# Intelligent game-playing agent for Abalone

Saint-Hillier, Achille
2081283
*Software and Computer Engineering*
*Polytechnique Montréal*
achille.saint-hillier@polymtl.ca

Benoit-Paraschivoiu, Mathéo
2079549
*Software and Computer Engineering*
*Polytechnique Montréal*
matheo.benoit-paraschivoiu@polymtl.ca

## I. Contextualization

This report presents an intelligent agent specialized in an altered version of the game Abalone. This game is played on a hexagonal board composed of 61 cells. Players have 50 moves to win the game, either by ejecting as many enemy pieces as possible from the board (up to a maximum of six pieces), or by getting as close as possible to the center. Table I illustrates the complexity of this game in a software context. It can be observed that it offers fewer possibilities than Go, but more than Chess on average. Therefore, it is obviously impossible to explore all game states. To develop an intelligent agent capable of playing this game, we have chosen to implement a ***Minimax***-type algorithm, which we will discuss in more detail later.

The constraints of this project are as follows: the agent must be implemented in *Python* using the *SeaHorse* library, it must play within 15 minutes per game, and it must use less than 4 GB of RAM.

In the following sections of this report, we will examine our implementation in detail, highlighting the selected heuristics and mentioning various optimization approaches undertaken. We will then present the final selected agent and the quantitative results obtained. Finally, we will discuss possible improvements and conclude on the project.

TABLE I
Statistics on various board games [1] [2]

| | **Chess** | **Abalone** | **Go** |
|---|---|---|---|
| **Branching Factor** | 35 | 50 | 250 |
| **Average Moves per Game** | 80 | 50 | 150 |
| **log(Game Tree Size)** | 123 | 86 | 360 |

## II. Implementation

As mentioned earlier, we use a *Minimax*-type algorithm. *Minimax* is a search algorithm used in two-player games, alternating between a player seeking to maximize their score and their opponent seeking to minimize it [2]. It recursively explores a game tree, evaluates final positions, and chooses the best move by considering possible opponent responses. By evaluating all possible moves to a certain depth, the algorithm determines the best sequence of moves assuming the opponent plays optimally as well. More precisely, we have implemented a *Minimax Heuristic*, as we evaluate the score and quality of states using heuristics.

Our implementation has therefore focused on two main aspects: the development of heuristics, and the pursuit of speed/performance in order to optimize our algorithm.

### A. Heuristics

We implemented several heuristics throughout the session, ultimately keeping four that would improve the performance of our intelligent agent. Here, we will explain the retained heuristics.

*1)* ***Center of Mass****:* Our main heuristic relies on a center of mass mechanism [3]. First, we calculate the board cell representing the center of mass of each group of marbles, as illustrated in step 1 of Figure 1. Then, we define a new cell resulting from a weighted average between the positions of the centers of mass and the center of the board. The coefficient assigned to the center of the board can vary to adjust the behavior of the intelligent agent. A high coefficient favors actions approaching the center, while a low coefficient favors a more aggressive approach by getting closer to the opponent. This step is visualized as step 2 in Figure 1. Finally, we evaluate the average distance from this previously defined cell. In the case of eliminating a marble, a high distance is associated with it to apply a significant penalty to actions resulting in the loss of a marble. This last step is represented as the final step in Figure 1. This heuristic is versatile and generates good

actions on its own. It encompasses the two main features of the game: avoiding elimination and getting closer to the center.

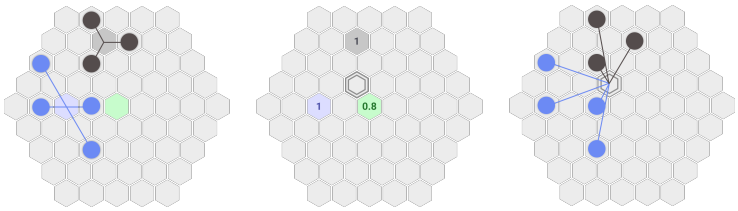


Fig. 1. Illustration of the "Center of Mass" heuristic

*2) **Numerical Superiority**:* We have also implemented a numerical superiority heuristic. For each row and each diagonal present on the board, we evaluate the number of positions where numerical superiority can be established, defined as a position in which a player can push an opponent's marble. We calculate the difference in numerical superiority between our marbles and those of the opponent, seeking to maximize this disparity. This heuristic allows us to adopt more advantageous positions and pursue a more aggressive strategy. A visual representation is available in Figure 2.

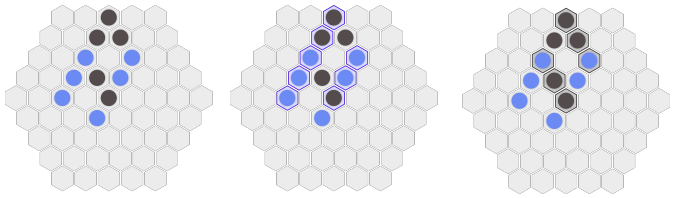

Fig. 2. Illustration of the "Numerical Superiority" heuristic

*3) **Cluster Formation**:* Additionally, we have incorporated a heuristic aimed at favoring a clustered arrangement. For each marble in our set, we evaluate the number of immediate neighbors of the same color. This evaluation generates a score ranging from zero to six for each marble, which we seek to maximize. Although this score is specific to each marble, the group effect promotes the formation of a complex and clustered neighborhood, thus creating advantageous positions. Indeed, in this game, unity is strength. The graphical representation of this heuristic is accessible via Figure 3.

*4) **Pieces on the Edge**:* Finally, we have also implemented a very simple heuristic that involves counting and returning the number of our pieces positioned on the edge of the board, which are considered to be in dangerous positions, exposed to possible elimination. This approach allows us to minimize situations in which our pieces could be easily captured.

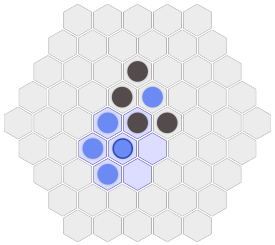

Fig. 3. Illustration of the "Cluster Formation" heuristic

It is important to note that all the heuristics mentioned above are calibrated by weights (hyperparameters). The objective of these weights is to assign differentiated importance to each heuristic while normalizing the values of all heuristics on a common scale. These weights were initially configured intuitively and then manually adjusted following simulations of games to ensure their effectiveness. An improvement to this approach will be proposed later on.

### *B. Seeking Performance and Speed*

In order to optimize our *Minimax* algorithm to visit fewer states and make the execution of our agent faster, we employed three techniques: alpha-beta pruning, the use of transposition tables, and sorting of possible actions.

**Alpha-beta pruning** is a classic technique covered in the course, and our implementation reflects the teachings provided within the course [2]. Therefore, we will not further discuss it in this report.

We have also implemented a **transposition table**, which keeps track of all previously visited states, aiming to avoid revisiting already explored states [2]. To achieve this, we attempted to use *Zobrist hashing*. However, since the *SeaHorse* library used returns a complete state rather than a move difference, we could not implement this hashing in the recommended way (in $O(1)$). Instead, we had to hash the entire state, including the entire board (in $O(N)$), and keep these hashes in memory using a dictionary. Despite this less optimal approach, it nonetheless contributed to optimizing our algorithm, as shown in Tables II and III, and thus we chose to retain it.

To further optimize alpha-beta pruning to cut more subtrees, it is necessary to proceed with the **ranking of possible actions**, thus favoring the analysis of actions likely to generate a higher score first. To determine which actions have promising potential, we have implemented a certain logic. Firstly, we give priority to actions that move three marbles while simultaneously eliminating an

opponent's marble. Next, we prioritize actions involving the movement of two marbles, also with the elimination of an opponent's marble. The next most prioritized actions are those where we move three marbles without eliminations, followed by actions involving the movement of two marbles, and finally, those involving the movement of a single marble. The illustration of this logic is available in Figure 4.

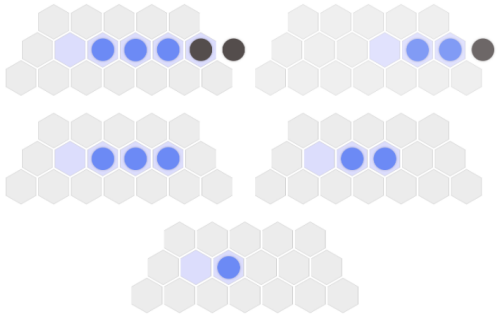

Fig. 4. Move Ordering Priority

The speed improvements of previous approaches based on positions are illustrated in Tables II and III for a depth of three. Thus, we can observe enormous speed gains and a reduction in the number of visited states. The time per move is approximately 46 times faster for the classic position and 100 times faster for the alien position. Furthermore, the number of visited states is reduced by a factor of 133 for the classic position and 234 for the alien position.

TABLE II
VISITED STATES AND AVERAGE TIME PER MOVE BASED ON OPTIMIZATION TYPE FOR A CLASSIC POSITION AT DEPTH THREE

| **Optimization type** | **Visited States** | **Move Duration (s)** |
|---|---|---|
| *Minimax* (base) | 3373177 | 650.2 |
| *Minimax* ($\alpha/\beta$) | 104019 | 41.4 |
| $\alpha/\beta$, *hashing* | 82654 | 37.1 |
| $\alpha/\beta$, *hashing* and ordering | 25368 | 14.2 |

TABLE III
VISITED STATES AND AVERAGE TIME PER MOVE BASED ON OPTIMIZATION TYPE FOR AN ALIEN POSITION AT DEPTH THREE

| **Optimization type** | **Visited States** | **Move Duration (s)** |
|---|---|---|
| *Minimax* (base) | 4914129 | 857.5 |
| *Minimax* ($\alpha/\beta$) | 141296 | 41.7 |
| $\alpha/\beta$, *hashing* | 106373 | 28.4 |
| $\alpha/\beta$, *hashing* and ordering | 20998 | 8.6 |

We limit ourselves to a depth of three due to the competition constraints mentioned earlier. Indeed, considering that the game has a maximum of 50 moves, and most of the time all moves are played, our agent must play 25 moves within a 15-minute interval, averaging 36 seconds per move. This time constraint thus restricts the depth of our search tree. Based on our heuristics and optimizations detailed here, and considering our development environment (Python), it appears in practice that a depth equal to or greater than four moves is not achievable within the allotted time. We can see that the time taken for a depth of four (see Table IV) significantly exceeds the allocated 36 seconds per move. This depth limit constitutes a Type **A** search strategy [2], in which the search stops at a maximum depth.

TABLE IV
VISITED STATES AND AVERAGE TIME PER MOVE ACCORDING TO GAME MODE FOR A DEPTH OF FOUR WITH ALL OPTIMIZATIONS

| **Game Mode** | **Visited States** | **Move Duration (s)** |
|---|---|---|
| Classic | 269315 | 85.7 |
| Alien | 229242 | 79.9 |

We have also introduced dynamic depth in our agent to speed up its gameplay. The first six moves in the classic game are analyzed at a depth of 1, allowing our agent to play more quickly. This optimization is made possible by the simplicity of the early stages of classic games, which often focus on a race towards the center and do not require a thorough analysis of future moves. Additionally, if we detect that we have less than 100 seconds left to play, we also reduce the depth to 1 to limit the time taken by the agent. For the rest of the game, our depth remains fixed at three.

Furthermore, we attempted to enhance this dynamic depth with a defensive agent principle. Indeed, if the tree has an odd depth, we end up studying one of our actions, while in the case of an even depth, it is an opponent's action that is studied. Depending on the case, due to the Minimax algorithm used, the agent will be more aggressive or defensive, respectively. Thus, we have some agents that, in addition to dynamic depth, have a defensive depth, meaning they drop to depth two when winning to play more defensively.

## III. Results

We implemented several agents based on combinations of heuristics and tree depths (see Table V). It is important to note that the performance of the various agents varies depending on the combinations of heuristics used, as well as the depth of our search tree. These performance disparities will be examined in this section.

TABLE V
DIFFERENT IMPLEMENTED AGENTS AND THEIR SPECIFICS

| Agent | Depth | Heuristics |
|---|---|---|
| A | 1 | Random |
| B | 1 | First 3 |
| C | 2 | First 3 |
| D | 3 (Dynamic) | First 3 |
| E | 3 (Defensive) | All |
| F | 3 (Dynamic) | All |
| G | 3 (Defensive) | First 3 |

In order to test the implementation of several intelligent agents to find the best one, we developed a game simulator. This simulator runs multiple games (the number of which is a parameter chosen by the user) between two players and then exports the results to an Excel file. To do this, two games are simulated in the classic position and two in the alien position, each with a different starting player. We do not simulate more than two games since our agents are deterministic due to the ordering of our actions, and therefore a game with the same two agents on the same board will be identical. Subsequent games are simulated on random but symmetric board positions (see Figure 5). Again, two games for each configuration, with each player starting once.

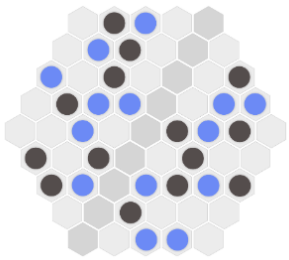

Fig. 5. Example of Randomly Generated Board

In order to obtain good statistics and an objective idea of the best agent, we simulated 20 games between each possible combination of players (21 matchups in total), for a total of 420 simulated games. The average results of these simulations are published in Table VI.

TABLE VI
AVERAGE RESULTS OF THE SIMULATIONS

| Agent | Score | Time Left | Winrate |
|---|---|---|---|
| A | 0 | 899.98 | 0.00 |
| B | 1.86 | 897.2 | 44.33 |
| C | 2.78 | 859.54 | 66.67 |
| D | 2.58 | 537.91 | 69.44 |
| E | 2.08 | 501.63 | 41.67 |
| **F** | 2.67 | 558.64 | **100.0** |
| G | 2.47 | 518.67 | 52.78 |

Another way to visualize these results, specifically the victories, is in the form of a graph, as seen below in Figure 6. An arrow from A to B means that A has lost to B on average (so out of 20 simulated games, it loses 11 or more).

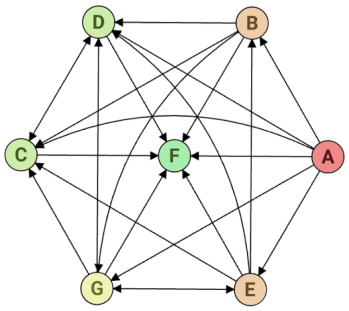


Fig. 6. Wins and Losses of the Different Agents

Analyzing these results, several observations emerge. Firstly, the defensive strategy does not seem to work. Indeed, for an agent with the same heuristics, the defensive version performs worse than the dynamic version. Agent E experiences a decrease in victory rate of approximately $28\%$ compared to D and loses against it, while agent G reduces its victory rate by about half compared to agent F.

Furthermore, our agents have ample time to finish their games within the allocated time, which is reassuring. Finally, we notice that the best agent is agent **F**, which is an agent with dynamic depth (but not defensive) utilizing all the aforementioned heuristics. This agent won all its games and has an average score of $2.67$, while playing in less than $400$ seconds per game on average (with an average remaining time of $558.64$ seconds out of $900$).

## IV. Discussion

We focused on developing multiple agents, allowing us to explore various theories, isolate modifications to evaluate their impacts, and ultimately determine our best agent. Our systematic approach enables us to confidently identify the best among our agents, which is a significant advantage. Additionally, we believe that our methodology, rooted in quantitative and scientific principles, substantially reduces uncertainty. We are also satisfied with our speed optimizations, allowing us to comfortably play well above the allocated time threshold. However, several elements could be beneficial for improving our agents.

Firstly, as discussed earlier, we assign weights to our heuristics relatively arbitrarily, based on our intuition, which is obviously not ideal. We are convinced that a notable advantage for our approach would have been to learn these weights. If we had had enough data (which eventually happened with a large number of approximately $1500$ simulated games during the session), we could have learned the weights through supervised learning. This approach would have involved systematically changing weights, evaluating the resulting scores, and determining match success. Although this method would have taken time and required converting simulated games into exploitable data, it would likely have significantly improved our agents' performance. Moreover, implementing such a method seems intriguing to us.

Furthermore, another possible improvement for our agents would have been to attempt to develop a reactive agent, i.e., an agent capable of detecting the opponent's playing style and adjusting its heuristics accordingly. Although this is challenging to implement, we see considerable potential for improvement. Indeed, during the course competition, our agent never lost in the group stage only to be eliminated in the first round of the final round. Additionally, a team against which we had won in the group stage reached the top four. Thus, it is evident that the win properties of our agent are not transitive, as a victory against one agent does not necessarily guarantee a victory against agents that lost to it, for example. An agent capable of adapting, at least partially, to the opponent's playing style might thereby potentially increase this transitivity and avoid losing to an agent whose playing style is incompatible with ours.

Lastly, considering the average remaining time for our agents, it would have been interesting to deepen the search to a depth of four in case of capturing pieces or other promising states. This approach aligns with the theory of Type B strategies [2], aiming to explore states requiring in-depth analysis further.

## V. Conclusion

In conclusion, we have presented the development of an intelligent agent specifically designed for the game of Abalone. We have elaborated on the underlying theory of its performance and detailed our systematic methodology, which includes simulating a large number of games. By quantitatively analyzing this data along with the behavior of our agents, we have successfully identified an agent surpassing its peers in terms of performance. We are satisfied with the development of our agent, particularly regarding the adopted methodology, which significantly limits the reliance on randomness. The insights gained from this competition have been invaluable, and creating a high-performing agent in the context of Abalone represents a rewarding achievement. This initial step in designing specialized intelligent agents for specific games has instilled in us the desire to continue these efforts and develop further agents.

## References


[1] Wikipedia. (2023). Game Complexity. [En ligne]. Disponible : https://en.wikipedia.org/wiki/Game_complexity

[2] Cappart, Q. (2023). INF8175 - Intelligence artificielle : méthodes et algorithmes. [En ligne]. Disponible : https://moodle.polymtl.ca/course/view.php?id=2667

[3] Aichholzer, O., Aurenhammer, F., & Werner, T. (2002). Algorithmic fun - Abalona. Telematik. [En ligne]. Disponible : http://www.ist.tu-graz.ac.at/staff/aichholzer/research/rp/abalone/tele1-02_aich-abalone.pdf