Université Libre de Bruxelles

INFO-H410

PROJECT

# Techniques of Artificial Intelligence :

## Othello/Reversi
## Benchmark of Minimax Alpha-Beta Pruning

Students

Chassagne Aurélien
000474581

Frauenkron Gaëlle
000459939

Salmani Salma
000507899

Professor

Bersini Hugues

May 2022

# 1 Introduction

Nowadays, **Artificial Intelligence(AI)** is considered as the fastest changing domain and the more famous one, it serves in developing different aspects in IT and non IT world. In this project, we implemented[1] the MiniMax algorithm with different parameters for the **Reversi/Othello** game. After this we will evaluates the influence of these parameters (heuristic, cost function, alpha-beta pruning and depth increasing).

# 2 Othello / Reversi game

## 2.1 Definition

Reversi is a strategy board game for two players, played on an 8×8 board. The game begins with four disks placed in a square in the middle of the grid, two facing light-side-up, two dark-side-up, so that the same-colored disks are on a diagonal. The player with the most pieces on the board at the end of the game wins.

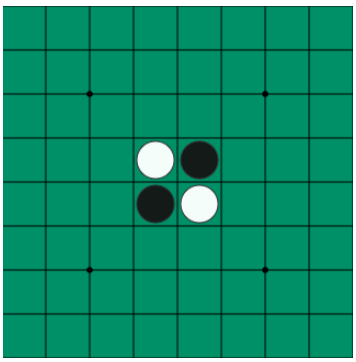


Figure 1: Initial configuration of Reversi board

A pawn can only be placed when it allows to capture pawns. When we place a pawn, we turn over all the adversary pawns sandwiched in a straight line between this pawn and our other pawns. For more details on the rules, please see the official rules[2]

## 2.2 Cost Function

This function calculates the difference in the score of the two players. There is 3 parameters: **Min, Max, Hybrid**. The Min and Max parameters always aim to take the min or max number of pieces but the hybrid parameter consists in minimizing the value at the beginning because the less pieces you have, the less moves your opponent can make and the more likely he is to be forced to make a 'bad' move. Then when we arrive towards the end (in our implementation at turn 45 out of 60) we start to maximize the cost factor to recover a maximum of points before the end.

## 2.3 Heuristic

The heuristic implemented here is a naive one that use the impregnable corner positions. In the cost function calculation, we say that the pawns on the corners are worth much more points (50 instead of 1). Moreover the value of the pieces adjacent to the corner, that would allow the adversary to take the corner, have bad score (-20 instead of 1) but only when the corner is free. As a result, the AI will do its best to not leave the corner to the opponent (penalty of -50) and take it instead (gain of 50).

[1] Git repository : `https://github.com/AurelienCha/AI_Othello`
[2] Rule of the game : `https://www.worldothello.org/about/about-othello/othello-rules/official-rules/english`

# 3 MiniMax Algorithm

## 3.1 Basic MiniMax

Minimax is a classic depth-first search technique for a sequential two-player game. The minimax algorithm rotates between the players and calls the **max_value** function for the player and the **min_value** for its adversary. The **max_value** search the optimal move for the player and the **min_value** search the optimal move for its adversary so the minimum of the player. The search tree is created by recursively expanding all nodes from the root in a depth-first manner until either the end of the game or the maximum search depth is reached.

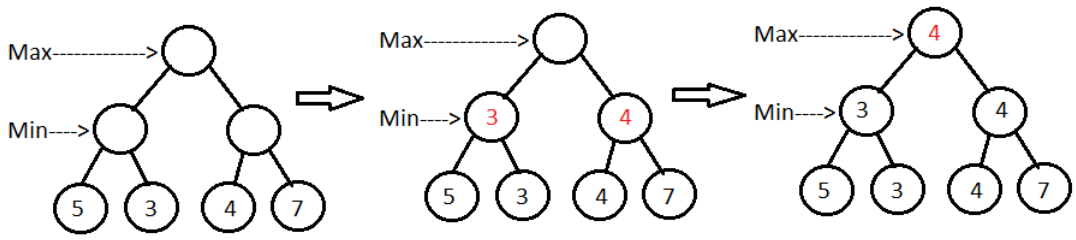



Figure 2: MiniMax example

## 3.2 MiniMax with Alpha-Beta Pruning

Alpha beta pruning is an algorithm that reduces the number of nodes explored by pruning branches that should not be important. To do this, the algorithm traverses the tree in depth-first way until it reaches a leaf or the maximum depth. Once the leaf has been reached, the score of the leaf is evaluated, which is sent back to the parent node. At each node level, we alternate between selecting the minimum and maximum value in order to simulate a two-player game (the opponent wants to minimize the gain of the other). When we arrive at a junction that cannot be better than what we already have, we stop exploring it. This prunes the tree and reduces the number of nodes explored.

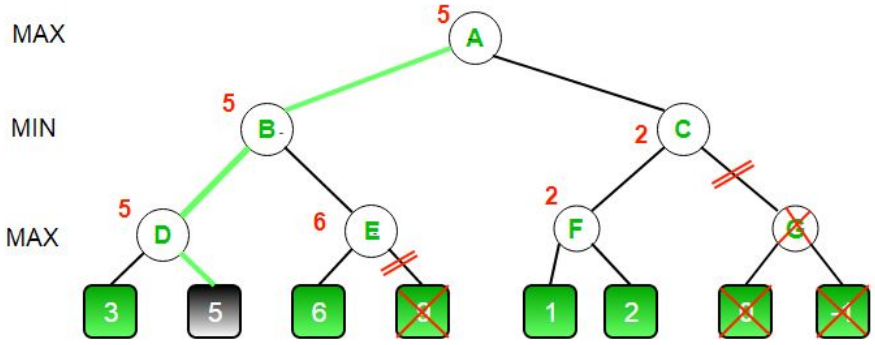



Figure 3: MiniMax with Alpha-Beta example

We use pruning to allow the algorithm to reduce the amount of time needed to compute the tree. So it allow us to go deeper in the tree for the same amount of time.

# 4 Results

Now that the notions and concepts have been explained and implemented, we can play with these parameters to evaluate their influence on the solution. Therefore determining which are the most relevant options for the algorithm.

## 4.1 Influence of Heuristic

A first parameter to be evaluated is the heuristic. To assess its impact, several runs are launched between 2 identical AIs, except that only one of the two uses the heuristic.

If we plot the advantage[3] of pawn between the AI with and without heuristic during the game for the different runs. Then we obtain the figure 4, clearly showing the importance of the heuristic on the power of the AI. We can notice that the heuristic disadvantages the AI a little at the beginning of the game because it slightly restricts its movements (avoids dangerous positions). However it pays well in the long run.

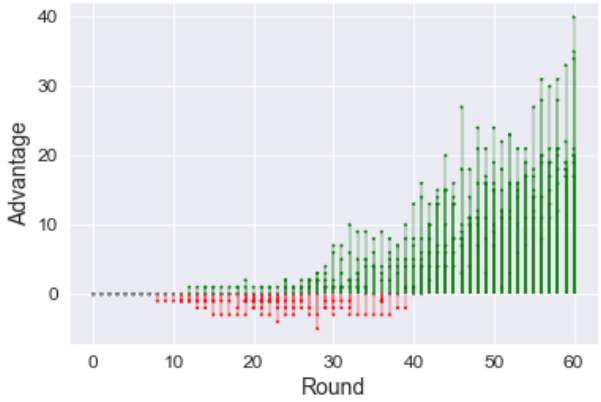



Figure 4: Average pawn advantage between AI with (green) and without (red) heuristic.

Heuristics therefore considerably improve the performance of AI, but at which cost ?

The average playing time per turn for different depths (figure 5), suggests that the impact of the heuristic on the thinking time is not considerable. However, the additional time of the heuristic becomes more and more important with the depth. This phenomenon is logical since the heuristic is computed at each leaf and the number of leaves increases exponentially with the size of the tree.

However, this additional computation time is well worth since it significantly increases the quality of the solution.

| Depth | Heuristic | |
|---|---|---|
| | Without | With |
| 1 | 0.002 | 0.002 |
| 2 | 0.007 | 0.007 |
| 3 | 0.028 | 0.035 |
| 4 | 0.133 | 0.146 |
| 5 | 0.282 | 0.390 |
| 6 | 1.493 | 1.887 |
| 7 | 5.650 | 10.734 |

Figure 5: Average time by round between AI with (green) and without (red) heuristic for different depths.

[3]Advantage is the difference in pawns between the two players.

## 4.2 Influence of Cost Function

A second parameter to be evaluated is the cost function. For this criterion, only the quality of the solution matters since the computation time is the same because the cost functions are dual to each other. The graph 6 seems to show that the best cost function is the "max".

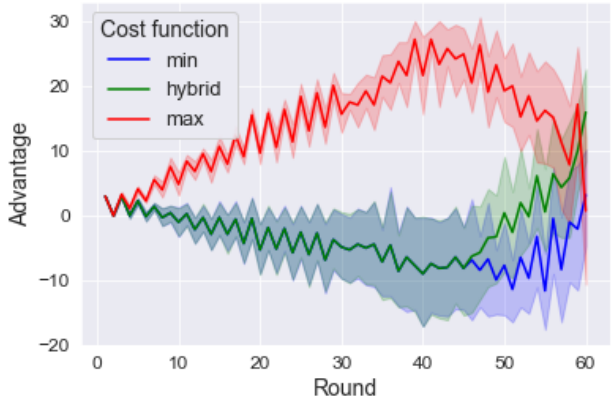
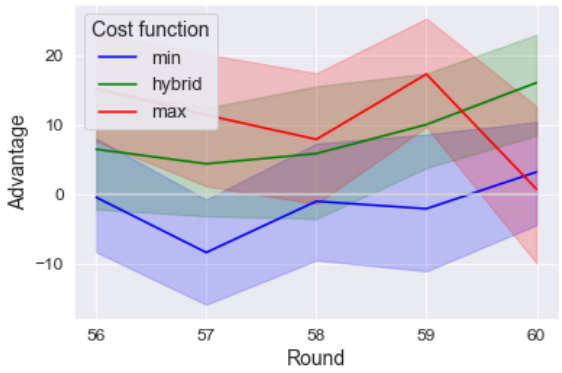



Figure 6: Average pawn advantage during the game for the different cost functions

However, if we zoom on the last rounds (figure 7), we see that the advantage of "max" function goes down a lot, below "hybrid" and even arrives at an efficiency similar to the min.

This proves the minimization technique, described earlier, makes it possible to restrict the adversary to play worse moves and thus gaining an advantage for the end of the game. Therefore, "hybrid" is a good cost function, it gives to the opponent a bit of a lead to better catch him at the end which is the most important part of this game.




| Cost function | Round | Score |
|---|---|---|
| hybrid | 60.0 | 16.00 |
| max | 60.0 | 0.75 |
| min | 60.0 | 3.20 |

Figure 7: Zoom on the endgame of the average advantage for the different cost functions

## 4.3 Influence of Pruning

A third interesting parameter to evaluate is the impact of pruning. Pruning reduces the number of nodes to explore, and therefore saving time. However, from a quantitative point of view, how important is this time saving ?

Figure 8 unsurprisingly shows that pruning reduces AI computation time and that this reduction is exponential[4] with tree depth. The bigger the tree, the more it can be pruned.

Therefore for this game, pruning only begins to be useful from a depth of 4 and becomes particularly beneficial for significant depths. For example, at a depth of 6, pruning makes it possible to gain one depth (calculation time equivalent with depth 5).

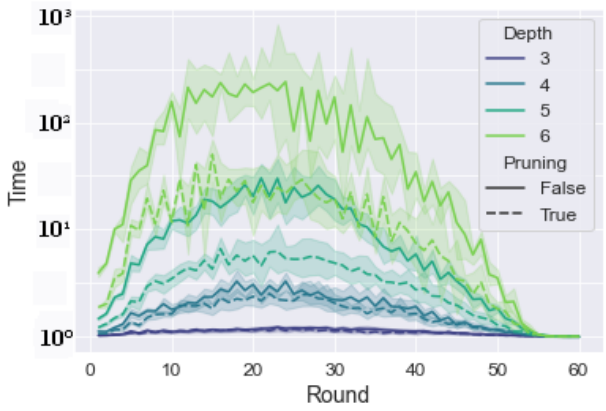



Figure 8: Average time by round between AI with and without pruning for different depths.

One important thing to check with pruning is that it does not reduce the quality of the solution. The graph 9 shows that pruning does not deteriorate the solution even against a random AI which will potentially play in the pruned branches. Indeed, because of the pruning, the AI will only look at the "worst case" (assumes the opponent plays the best moves), so if the opponent plays an unexpected move, it could penalize the AI but we can see in this schema that it is not the case.

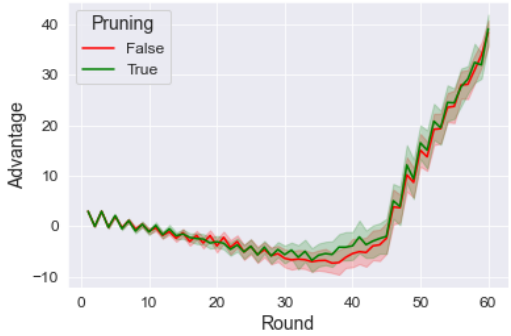



Figure 9: Average advantage of AI with and without pruning against random AI

## 4.4 Influence of Increasing Depth

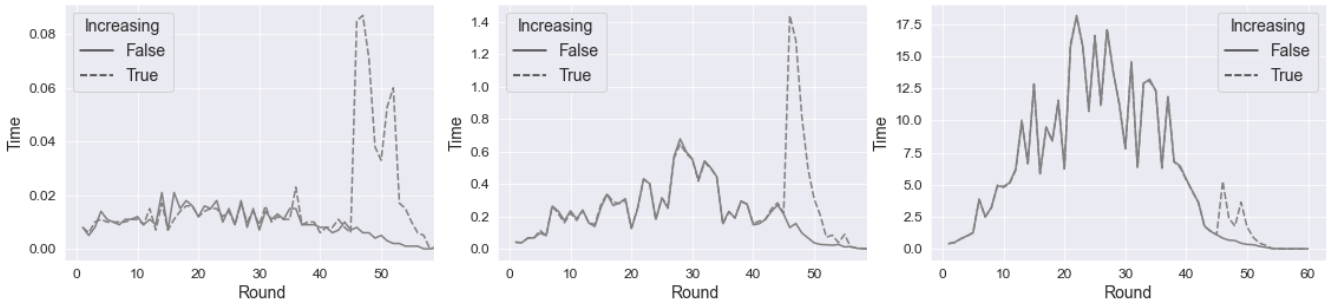
The last parameter to evaluate is the influence of increasing the search depth at the end of the game (where the number of possible moves decreases drastically). As shown in figure 10, its impact on computation time depends on the depth of the tree.

Indeed, adding 2 depth, on a tree where the depth is low (figure 10b) will considerably increase the calculation time. However, the search for such a small tree is fast, so this increase will be "negligible" (¡ 1 sec).

On the other hand, adding 2 depth, on a tree where the depth is high will only have a slight impact on the average calculation time. As shown in the figure 10c, by adding 2 depth on turn 45, we only slightly

[4]The time scale is logarithmic in base 10. This means that each unit multiplies the value by ten ($1 \rightarrow 10, 2 \rightarrow 100$).

increase the calculation time compared to the time needed in the middle of the game. We even see that we could increase the depth more and earlier in the game to further improve the AI.




(a) Depth 2 (b) Depth 4 (c) Depth 6

Figure 10: Influence of adding 2 layers of depths at the $45^{th}$ turn for different depths

Finally, the evaluation of the quality gain brought by the addition of 2 layers of depth in the $45^{th}$ round, is not negligible as shown in figure 11. It is logical to assume that the improvement in quality increases with the number of depths added. And as seen previously, it is possible to optimize even more and add more than 2 layers of depth to further improve the AI.

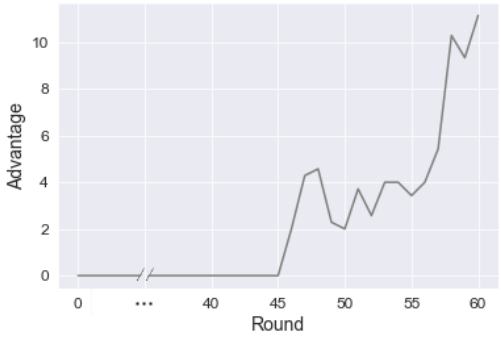



Figure 11: Average advantage of AI with and without adding 2 layers of depth at round 45

# 5 Improvements & Further Research

This project aimed to implement a game with an AI. And of course, there are still many ways to improve it if students want to take this project in the years to come.

For example, in what has already been implemented, the heuristic is particularly effective. However, we have implemented a single trivial heuristic. It might be worth trying and comparing with other heuristics to get even better AI. Another enhancement to an existing feature would be to dynamically vary the depth of the tree search to constantly get maximum AI power in a limited amount of time.

Finally, new improvements are welcome to improve the AI. Whether it is the implementation of other algorithms (like the Monte-Carlo Tree Search), code optimization or a nice graphical interface.

# 6 Conclusion

To conclude, understanding the algorithms used as well as knowledge and analysis of the game are key elements for designing an AI and improving it intelligently.

Indeed, a good understanding of the game makes it possible to design good heuristics and an appropriate cost function. Therefore drastically improves the quality of AI solutions.

Then, the optimization of algorithms makes it possible to reduce the computation time. A classic example is tree pruning, which greatly reduces the number of possibilities to explore, while avoiding deteriorating the solution. This technique is most effective on deep trees, where it is most needed.

Finally, design engineering, coupled with a good algorithm knowledge and game analysis, is always a good feature to stand out and further improve the algorithm. A simple idea for a MinMax algorithm is just to dynamically vary the search depth to use the algorithm every time to its full potential within a limited time.