Report LINFO1361: Assignment 2

Group N°13 (Moodle and Inginious)

Student1: Mathis Delsart

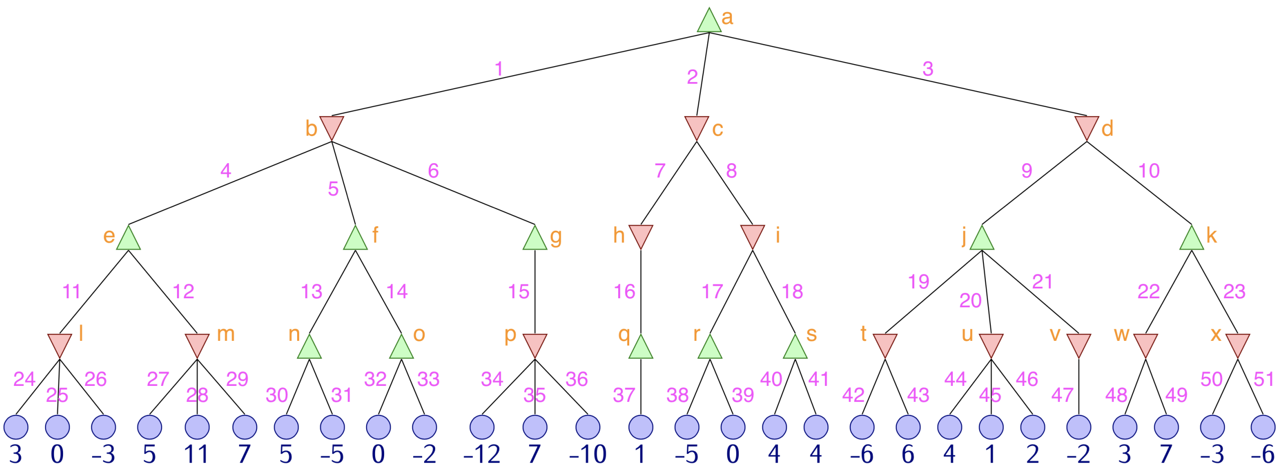
Student2: /

April 23, 2024

Answer to the questions by adding text in the boxes. You may answer in either **French or English**. Do not modify anything else in the template. The size of the boxes indicate the place you **can** use, but you **need not** use all of it (it is not an indication of any expected length of answer). **Be as concise as possible! A good short answer is better than a lot of nonsense!**

1 Exercises (5 pts)

The following figure assigns a unique letter to each node, and a unique number to each branch. Use it to answer the following questions.



1. Perform the MiniMax algorithm on the following tree, i.e. put a value to each node. What move should the root player do? **(1 pt)**

   Assign a numerical value to each node, and indicate the move (i.e. 1, 2, or 3) to perform:

a: 1	**f:** 5	**k:** 3	**p:** -12	**u:** 1
b: -12	**g:** -12	**l:** -3	**q:** 1	**v:** -2
c: 0	**h:** 1	**m:** 5	**r:** 0	**w:** 3
d: 1	**i:** 0	**n:** 5	**s:** 4	**x:** -6
e: 5	**j:** 1	**o:** 0	**t:** -6	**Move:** 3

2. Perform the Alpha-Beta algorithm on the same tree. At each non terminal node, put the successive values of α and β. Cross out the arcs reaching non visited nodes. Assume a left-to-right node expansion. **(1 pt)**

   Indicate the successive α and β values of each node in the table below. Separate successive values by a comma (,). Indicate at the bottom the identifiers of the branches that are cut (in increasing order, separated by a comma) (indicate only the branches where the cuts happen, i.e. don't *indicate the branches that are below a cut).*

Node	α values	β values
a	- ∞, -12, 0, 1	+ ∞
b	- ∞	+ ∞, 5, -12
c	-12	+ ∞, 1, 0
d	0	+ ∞, 1
e	- ∞, -3, 5	+ ∞
f	- ∞, 5	5
g	- ∞, -12	5
h	-12	+ ∞, 1
i	-12	1, 0
j	0, 1	+ ∞
k	0, 3	1
l	- ∞	+ ∞, 3, 0, -3
m	-3	+ ∞, 5

Node	α values	β values
n	- ∞, 5	5
o	/	/
p	- ∞	5, -12
q	12, 1	+ ∞
r	-12, -5, 0	1
s	-12, 4	0
t	0	+ ∞, -6
u	0	+ ∞, 4, 1
v	1	+ ∞, -2
w	0	1
x	/	/

Cuts: 14, 23, 31, 41, 43

3. Do the same, assuming a right-to-left node expansion instead. **(1 pt)**

Node	α values	β values
a	- ∞, 1	+ ∞
b	1	+ ∞, -10
c	1	+ ∞, 0
d	- ∞	+ ∞, 3, 1
e	/	/
f	/	/
g	1	+ ∞
h	/	/
i	1	+ ∞, 4
j	- ∞, -2, 1	3
k	- ∞, -6, 3	+ ∞
l	/	/
m	/	/

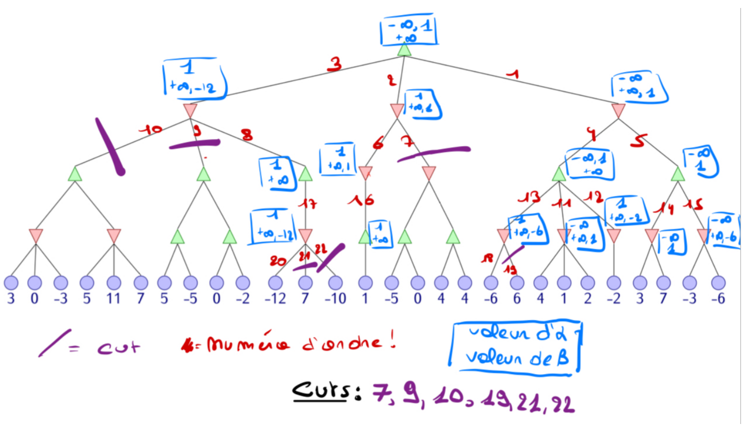
Node	α values	β values
n	/	/
o	/	/
p	1	+ ∞, -10
q	/	/
r	1	4
s	1, 4	+ ∞
t	1	3, -6
u	-2	3, 2, 1
v	- ∞	3, -2
w	-6	+ ∞, 3
x	- ∞	+ ∞, -3, -6

Cuts: 4, 5, 7, 34, 35

4. Is there a node ordering that can lead to a more optimal pruning of the tree (in the sense where the algorithm prunes more branches than in the two other considered cases)? If no, explain why. If yes, give a new node ordering and the resulting new pruning. **(1 pt)**

   Insert an image below containing the reordered tree, with successive α/β values indicated next to each node, and where the branches that are cut by the algorithm are crossed out. This may either be an edited version of `minimax_empty.png` *(using paint, gimp, etc.), a photograph of a drawing you made by hand, etc. In any case, the image must be* ***clear*** *in order to be graded.*

Yes, there is a **node ordering** that leads to a more optimal pruning of the tree.
For the maximizing player, nodes should be ordered in **decreasing order**, ensuring the best node is explored first. Conversely, **for the minimizing player** (the opponent), nodes should be ordered in **increasing order**, prioritizing their best move.
The **goal of this ordering method** is to **always visit the best move available for the player currently taking their turn**.



5. How does Alpha-Beta need to be modified for games with more than two players? **(1 pt)**

To modify Alpha-Beta for games with more than two players, the basic Minimax algorithm must first be **generalized into the MaxN algorithm**. In MaxN, players take turns making moves and aim to **maximize their perceived returns, while being indifferent to the returns of other players**. At the leaf nodes, the evaluation function is applied, now **returning an N-tuple of values**, where N is the number of players. Each component corresponds to the estimated merit of the position for one of the players. **When it's player i's turn, they choose the i-th component of the N-tuple from a child node that maximizes this value**.

Additionally, the pruning principle needs to be generalized, thus adapting the Alpha-Beta algorithm. It's crucial to establish **upper bounds for the total sum of components of each value tuple**, as well as **lower bounds for each individual component**. Ensuring that **each individual value cannot exceed a certain lower limit**, typically **set at zero**, is also necessary to avoid unbounded values that could lead to excessive computations.
Now, there are **two forms of pruning**. The first form, called **immediate pruning**, occurs when **player i is to move** and the **i-th component of one of its children equals the upper bound on the sum of all components**. In this case, **all remaining children can be pruned** because **no child's i-th component can exceed the upper bound on the sum**. Then, there is **shallow pruning**, a more complex situation. For example, in a three-player game, evaluating a node involves establishing upper and lower bounds on the remaining components, depending on the player involved. If the upper bound of a child is lower than the lower bound of the previous child, the latter and its children can be pruned.

<u>**Source:**</u> $https://faculty.cc.gatech.edu/thad/6601-gradAI-fall2015/Korf_{M}ulti-player-Alpha-beta-Pruning.pdf$

2 Shobu (35 pts)

2.1 Alpha-Beta agent (4 points, to be submitted on Inginious)

2.2 Monte-Carlo Tree Search agent (5 points, to be submitted on Inginious)

2.3 Warm-up questions (3 points)

1. What is the branching factor at the start of the game? What is the mean empirical branching factor? (The branching factor is considered here as the number of possible moves that a player can do from a given state). **(1 point)**

The game Shobu is characterized by its **symmetry about the board**, meaning the initial moves for players on both boards are equivalent. Therefore, we'll **analyze a passive move on one board and an aggressive move on the other, multiplying the result by 4** to account for this symmetry for both types of moves.

On one of the boards: vertical move (1 square) $\Rightarrow 4 \times 4 = 16$, vertical move (2 squares) $\Rightarrow 4 \times 4 = 16$, diagonal move (1 square) $\Rightarrow 2 \times 3 \times 3 = 18$, diagonal move (2 squares) $\Rightarrow 2 \times 2 \times 2 = 8$.
Thus, for the start of the game only, $b = 4 \times (16 + 16 + 18 + 8) = 232$.
We **confirmed the result using "len(game.actions(game.initial))"**, and **found the same result** (= 232).

To **estimate the mean branching factor**, we **simulated 100,000 games randomly.**
For each game, we **calculated the branching factor and then took the average of these values**. By the **law of large numbers**, we observe that **the result converges** to $b = 85$.

2. What would be one (of the many) drawbacks of the simple heuristic that is imposed for the basic alpha-beta agent above? **(1 point)**

One of the main drawbacks of this simple heuristic is its profound **lack of knowledge about the game**.

By solely considering the minimum number of pieces on the boards, it **completely overlooks crucial aspects**.

It fails to take into account **imminent threats**, **defensive** and **offensive structures**, and **does not differentiate** between **advantageous or disadvantageous situations** due to the **absence of consideration** for the **quality of piece placement**. By example, it does not take into account the **central positions** (the 4 middle squares) **as advantageous** and it makes **no distinction** in strategy between ***Home*** **and** ***Opponent*** **boards**.

Additionally, it lacks the ability to **distinguish between different game states** that may have **the same minimum number of pieces** and thus the **same score**, resulting in **suboptimal decision-making**.

A good heuristic try to give a different score for each single state !

See *question 2.4* for the optimisation of this simple heuristic.

3. Considering the Monte-Carlo Tree Search algorithm, what is the hypothesis supporting the use of random simulation to estimate the win rate? Is this hypothesis always valid? **(1 point)**

The **fundamental assumption** underlying the **use of random simulations** in the MCTS algorithm is the **law of large numbers**. This mathematical law states that **with a sufficiently large number of simulations**, the **estimated win rate should converge to the true win rate of the game**.

Mathematically, this law can be expressed as follows:

$$\lim_{n \to +\infty} \mathbf{P}\left(\left|\overline{X}_n - \mu\right| > \varepsilon\right) = 0 \quad \forall \varepsilon > 0$$

In this formula, n represents **the number of samples (simulations)** performed, X_i is a random variable representing the **result of the i-th simulation** (where **1 indicates a win, -1 a loss and 0 a draw**), $\overline{X}_n$ is the **average of the simulation results**, μ is the **mathematical expectation of X_i**, and ε is a **small margin allowed from μ**.

It is important to note that for the law of large numbers to be applicable, a **sufficiently large number of samples must be used**. Otherwise, the **estimated win rate may not be representative of the true win rate of the game**.

However, **this hypothesis is not always valid**, particularly in games where the **game tree is too large to adequately sample with random simulations alone** like the game *Shobu*. Another case where this assumption is invalid is when there are **very abrupt and extreme changes in state evaluations**. In this scenario, the convergence of the sample mean to the true mean can be compromised. However, this is not the case for the game Shobu. In such cases, the effectiveness of the MCTS algorithm may be limited.

2.4 Description of your agent (8 points)

Describe in the boxes below what you have implemented for your contest agent. You can also mention things that you tried and did not work but focus on what you have submitted in the end.

2.4.1 Algorithm Used

We implemented the **MiniMax algorithm with alpha-beta pruning**. This choice was made after comparing the two basic versions (AlphaBeta and MCTS), and observing that MCTS performed poorly in the Shobu game. Moreover, **MiniMax seemed simpler to understand and optimize**.

2.4.2 Heuristic

Here are the features considered in our heuristic guiding our tree search. Everything detailed below was done for both players. When something is **advantageous for the AI**, we **assign a positive score**, while if it's **advantageous for the opponent**, we **assign a negative score**. This allows MiniMax to **maximize the AI's score and minimize the opponent's score** during the search.

- ***Material advantage***

We considered the **minimum number of pieces remaining on either board**, as well as the **total number of pieces remaining on all four boards**.

- ***Positional advantage***

We **favored central positions** (the 4 center squares). A **bonus** is also given if these **central positions are reached on the opponent's boards**, as actions can still be taken on both opponent boards, allowing **more attacking options**. A **bonus** is also awarded if the **central position is on the opponent's side** (thus, 2 out of 4 squares), as it's a strategically advantageous attacking position.

- ***Mobility advantage***

We simply calculated the **number of possible moves for a state for both players**.

- ***Strategic advantage***

This feature encompasses several sub-features, all representing strategic advantages through positioning. We assign a score to **stones located on the opponent's side** (position > 7 for white player), favoring attacks. We also give a significant bonus for a **specific attacking position** that we found interesting to implement. For example, if the white player is at position 14 (resp. 13) and there are opponent stones on squares 13 and 15 (resp. 12 and 14), the win percentage for white player in this case is very high because it's a **strategically crucial position**. A **high score was assigned to this special position**. And smaller scores were still assigned if there's only one opponent stone on each side of the white stone. The last strategic feature **assigns a score to stones placed between two opponent stones in the same direction**. This position is interesting because the **player's stone can push both opponent stones, but the opponent stones cannot push the player's stone**!

To ensure that all features are on **the same scale**, we searched for the maximum and minimum values each feature can take, and applied this formula: $valueScaled = \frac{valueNotScaled - minValue}{maxValue - minValue}$. This yields scores **between 0 and 1**.

We then **manually determined the weights for each feature to obtain the best possible evaluation**. We had considered **dynamically adjusting the weights using a deep learning algorithm**, but we lacked the time (and talent) to implement it.
The chosen weights are **0.50, 0.25, 0.20, and 0.05, respectively, for material advantage, positional advantage, strategic advantage, and mobility advantage**.

2.4.3 Node Rearrangement

We initially implemented a **complete node rearrangement function using the evaluation function**. However, the **computation time was excessively high**, so we abandoned the idea and opted for a less optimal but much more accessible strategy in terms of computation time. This strategy is to **place all actions leading to pushing an opponent stone out of one of the boards first in the action list**. This feature doesn't make much sense initially, but eventually, it allows for **more pruning using this node rearrangement function because nodes likely to lead to more advantageous game states for the player are explored first by the algorithm**.

2.4.4 Iterative Deepening

Since **time was the constraint of the algorithm** (one game = 300 seconds for our agent), we opted for an **iterative deepening algorithm while keeping track of the best action from the previous iteration to place this action first in the action list in the next iteration**. This technique is a node rearrangement technique as discussed in the previous section. It's often referred to as a ***killer move heuristic***. The first depth used is 1 and then the depth in increased by 1 at each iteration. **If the maximum time allowed for an iteration is reached, then the algorithm stops and returns the best action found among all iterations performed**. With this algorithm, we can **avoid for sure the time-out**, provided that we have our maximum time allowed for a single iteration not too long. **We estimate it at 15 seconds**.

2.4.5 Transposition Table

We implemented a **transposition table** taking into account the symmetries of the game discussed in the next section.
The transposition table allows us to **retrieve the evaluation of a state if this game state has already been encountered and evaluated before**. To store the state in memory, we **hash it into a unique string by scanning from left to right and bottom to top of the board**. A "." represents an empty square, "o" represents a white stone, and "x" represents a black stone. The initial state of the game is represented as follows: "oooooooo................xxxxxxxxxxxxxxxx................oooooooo".
This improvement **consumes a lot of memory** but is **very useful for avoiding recalculation of many parts of the tree already calculated before**.

2.4.6 Shobu Game Symmetry

The Shobu game exhibits a **lot of symmetry**. Therefore, we created a class that takes a **game state hashed as input** and **outputs whether a symmetry of this state already exists in the transposition table**. If a symmetry exists, then the evaluation of the state is already known and doesn't need to be recalculated! This improvement is therefore only used with the transposition table discussed in the previous point.

We considered **various symmetries**. The simplest are the **rotations of the 4 boards individually**. Each board **can rotate at 90° / 180° or 270°**. *Note that all boards must rotate by the same amplitude.* There are also **switches between the two Home boards, the two Opponent boards, or even both switches simultaneously**. *Note that Home and Opponent boards cannot be modified between each other, as this would change the game properties.* Additionally, for each board taken individually, there are **vertical, diagonal, and inverse diagonal symmetries** (horizontal already accounted for during the 180° rotation). The last symmetry is the **color change of stones**. Note that in this symmetry, the evaluation value is inverted (17 becomes -17 and vice versa)!
This improvement allows us to **avoid a large part of the search tree because symmetries in this game are omnipresent due to the idea of the 4 distinct boards in the game**.

2.4.7 Features Ideas Not Implemented

- ***Quiescent search***

In our iterative deepening algorithm, we tried to implement **quiescent search to enhance the search process**. Quiescent search is particularly beneficial for **resolving the horizon effect problem**. This technique **extends the search to capture tactical positions where the game is relatively stable, ensuring that the evaluation captures critical tactical threats and captures**.
Unfortunately, we have not observed good results. Either we made errors in the implementation, or this feature is not useful for the game Shobu.

- ***EndGame Lookup table***

We had considered implementing a **lookup table for endgames** as it's often used in chess since endgame moves are often known because the **branching factor is much smaller**. This would have allowed us to save a lot of time and **enable deeper searches in the middle of the game**.
However, we didn't do it because there was almost no documentation on this topic for the Shobu game, making it practically impossible.

2.5 Comparison of agents (5 points)

Describe here the comparison of the different agents: random, Alpha-Beta, MCTS and your agent (and others if you want!). Remember that it should be a statistical comparison. Describe how you compare the agents. Draw some observations and conclusions based on the results you have obtained.

2.5.1 Introduction

For the comparison of our four agents, we conducted three tests that we consider relevant. For each of these tests, we will present the results (table or graphs) and provide interpretations of the results.

2.5.2 Test 1: Win Rate

In this test, each agent played against every other agent. This test was conducted under fair conditions, meaning each agent played 10 games, with 5 as the white player and 5 as the black player. Figure 1 shows the results in the form of a heatmap.

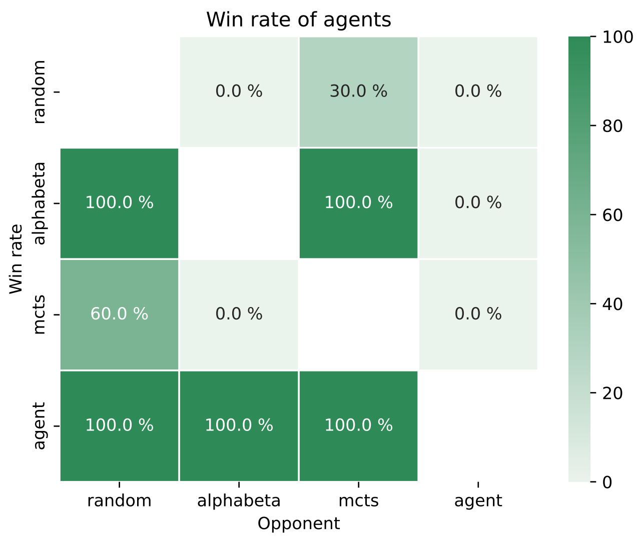
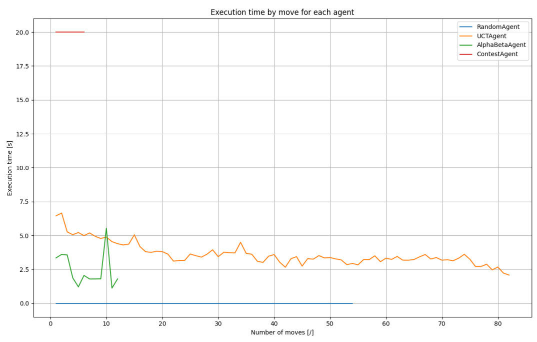


Figure 1: Win Rate HeapMap

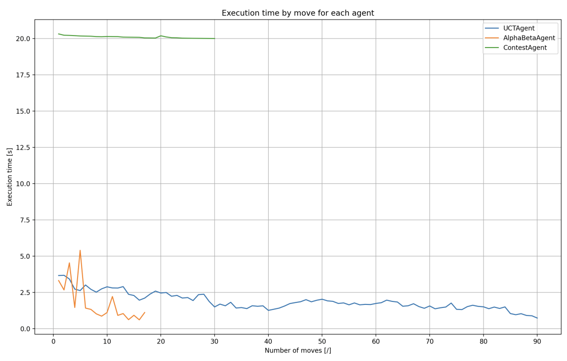
We observe that our agent emerges victorious in all games against the other agents. The AlphaBeta agent, on the other hand, suffers defeat in all games against our agent. Additionally, the MCTS agent only manages to win 60 percents of the games against the random agent, indicating poor performance in the Shobu game. This could be attributed to the fact that MCTS simulations are too slow, thereby limiting its ability to explore a substantial number of nodes within the given time frame. Consequently, it fails to gather sufficient information to make informed decisions, akin to the random agent's approach. Thus, it's not surprising that its win rate is relatively low. The Alphabeta (and thus our Contest agent as well) is significantly superior to MCTS because Shobu is a game with complete information, and Alphabeta's search is guided by a heuristic. These two conditions make MCTS perform poorly in such scenarios (without notable improvements).

2.5.3 Test 2: Execution Time

For this test, we analyzed the execution time per move for each agent when playing against a random agent and against themselves. This test also allows us to observe, for each agent, the number of moves required to finish the game against a random agent and against itself. We did not have the random agent play against itself because it is not very meaningful, and we limited the maximum number of moves to 100 for MCTS as it acts almost like a random agent and could technically make an infinite number of moves. Figure 2 shows the results obtained in graphical form.



(a) Against Random Agent



(b) Against Themselves

Figure 2: Execution Time by Move for Each Agent

- **Interpretation of Figure 2.a**

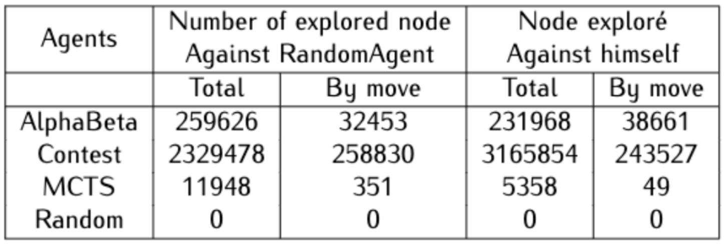
We can observe that the execution time of our Contest agent is constant at 20 seconds, which is normal as it is the allocated time limit per move that we have set for maximum calculation time. We can also see that the MCTS agent is very slow to converge as it takes more than 80 moves to finish the game compared to less than 60 moves for the Random agent. The execution time for the Random agent is constant at 0 seconds because it directly returns a random legal action, so the calculation time is negligible. The last significant observation here is the comparison of the number of moves to finish the game against the Random agent for our agent and for the AlphaBeta agent. The AlphaBeta agent finishes the game in a higher number of moves, demonstrating that our agent plays better and finds optimal moves more easily to finish the game quickly.

- **Interpretation of Figure 2.b**

We observe that MCTS does not converge because it reaches the limit of 100 moves allowed for the test. The execution time of our agent remains constant at 20 seconds, as explained in the previous section. We can also see that the number of moves in the game is lower for the AlphaBeta agent compared to our agent. This is entirely normal because our agent strives to develop strategies, to position itself in the center, whereas the AlphaBeta agent simply aims to eliminate pieces without considering its positioning. Consequently, it finishes games more quickly, but it also takes a considerable risk as it never adopts a defensive strategy; it only attacks.

2.5.4 Test 3: Explored Nodes

For this third test, we calculated the number of nodes explored by each agent when playing against a random agent and against itself. Additionally, we took into account the number of moves in the game to demonstrate the number of nodes explored per move. The following figure 3 shows the results obtained.



Agents	Number of explored node Against RandomAgent		Node exploré Against himself	
	Total	By move	Total	By move
AlphaBeta	259626	32453	231968	38661
Contest	2329478	258830	3165854	243527
MCTS	11948	351	5358	49
Random	0	0	0	0

Table 1: Explored Nodes for each agents

Figure 3: Explored Nodes Table

We observe that our agent explores many more nodes than the other agents, which could contribute to its ability to make more informed decisions and therefore achieve better results in the game. The AlphaBeta agent explores a significant number of nodes, but it remains much lower than our Contest agent. The MCTS agent, on the other hand, explores very few nodes due to the slowness of its simulations, which may limit its ability to achieve satisfactory results within a reasonable timeframe. Additionally, the Random agent does not explore any nodes and simply makes random decisions, that's why it's very bad. Thus, the number of nodes explored plays a crucial role in the quality of decisions made by each agent and affects their speed of convergence towards optimal or near-optimal solutions in the game.

2.5.5 Conclusion

These three tests demonstrate that our agent outperforms the MCTS, AlphaBeta, and Random agents, which is not surprising since they are basic algorithms without improvements. Our agent is much slower to maximize calculation time but is more optimal in its decision-making. It also explores many more nodes than the other agents, showing that it acquires better knowledge of the search tree with each move.

2.6 Note

My partner hardly did anything. So, I didn't include his name. It wasn't an oversight. I sent you an email explaining the situation (and to the professor as well). For this assignment, he did questions 1.2, 1.3, and the HeatMap graph for this question. That's all... Everything else is mine, including all the code.

2.7 Contest (10 points, to be submitted on Inginious)