which will have the highest evaluation score, is selected. This process can be repeated whenever it is the agent's turn.

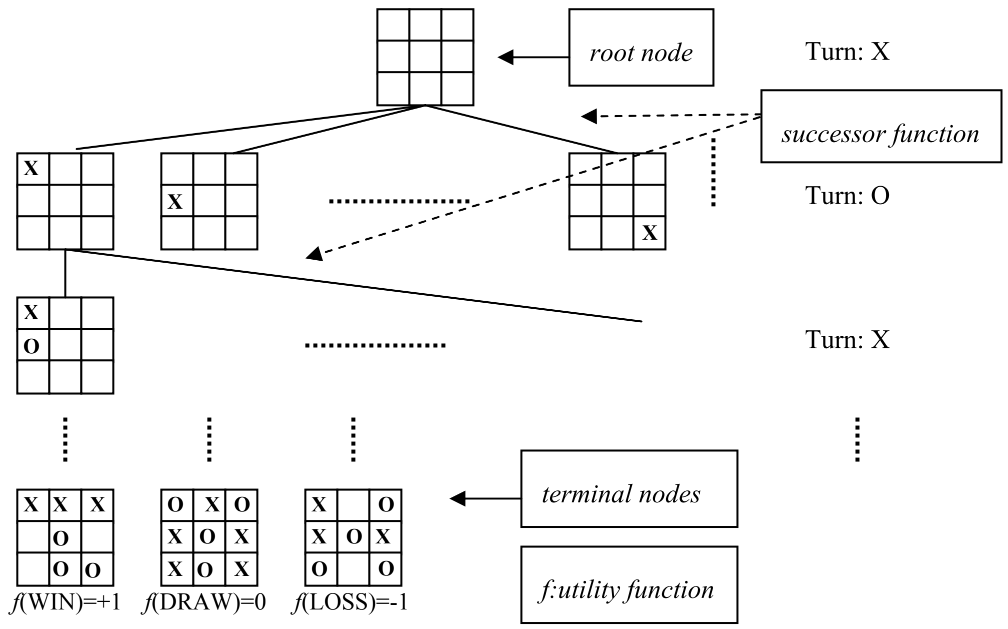


Figure 1. A part of the search tree for tic-tac-toe starting a new game with X's turn.

2.1 The MINIMAX Algorithm and Pruning

The first search algorithm that is used on game trees' is MINIMAX algorithm, introduced by Shannon in [9]. The algorithm performs a depth first search on the game tree generated up until a certain depth. Each node contains an additional value called *minimax* value. Each level of the game tree where the turn belongs to the agent is marked as MAX level and each level where the turn belongs to the opponent is marked as MIN level, starting from the root. Minimax values are calculated as follows:

$$minimax\_value(N) = \begin{cases} eval(N) & if\ N\ is\ a\ leaf\ node \\ \max_{n \in S(N)} \{minimax\_value(n)\} & if\ N\ is\ a\ MAX\ node \\ \min_{n \in S(N)} \{minimax\_value(n)\} & if\ N\ is\ a\ MIN\ node \end{cases}$$

Minimax value of a leaf node is set by the evaluation function. All the rest of the internal nodes receive a *backed up* minimax value via the unfolding recursion.

Alpha-Beta Pruning is introduced by Edwards *et. al.* in [3], improving on MINIMAX algorithm. It is a technique that prunes the nodes that are impossible to reach from the current position of the board by ignoring branches on the game tree having no further contribution on the outcome. The tree construction and search process overlaps, reducing the computation time required.

Slagle *et. al.* in [10] showed that the number of terminal nodes examined by alpha-beta algorithm must be at least $b^{|d/2|} + b^{|d/2|}$, but b^d in the worst case, where d is the bounded depth. However, Knuth *et. al.* in [4] proved that the terminal nodes can be arranged, so that the worst case will be $b^{|d/2|} + b^{|d/2|} - 1$. However, to manage the terminals, the whole search tree is needed to be constructed, as mentioned earlier which is not preferable and infeasible anyway in most of the games.

3 Abalone

Two-player, strategy game Abalone was invented in 1990 by Laurent Levi and Michel Lalet. In the official web site ([13]), Abalone is introduced as "never alone" game, since the prefix "ab" means "never". Hence, it is suggested that its own name summarizes the main strategy required: "winning against loneliness". The concept of the game is based on the popular Japanese Sumo wrestling.

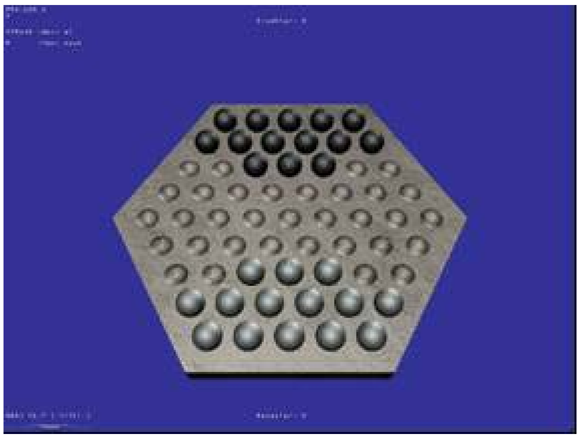
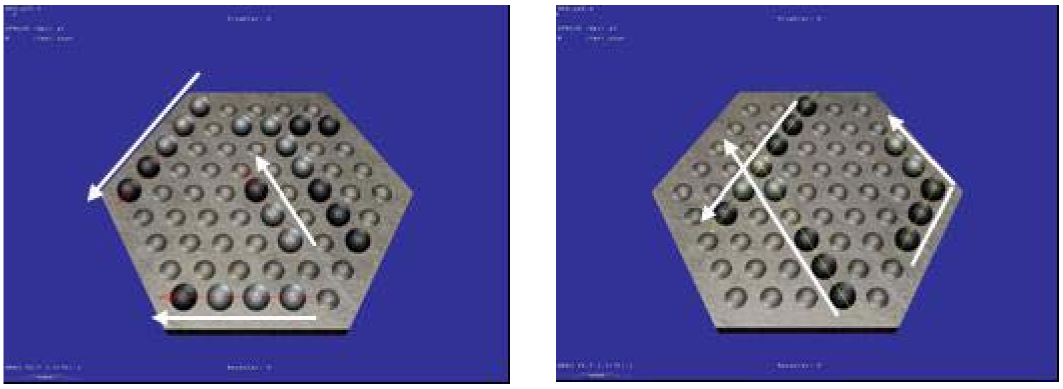


Figure 2. Initial game position of Abalone.

The board is a hexagonal field representing the Sumo arena. There are 4 interlaced different size hexagons, where the most exterior hexagon has 5 circular locations at each side. In the middle there is a single location. As a total, there are 61 circular board locations. Each player has fourteen marbles that can rest in these locations. Initial game position for players is illustrated in Figure 2. Each color (black-white, or black-gray) represents a Sumo-fighter.

3.1 Rules of the Game

Starting from the initial configuration, each player takes a turn. During a turn, a player can shift one, two, or three marbles together in any of the six directions, provided that there is an adjacent space, i.e., in line or broadside. Furthermore, whenever a player has a numerical superiority in a line (three to two, three to one, or two to one), during a turn the player is allowed to push the opposing marbles with an inline move as demonstrated in Figure 3, even off the board. No broadside pushes are allowed. The objective of the game is to push six opposing marbles off the edges of the board.



(a) **(b)**

Figure 3. Examples showing legal and illegal moves: **(a)** Sequence of gray marbles that has arrow on them is allowed to push the set of black marbles in front of them. **(b)** None of the set of marbles may push the opposite side's marbles

Rules are very simple, but numerous strategic moving, pushing and defending possibilities make the game complex. Note that, there are some variants of this game that we do not consider in this paper.

3.2 Winning Strategies

To reach the objective, pushing six marbles of the opponent off the board and at the same time protecting one's own marbles from being pushed off, some strategies have already been suggested. Following strategies are collected from various forums of Abalone communities ([13], [17]), reported as winning strategies:

- The amount of “2 or 3 in a row” marbles makes one’s army’s defense and offence powerful.
- Creating a hexagon of marbles among the opponents’ marbles, allows one’s army to push or provide defense in all directions.
- Moving a marble to a location where there will be no teammate neighbour is not a good strategy.
- Forcing the opponent to keep its marbles at the edges of the board provides potential scores at any time.
- Pushing an opponent’s marble off the board might not be advantageous at all times. If such a move yields defense gaps or decreases player’s attack power, it is better not to push off.
- Make defense as tight as possible which might provide attack power in the long run, if the opponent gets careless about its defense.
- Divide and conquer; breaking the opponent’s army into two weakens the opponent’s power. It is easier to deal with two weak armies, reducing the opponent’s offensive ability.

The arguments mainly points out two important elements of playing Abalone well:

1. Keep the marbles around the middle of the board and force the opponent to move towards the edges.
2. Keep the marbles together as much as it is possible, to increase both offensive and defensive power.

4 Abalone Player – ABLA

There are two major Abalone programs that are widely used by human players. One of them is developed by Random Software ([13]), and the other is developed by Tino Werner at the Institute for Theoretical Computer Science, in the University of Technology Graz, named as ABA-PRO ([14]). Aichholzer *et. al.* provide technical details in [1].

We have also implemented a simple intelligent agent for playing Abalone game, named as Abalone Player – ABLA. ABLA utilizes a minimax algorithm and alpha-beta pruning, introducing a new heuristic function. MS Visual Studio IDE is used as a programming environment with DirectX API support. Figure 2 and Figure 3 are example screen shots from ABLA.

4.1 Heuristics used in ABLA

Many Abalone players suggest that placing the marbles as close as to the center of the game board and distracting opponents’ marbles from center is the key to winning a game. By capturing the center, a player achieves two sub-goals:

1. Player’s marbles can not be pushed off the board.
2. Player forces its opponent to make a decision between two disadvantageous choices:
    a. Dividing the army into two, making the army more vulnarable to attacks, and reducing the neighborhood bonus due to the second part of the heuristics
    b. Staying at the edges of the board, hence marbles may be thrown away by the opponent, or at the best the game might end with a draw

As a result, heuristics of ABLA consists of two main parts, supporting most of the winning strategies explained in Section 3.2:

1. **Evaluate closeness to center**: Sum of the *Manhattan distances* of marbles belonging to each player to the center of the board is computed respectively, and their difference is calculated, denoted as $f_1(s)$ for a given board state s. Manhattan distance for a marble’s location to the center on the hexagonal field corresponds to the minimum number of moves required for a marble to get to the center.
2. **Evaluate adjacency**: Sum of the count of neighboring teammates for marbles is computed respectively for each player, and their difference is calculated, denoted as $f_2(s)$ for a given board state s.

First evaluation function serves as a mechanism to achieve the sub-goals mentioned earlier. When the number of allied neighbors for each marble increases, the offensive power of a team increases. The second evaluation function serves keeps the marbles as close as it can be, therefore, increasing their both attack and defense attributes. Both functions, at the same time provide a resistance against being pushed off the board. Overall evaluation function is a weighted linear function combining both features as follows:

$$eval(s) = w_1 f_1(s) + w_2 f_2(s)$$

4.1 Performance of ABLA

ABLA is tested against other Abalone programs that are found on the Internet: Random Soft and ABA–PRO (shareware version). Note that ABA-PRO uses a single feature in its evaluation function that computes centers of mass for each player measures distances of all marbles to their team’s center of mass. For this reason, some initial experiments are performed to understand the contribution of adjacency evaluation. *Combined* heuristic utilizes *eval* function with $w_1 = -1$, $w_2 = 1$, where as *single* heuristic utilizes *eval* function with $w_1 = -1$, and $w_2 = 0$. During all the games gray starts the game first.

A tournament is arranged between four different ABLAs. Two ABLAs utilize both heuristics and 4-ply game tree. The other two ABLAs utilize the very same heuristics, but 3-ply game tree. Result of each game in the tournament is provided in Table 1. The overall result is summarized in Table 2, verifying that the combined heuristic is better than the single one. Considering the games 3 and 6, even if combined’s ply is less than its opponent, ABLA, still, does not lose if it starts the game first and it is able to force the game to a draw at the least. As expected 4-ply player is better than 3-ply player.

Table 1. Results of the games between different ABLAs utilizing *combined* and *single* heuristic, and 3-ply and 4-ply game trees.

Game No.	Gray Player	Difficulty Level	Black Player	Difficulty Level	Score	Winner
1	*Combined*	4-ply	*Single*	4-ply	3 – 3	Draw
2	*Single*	4-ply	*Combined*	4-ply	0 – 1	Draw
3	*Single*	3-ply	*Combined*	4-ply	2 – 6	*Combined*
4	*Single*	4-ply	*Combined*	3-ply	6 – 2	*Single*
5	*Combined*	4-ply	*Single*	3-ply	6 – 1	*Combined*
6	*Combined*	3-ply	*Single*	4-ply	1 – 0	Draw

Table 2. Final scores of each player utilizing combined vs. single heuristics and 3-ply vs. 4-ply search trees.

Agent Name	No. of Games	Win	Loss	Draw	Final Score
4-ply	4	3	0	1	**+12** (18-6)
3-ply	4	0	3	1	-12 (18-6)
Combined	6	2	1	3	**+7** (19-12)
Single	6	1	2	3	-7 (12-19)

ABA–PRO supports a variety of difficulty levels, from *beginner* (denoted as *B*) to *champion*. Werner states that, it has never been beaten in the champion level. On the internet ([14]), only the shareware version is available and a game ends after a certain number of turns, depending on the chosen level. The hardest level provided by the shareware version of ABA-PRO is the *Apprentice* level, denoted as *A*. Random Soft supports several levels; *low*, *medium* and *high*, among which only *medium* level, denoted as *M* could be tested. ABLA is fixed to utilize 4-ply a game tree. Then several games are arranged against ABA–PRO with difficulty levels *A* and *B*. Similarly, ABLA competed against Random Soft. Additionally more games are arranged by assigning the first and second turns to each program.

Table 3. ABLA versus other programs utilizing different difficulty levels.

Game No.	Grey Team	Difficulty Level	Black Team	Difficulty Level	Score	Result
1	Random Soft	*M*	ABLA	4-ply	0 – 0	Draw

2	ABLA	4-ply	Random Soft	*M*	6 – 1	ABLA
3	ABA – PRO	*B*	ABLA	4-ply	0 – 0	Draw
4	ABLA	4-ply	ABA – PRO	*B*	6 – 3	ABLA
5	ABA – PRO	*A*	ABLA	4-ply	0 – 0	Draw
6	ABLA	4-ply	ABA – PRO	*A*	3 – 1	Draw

Table 3 indicates that ABLA could not be beaten even once against its opponents. The draws are due to the repeating, cyclic movements of both players.

5 Conclusions

Playing board games against a computer makes the man test his mind, wisdom, and tactical abilities. By writing computer programs that challenge human opponents, we can see our deficiencies during a play and create new strategies that might have never been thought before.

There are many intelligent opponents, created for chess, checkers, GO, etc. to train human opponents, but there is lack of computer opponents for Abalone, which is less known, being a new game. A simple intelligent agent named as "Abalone Player-ABLA" is implemented using traditional approaches with a new heuristic function. ABLA is a challenging opponent for human players who can not find others to play or need to train themselves alone. Its heuristic seems to be better than the existing versions of Abalone programs, as summarized in the previous section. Therefore, its difficulty level using a 4-ply game tree can be considered medium-high. With its speed and combined heuristic, ABLA provides everything that a human Abalone player would require.

As a future work, numerous features can be added and other approaches can be tested, providing a faster player, such as, utilization of a database of openings and endings, fine tuning of weights in the evaluation function, iterative deepening alpha-beta search, embedding machine learning schemes, etc. Game tree complexity of Abalone seems to be worse than the chess with 60 possible moves on average, at a typical position as reported in [1]. Note that analyses of complexities related to Abalone game are still open for research.

References

1. O. Aichholzer, F. Aurenhammer, and T. Werner, *Algorithmic fun – Abalone*, Institute for Theoretical Computer Science, Graz University of Technology, Austria. (2002)
2. L. V. Allis, *Searching for Solutions in Games and Artificial Intelligence*, Ph.D. thesis, University of Limburg, Maastricht. (1994)

3. D. Edwards, and T. Hart, i963. *The alpha-beta heuristic*, Tech. Rep. 30, MIT AI Memo, Computer Science Dept., MIT, Cambridge, Mass., Oct. Originally published as the Tree Prune Algorithm. (1961)
4. D. E. Knuth, and R. W. Moore, *An analysis of alpha beta pruning*, Artificial Intelligence, Vol. 6 (4) pp. 293-326. (1975)
5. H. Matsubara, H. Iida, and R. Grimbergen, *Chess, Shogi, Go, natural developments in game research*, ICCA J. 19 (2), pp. 103-112. (1996)
6. M. Müller, *Computer Go*, Artificial Intelligence, 134 (1-2), 145-179. (2002)
7. J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA. (1984)
8. A.L. Samuel, *Some studies in machine learning using the game of checkers*, IBM Journal of Research and Development, vol 3, nr. 3, pp.210-229. (1959)
9. C. E. Shannon, *Programming a computer to play Chess*, Philosophical Magazine, Ser.7, Vol. 41, No. 314. (1950)
10. J. R. Slagle, J. E. Dixon, *Experiments With Some Programs That Search Game Trees*, J. ACM 16(2): 189-207. (1969)
11. T. Standage, *The Turk: The Life and Times of the Famous Eighteenth-Century Chess-Playing Machine*, Walker & Co. (2002)
12. T. Takizawa and R. Grimbergen, *Review: Computer Shogi Through 2000*, Lecture Notes in Computer Science, vol. 2063, pp. 433-443. (2001)
13. Abalone Official Site: http://uk.abalonegames.com/
14. ABA – PRO, http://www.cis.tugraz.at/igi/oaich/abalone.html
15. IBM's official Website for Deep Blue "http://www.research.ibm.com/deepblue/
16. Random Software, http://www.randomly.com.
17. Thomas Fenner, Mind Sports Olympiad, Abalone, Champion of the World (2001), Web Site: http://www.tfenner.com/abalone.html