Monte Carlo Tree Search Applied to the Game Abalone*

Joshua Hutson, George Lamb, Chris Alvin
Department of Computer Science
Furman University
Greenville, SC 29613
{joshua.hutson, george.lamb, calvin† }@furman.edu

Abstract

We consider the perfect knowledge, abstract strategy game Abalone as a means of investigating AI search. In particular, we apply a Monte Carlo Tree Search technique to Abalone and assess its efficacy.

1 Introduction

In an artificial intelligence course, we sometimes find it difficult to define a domain and corresponding simple codebase for students to implement novel techniques. In this paper, we consider the abstract strategy game Abalone [3] as a means of exploring tree search. Some modern board games require fifty or more pages of rules, complex boards, and many pieces. By contrast, Abalone is a perfect knowledge game, in the vein of checkers, in which the rules can be described on an index card with a simple hexagonal board and two colors of pieces. Abalone is a game that has sold millions of copies worldwide [3], but is relatively unknown in the United States. Pedagogically it is therefore ideal to explore for a class interested in games, search, or strategy development in a competitive bot.

It is common for search tree techniques such as Minimax to be applied to games such as Chess, Checkers, and Go. While such search tree algorithms have been successful, even greater success has been achieved when combined with a Monte Carlo

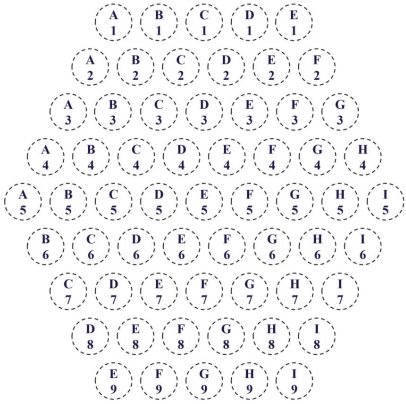

Figure 1: Coordinatized Abalone board.

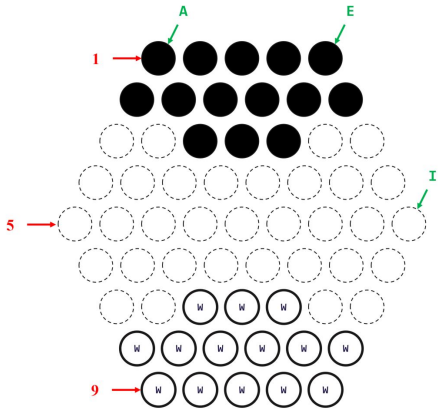

Figure 2: Starting state of Abalone.

technique. In this paper we apply the Monte Carlo Tree Search (MCTS) [4] technique to Abalone with moderate success. Our work is the result of two undergraduate summer research students and serves as a precursor to inclusion in an undergraduate AI course. In Section 2 we describe the game of Abalone. We then provide an introduction to the MCTS applied to Abalone in Section 3. We then empirically report on several MCTS players in Section 4.

2 Game Rules

Gameplay. The Abalone board is a hexagon consisting of 61 individual positions as shown in Figure 2. Each player begins with their own set of 14 black or white marbles positioned symmetrically at opposite ends of the board; black marbles are filled in, white marbles contain a `W`, and empty spaces are dashed. For clarity, we have labeled the horizontal rows from 1 through 9 and the diagonal 'columns' (diagonals, hereafter) from **A** through **I** as shown in Figure 1. This labeling will allow us to speak about particular cells. For example, at the beginning of the game black marbles populate spaces **A1** through **E1**, **A2** through **F2**, and **C3** through **E3**.

Play always begins with black having the first move. The players alternate turns until a player has pushed 6 opposing marbles off the board. Each turn consists of a player making a single *action*. We describe each of the five possible actions noting one definition. We define a **Line** to be a sequence of 2 or 3 marbles of the same color positioned linearly (i.e., all marbles fall on the same horizontal or diagonal); Figure 2 contains a line of 3 white marbles from **E7** through **G7**. According to this definition any two adjacent marbles of the same color form a Line. Therefore, every

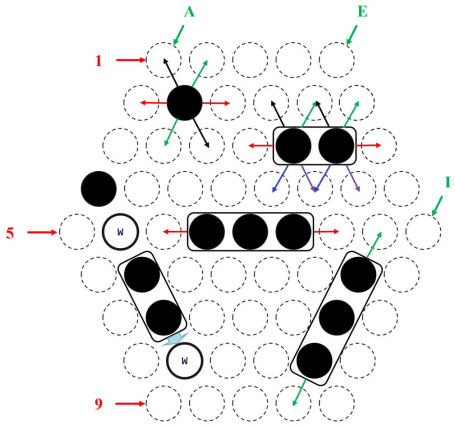

Figure 3: Configurations for possible player actions (not a realistic board state).

Line of length 3 can be broken down into two Lines consisting of 2 marbles. We limit Lines to length 3 as there is no greater utility for longer lines in the game as we will see below with *actions*.

Actions. During a turn, each player must take a single action (i.e., make a move). Although our nomenclature for player moves may differ from some Abalone game instruction nomenclature, our choices are for clarity. Independent of our definitions, we always adhere to proper game rule interpretations.

Simple. A player may move a single marble to any adjacent, valid, empty space. For example, in Figure 3 the black marble in **B2** may be moved to any of the six adjacent spaces (e.g., **A1**, **A2**, etc.).

Line shift. All marbles in a Line may be shifted one position linearly as long as the destination space is valid and empty. In Figure 3, the Line of length 2 in row 3 can be shifted from positions **E3** and **F3** either left or right as indicated by the red arrows. A left Line shift results in the 2 black marbles being moved into spaces **D3** and **E3**; a right Line shift results in the 2 black marbles being moved into spaces **F3** and **G3**. Shifts can apply to any Line; the Line of 3 black marbles in 'column' **H** may shift into positions **H7** through **H9** as indicated with an downward green arrow; similarly upward into spaces **H5** through **H7**.

Sidestep. Similar to a simple move, an entire Line may move as a unit in any 'open' direction. A Line may be moved laterally ('broadside') as a unit assuming all destination spaces are valid and empty. In order to avoid clutter in Figure 3, we focus our example on the horizontal Line of length 2 in row 3. This line may move as a unit in four directions: northeast (green), northwest (black), southwest (blue), or southeast (purple). For clarity, we differentiate Line shifts from side step moves since Line shifts

only require one destination space be open and valid while side step moves require we verify all marbles can be moved to valid and empty spaces.

Push move. The goal of Abalone is to 'eject' six of your opponent's marbles off of the board by pushing. Pushing is accomplished by using numerical superiority ('Sumito') to force an opponent's marble(s) to move. In particular, a Line of length 3 can push a linearly aligned marble of the opponent (3-Push-1 Sumito) or a linearly aligned Line of an opponent of length 2 (3-Push-2 Sumito). Similarly, a line of length 2 can push a single opponent marble (2-Push-1 Sumito). In Figure 3, there is a 2-Push-1 Sumito of the black Line including **C6** and **D7** onto opponent's white marble at **E8** resulting in the black Line at **D7** and **E8** and the white marble onto **F9**. We can only perform a push when the pushing destination space is empty or is off the board. So the black Line of **C6** and **D7** cannot push the white marble at **B5** since **A4** is occupied.

3 Monte Carlo Tree Search

A Monte Carlo method refers to a class of algorithms that use random sampling to identify patterns. In our case, we applied the classic MCTS as a competitor in Abalone. Although many resources exist, we describe the MCTS algorithm to place our experimental parameters into context.

Algorithm 1 Monte Carlo Tree Search Algorithm

```
1:  function MCTS(root)
2:      root: Root of Tree
3:      EXPAND(root)
4:      while RESOURCESREMAIN( ) do                    ▷ Time or Iterations
5:          ℓ ← TRAVERSE(root)          ▷ Identify a leaf node via best UCB score
6:          if ROLLEDOUT(ℓ) then                       ▷ Previously rolled out node.
7:              EXPAND(ℓ)
8:              ℓ ← RANDOMCHILD(ℓ)
9:          SF_ℓ ← ROLLOUT(ℓ)              ▷ Monte Carlo game simulation from ℓ
10:         BACKPROP(ℓ, SF_ℓ)              ▷ Back-propagate result from ℓ up to root
11:     return BESTCHILD(root)
```

Algorithm. As described in Algorithm 1, MCTS is a technique defined by four core sub-routines: EXPAND, TRAVERSE, ROLLOUT, and backpropagation (BACKPROP). We consider each of these operations in turn.

Expansion. Given a node n in a search tree T, *expansion* refers to adding all children of n to T. With Abalone, each node in the tree maintains a game board layout. A move m (as described in Section 2) transitions a parent node $p \in T$ to a child node cT: $p \xrightarrow{m} c$. Expansion can be precarious with Abalone since there are 44 possible moves for the black player in Figure 2 as implemented on Line 3 of Algorithm 1.

Traversal. Each iteration of the MCTS algorithm must traverse the existing tree to identify a leaf node to either *expand* or *rollout*. Each node n in a tree T maintains two attributes: (1) a cumulative score (S) updated when n is included in a simulated gameplay path and (2) a count (N) of number of times n is visited. The decision about which child c of n to visit next is based on the largest value of the Upper Bound Confidence Bound 1 applied to Trees [4] (UCB1) for a parent and each of its children:

$$\mathsf{UCB1}\,(p, c) = \frac{c_S}{c_N} + E\sqrt{\frac{\ln p_N}{c_N}} \tag{1}$$

where E is an exploration parameter. Equation 1 consists of two expressions. The expression $\frac{c_S}{c_N}$ refers to the level of success previously found by traversing the particular edge between parent and child; hence, the colloquial name 'exploitation' expression. The second expression (bandit expression) $E\sqrt{\frac{\ln p_N}{c_N}}$ values new nodes to explore; UCB1 returns ∞ for a child that has not been explored (i.e., $c_N = 0$). We can tune the importance of exploration in a game by selecting E to be small in order to avoid a breadth-first exploration that may arise in games with many options for moves.

Rollout. Given a leaf node ℓ from traversal (Line 5 in Algorithm 1), we expand it if needed (Line 6 through Line 8), and simulate a game between two players who choose their moves randomly (Section 4 describes how our approach to rollouts is semi-random). When a rollout has reached its conclusion either by the game finishing or a cutoff parameter being reached, we return a score $\mathsf{SF}_\ell \in [-1, 1]$ indicating success or failure attributed to a game starting with ℓ for the root player's turn.

Backpropagation. For each node n in the path from ℓ to the root tree T on Line 10, we update n_S (total score of n), additively with $\mathsf{SF}_\ell(Board, P)$ where P is the current player, and increment the number of times n was visited (i.e., $++n_N$).

Iteration and Return. We continue to iterate on Line 4 through traversal, rollout, and backpropagation until we exhaust our search parameter(s): time, space, or some other cutoff (e.g. maximum number of rollouts, etc.). Last, on Line 11, we choose move m from $root \xrightarrow{m} c$ where c_S is the maximum among all children of $root$.

4 Experimental Analyses

The goal of our experiments is to explore the utility of the MCTS algorithm for Abalone.

Bot Players. There are several parameters that can be tuned with the MCTS algorithm. Our experiments attempt to demonstrate the goodness of a MCTS player compared to other naive players. As shown in Table 1, we had 7 different players compete against one another. We defined three Player Types that choose moves at random: (1) a player that randomly chooses one move out of all possible moves, (2) a player that prioritizes a random *Push* move (if one is not available it randomly chooses from

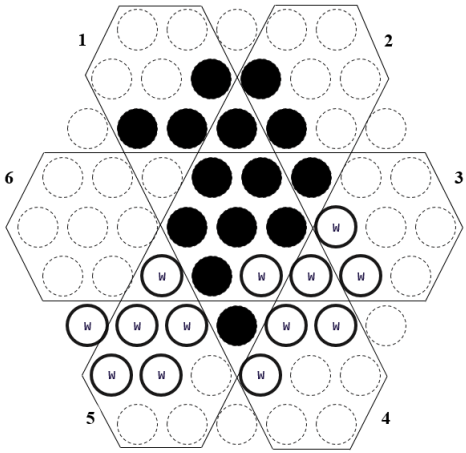

Figure 4: A sample penetration-based mid-game board state with regions indicated for dominance-based board-state scoring.

one of the other moves), and (3) a tiered player that chooses a random move from a category of moves in the order *Push*, *Line Shift*, *Sidestep*, and *Simple*.

Our final four players choose moves based on MCTS. Player Type 4 implements the MCTS loop condition (Line 4 of Algorithm 1) with a fixed number of iterations. Player Type 5 iterates a fixed amount of time.

For our remaining players, we recognized that early-game board states and mid-game board states are quite different. We define a *formation* to be a connected subgraph of marbles of a single color; for example, Figure 2 has two distinct formations. We also define *adjacent formations* as two formations, W and B, of opposing color marbles in which at least one marble of W is adjacent to one marble of B. Although it is difficult to identify when a mid-game board state has occurred, we relied on our experience with the game to define two criteria. We define *mid-game* as a board state in which (a) one marble has been pushed off the board or (b) one formation of marbles, B, penetrates through an opponent's marbles resulting in two formations, W_1 and W_2, both adjacent to B (see Figure 4 for an example). Thus, our last two hybrid players switch modes mid-game. Player Type 6 starts with an iteration-controlled MCTS and transitions to time-controlled; Player Type 7 took the opposite order of loop control (time-based MCTS followed by iteration).

MCTS Parameters. Players following a MCTS must define the exploration constant: E in Equation 1. Since Abalone defines an exponential search space, it is easy to explore vast shallow elements of the search tree. In order to favor exploitation over exploration we empirically identified $E = 0.05$ to provide a balance between depth (roughly 8 to 10 moves deep) and breadth (approximate maximum width of 15000).

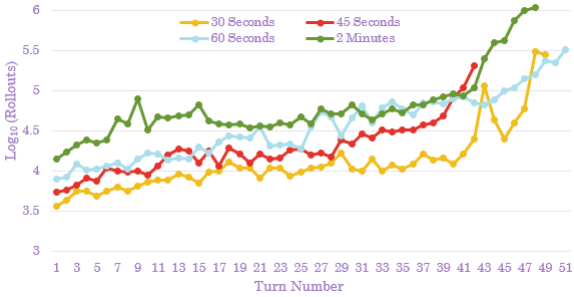

Figure 5: Scaled number of rollouts per turn for time-based iteration of MCTS.

For the rollout operation of MCTS we must identify a reasonable upper bound for the number of turns in a rollout-played game as well as what random Player (either Type 1, 2, or 3) will play rollout games for our MCTS Players (Type 4, 5, 6, and 7). Empirically, we observed strictly random rollout games completed after 750 moves; hence we set the maximum number of moves for rollout to be 800. Although this is excessive compared to a standard game which last about 100 turns, we wanted to ensure rollout games completed. To expedite the rollout process, instead of using strictly random players (Player Type 1), we instead used a random push mover (Player Type 2) for all MCTS Players. Our goal in this choice was to maintain as much randomness as possible in the spirit of a Monte Carlo method, but to expedite execution as well; 10000 iterations of MCTS with rollout using Player Type 2 was almost 8 times faster than using Player Type 1.

As described in Section 3 with Algorithm 1, it is common for iteration to be based on time. We executed Player Type 4 using 30, 45, 60, and 120 seconds per move with the number of resulting rollouts shown in Figure 5. We observe in Figure 5 that there was a significant difference between the number of rollouts in the early- and mid-game. Ultimately, we want as many rollouts as possible and thus the 30 and 45 second players were not able to make a strong, informed move. We observed that a 60-second turn gives satisfactory results while the 120-second did not make a notable, significant success in move selection (and win ratios) compared to 60-seconds. Hence, Player Types 5, 6, and 7 use 60-second turn bounds.

For Player Types 4, 6, and 7 using iteration-based MCTS we must identify a reasonable upper bound number of iterations. Comparing 1000, 5000, and 10000 iterations, we observed a reasonable balance between tree depth and execution time using 5000 iterations noting the Player took less than a minute toward the beginning of the game and less than five seconds by the end of the game.

Board-State Scoring. When rollout game play is stopped or it completes, we define $\mathsf{SF} : Board \times P \rightarrow [-1, 1]$ where -1 indicates a loss for a player P and 1 indicates a win. If the game is not complete, we weight two different methods for

scoring a board equally, one method based on number of pieces and one based on piece placement so that players would seek avenues to win while attempting to maintain strong board positioning.

Piece-Based Scoring. It is a distinct advantage to have more marbles than your opponent. We defined an exponential piece scoring function (ps) emphasizing the difference in the number of remaining marbles ($|M|$) between players:

$$\mathsf{ps}\,(Board) \;=\; \mathsf{ms}\,(B) \;-\; \mathsf{ms}\,(W)\;;\; \mathsf{ms}\,(M) \;=\; \begin{cases} 2^{(14-|M|)/7} & 10 \le |M| \le 14 \\ 2^{(14-|M|+1)/7} & |M| = 9 \end{cases}.$$

We note the non-linearity of this function as to award higher scores for a player being closer to a win. In particular, we valued having 5 opponent marbles pushed off (9 remaining on board) to indicate a strategic advantage (allowing the player to be more aggressive); this accounts for dividing by 7 instead of 6 in ms since the fifth marble pushed off can be viewed as twice as important as all others. As an example, consider the case where a player has pushed off 5 opponent marbles compared to their opponent pushing off only 2 of theirs, we would calculate $\mathsf{ps}\,(Board) = 2^{6/7} - 2^{2/7} \approx 0.592$.

Dominance-Based Scoring. Our second scoring method considers board dominance of a player and it consists of two equally weighted parts. The first part splits the board into 7 hexagonal regions as shown in Figure 4 (region 7 is in the center of the board) and their corresponding center spaces in **B2**, **E2**, etc.; in this configuration, there are 6 interior and 6 exterior spaces on the board that are not within a region. A player earns a score of 1 for each region except for the center hexagon which receives a score of 2. Dominance of a region r is determined by the color marble that is situated on the center space of r. If no marble occupies the center, r is considered to be dominated by the player with the majority of marbles in r. In Figure 4, white dominates area 5 since it occupies the center and black dominates regions 1 and 2 since black has the majority of marbles. In Figure 4 white dominates 3 regions and black dominates 3, including the center. We would compute a dominance score for the black player as $\mathsf{dom}\,(B) = (4-3)/8 = 0.125$ where division normalizes the value to $[-1, 1]$.

The second part of dominance-based scoring considers *cohesion* of marble formations; tighter groupings are more valuable to prevent opponent Sumitos. If a player's marbles consist of multiple formations, the cohesion score is 0. For a player with a single formation, we compute the *spread* of the formation as the straight-line distance between the farthest nodes in a formation subgraph. In Figure 4, black consists of a single formation so we compute the distance between **F7** and **C2** (using Law of Cosines) as $\sqrt{(7-2)^2 + (6-3)^2 - 2\cdot(7-2)\cdot(6-3)\cos 60^\circ} = \sqrt{19}$; normalized to the maximum distance on the board (9), we have $\sqrt{19}/9 \approx 0.484$. Since we want formations with less spread, we take the complement: $1 - 0.484 = 0.516$.

As an example of the overall scoring computation, consider the board state in Figure 4 evaluating for black. For piece count we have $\mathsf{ps}\,(Board) = 2^{1/7} - 2^0 \approx 0.104$. Hence, $\mathsf{SF}(Board, B) = \frac{1}{2}\cdot\left[0.104 + \frac{1}{2}\cdot(0.125 + 0.516)\right] \approx 0.212$.

Analyses. We executed each of our Player Types against one another with win

Player 1 \ Player 2	Random Player All Move	Random Player Push Move	Random Player Tiered Move	Monte Carlo Tree Search	Time Based Tree Search	Hybrid Player Time First	Hybrid Player Move First
Random Player All Move	$\frac{819,472}{2,000,000} = .410$	$\frac{500}{2,000,000} = .00025$	$\frac{600}{2,000,000} = .0003$	$\frac{0}{50} = 0.0$	$\frac{0}{50} = 0.0$	$\frac{0}{50} = 0.0$	$\frac{0}{50} = 0.0$
Random Player Push Move	$\frac{1,999,980}{2,000,000} = .999$	$\frac{996,441}{2,000,000} = .498$	$\frac{848,064}{2,000,000} = .424$	$\frac{0}{50} = 0.0$	$\frac{0}{50} = 0.0$	$\frac{0}{50} = 0.0$	$\frac{0}{50} = 0.0$
Random Player Tiered Move	$\frac{1,999,990}{2,000,000} = .999$	$\frac{1,144,301}{2,000,000} = .572$	$\frac{996,631}{2,000,000} = .498$	$\frac{0}{50} = 0.0$	$\frac{0}{50} = 0.0$	$\frac{0}{50} = 0.0$	$\frac{0}{50} = 0.0$
Monte Carlo Tree Search	$\frac{50}{50} = 1.0$	$\frac{50}{50} = 1.0$	$\frac{50}{50} = 1.0$	$\frac{25}{50} = .5$	$\frac{23}{50} = .46$	$\frac{20}{50} = .4$	$\frac{25}{50} = .5$
Time Based Tree Search	$\frac{50}{50} = 1.0$	$\frac{50}{50} = 1.0$	$\frac{50}{50} = 1.0$	$\frac{21}{50} = .42$	$\frac{17}{50} = .34$	$\frac{19}{50} = .38$	$\frac{17}{50} = .34$
Hybrid Player Time First	$\frac{50}{50} = 1.0$	$\frac{50}{50} = 1.0$	$\frac{50}{50} = 1.0$	$\frac{23}{50} = .46$	$\frac{18}{50} = .36$	$\frac{21}{50} = .42$	$\frac{21}{50} = .42$
Hybrid Player Move First	$\frac{50}{50} = 1.0$	$\frac{50}{50} = 1.0$	$\frac{50}{50} = 1.0$	$\frac{22}{50} = .44$	$\frac{22}{50} = .44$	$\frac{20}{50} = .4$	$\frac{23}{50} = .46$

Table 1: Empirical win rates for first player (black) $\times$ second player (white) games.

rates reported in Table 1. As a measure of confidence in our implementation, we observe that the main diagonal of the upper left corner of the table indicates a fair, 50% win rate specifically over the large sample size of 2000000 games. Of note, we believe the lower win ratio for the Player Type 1 is a result of not finishing games in the number of moves allowed. We also observe that all MCTS Player Types 4, 5, 6, and 7 beat all random Players (Types 1, 2, and 3) 100% of the time. The remaining games involving MCTS Players seem to represent a relatively small sample size. However, a typical game between two MCTS Players such as two Player Type 5 can take approximately 2 hours to complete; hence, Table 1 represents weeks of computing time. Among the MCTS Player Types, there is no clear evidence of a 'best' player as all had win rates below 50%. As a result, while our sample size is small, these data suggest that the second (white) player has an advantage over the first (black) player.

5 Related Works

Chorus [1] considers different agents to play Abalone. While they conclude that the Alpha-Beta algorithm is superior to MCTS, their MCTS scoring approach is naïve in that they only use the number of pushed marbles as a method of scoring in a case where one player does not win outright. Whereas our scoring technique relies on state-based scoring of the board as much as it does counting pushed marbles by giving weight to the number of marbles pushed off thus reflecting an advantage of having more marbles on the board in the late game. Our approach does not consider an alpha-beta approach since our proposed goal is to investigate a more recent technique that has experienced some success with other games (e.g., Go using AlphaGo [2], Chess

using an AlphaGo adaptation [6] in Leela Chess Zero, and more).

Ozcan and Hulagu [7] attempt to heuristically play Abalone through knowledge of the value of the center. Their Abalone player attempts to occupy the center of the board first to put their opponent in a disadvantageous position. We attempted something with our scoring method, but found that incentivizing the center of the board created a player that was afraid to push marbles off the board. Hence, we incentivized cohesion over acquiring and maintaining the center.

Our lack of a conclusion as to the best MCTS Player is similar to the findings of Kozelek [5] when applying MCTS to Arimaa, a game that is similar, yet more complex than Chess. He stated the MCTS "gave very poor performance" and suggested a static evaluation of the board at given points in time.

6 Conclusions

We considered the game Abalone as a venue for agent play. We presented the MCTS algorithm along with many different parameters for tuning the algorithm with respect to Abalone. We were not able to conclude a best MCTS Player for Abalone, but we were able to intuit that the second player has the advantage. There are many potential avenues for further consideration of MCTS with Abalone as an academic exercise; however, the search space is prohibitive.

References

[1] Pascal Chorus. Implementing a computer player for abalone using alpha-beta and monte-carlo search. Master's thesis, Maastricht University, June 2009.

[2] Michael C. Fu. Alphago and monte carlo tree search: The simulation optimization perspective. In *Winter Simulation Conference (WSC)*. IEEE, 2016.

[3] University Games. The game: Abalone, July 2021. https://www.universitygames.com/abalone.

[4] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*. Springer Berlin Heidelberg, 2006.

[5] Tomáš Kozelek. Methods of mcts and the game arimaa. Master's thesis, Charles University in Prague, 2009.

[6] Gary Linscott. Leela chess zero, July 2021. https://lczero.org/.

[7] Ender Ozcan and Berk Hulagu. A simple intelligent agent for playing abalone game. Turkish Symposium on Artificial Intelligence and Neural Networks, 2004.