# COMP 424 Final Project Game: Reversi Othello

Fardin Abdullah, Kazi Ashhab Rahman
Comp 424, Fall 2024

## 1. Introduction

In this project, we aim to develop an AI agent for Reversi Othello, a strategic two-player game played on an M×M board, where M is an even number. The game revolves around players taking turns placing discs on the board to capture their opponent's discs by enclosing them. To determine the optimal approach for our AI agent, we evaluated two prominent algorithms: Minimax and Monte Carlo Tree Search (MCTS), both of which are well-suited for decision-making in games.

Minimax operates by systematically evaluating all possible moves up to a certain depth, alternating between maximizing the agent's chances of winning and minimizing those of the opponent. This method offers robust gameplay, particularly against skilled opponents, as it carefully considers the best counter moves an adversary might take. However, the computational demands of Minimax can be high. On the other hand, MCTS estimates winning probabilities by simulating numerous random games for each move. While MCTS is less computationally intensive and excels in exploratory scenarios, it relies on random sampling, which can lead to inconsistent results.

After careful consideration, we selected Minimax as the core algorithm for our AI. Its deterministic nature and strategic depth make it particularly effective for Othello. When enhanced with alpha-beta pruning, Minimax becomes computationally more efficient while retaining its methodical precision. Moreover, it can be seamlessly integrated with game-specific heuristics, such as mobility and corner control, to further refine decision-making. This combination enables the agent to avoid losing critical positions and execute strategies with greater consistency and accuracy, making it a superior choice for our implementation.

## 2. Executive Summary

Our approach to developing the Minimax-based AI agent involved an iterative, step-by-step refinement process focused on improving both efficiency and performance. We began with a straightforward implementation of the Minimax algorithm, paired with a basic evaluation function that calculated the difference between the agent's and opponent's disc counts. While functional, this initial version struggled with efficiency, particularly on larger boards (e.g., 12×12), as the exponential growth of possible game states with increasing depth made it prohibitively slow.

To address these inefficiencies, we introduced **Iterative Deepening Search (IDS)**, which allowed the agent to incrementally explore deeper game states while adhering to a strict 1.9-second time limit per move. If the time limit was reached, the agent would fall back to the best move identified in the last completed depth, ensuring responsiveness without sacrificing quality. We further enhanced the Minimax algorithm by integrating **Alpha-Beta Pruning**, an optimization technique that prunes irrelevant branches of the search tree—those that cannot affect the final decision. By maintaining two bounds—**Alpha** (the best-guaranteed score for the maximizing player) and **Beta** (the best-guaranteed score for the minimizing player)—this optimization drastically reduced the number of board states evaluated, enabling the agent to explore deeper depths while focusing computational resources on the most promising moves.

To elevate the agent's decision-making capabilities, we improved its evaluation function with advanced heuristics, including **Corner Control**, **Mobility**, **Blocking**, and **Disk Difference**. Additionally, we dynamically adjusted the weights of these heuristics to adapt to different stages of the game. In the **early game**, the focus is on maximizing mobility and restricting the opponent's options through blocking. In the **mid-game**, corner control becomes a key priority as stable positions emerge. In the **late game**, corner control, blocking, and coin parity take precedence to secure stability and maximize the final disc count.

These iterative refinements culminated in a highly competitive agent capable of outperforming simpler agents, including those with static weights or fewer heuristics, as well as random agents, the GPT greedy agent, and human players. By striking a balance between strategic depth, computational efficiency, and adaptability, the agent demonstrates a comprehensive understanding of Reversi strategy and consistently delivers strong performance across a wide range of scenarios.

## 3. Agent Design

### Evolution of the Initial Algorithm:

1. **Initial Simple Minimax**:
   The initial implementation of Minimax evaluated all possible branches, leading to an exponential explosion in the number of nodes explored, especially during the mid-game where move possibilities are vast. While functional, this approach struggled to explore meaningful depths within the allotted time.

2. **Introduction of Alpha-Beta Pruning**:
   To address the branching factor, Alpha-Beta Pruning was integrated to eliminate branches that could not influence the final decision. This significantly reduced computational overhead, enabling the agent to explore deeper levels of the game tree more efficiently. The pseudocode for Alpha-Beta Pruning was adapted from the class notes, which served as a foundational reference for our implementation.

3. **Incorporating Iterative Deepening**:
   Iterative Deepening Search (IDS) was introduced to dynamically manage time constraints. By incrementally increasing the search depth, IDS ensures the agent always returns a move, even under strict time limits. Combined with Alpha-Beta Pruning, it balances exploration depth with time efficiency, ensuring both thoroughness and reliability in decision-making.

The shift from simple Minimax to Minimax with Alpha-Beta Pruning and Iterative Deepening addressed the high branching factor and strict time constraints. Alpha-Beta Pruning improved efficiency by reducing unnecessary evaluations, while Iterative Deepening ensured a move was always returned by dynamically adjusting the search depth. Together, these enhancements, built on concepts from the course material, balance depth exploration and computational efficiency, making the agent robust and effective in complex, time-limited scenarios.

Now let's focus on the heuristics that I have implemented for our agent.

## Corner Control:

The corner heuristic evaluates the control of the four corners on the board, as these positions are stable and immune to being flipped once captured. The code calculates the difference between the number of corners controlled by the agent and those controlled by the opponent. A weight of +10 is assigned for each corner controlled by the agent, while a penalty of -10 is applied for each corner controlled by the opponent. This simple yet effective evaluation emphasizes the importance of corner control, as it provides long-term stability and strategic advantage throughout the game.

## Mobility:

The mobility heuristic evaluates the difference in the number of valid moves available to the agent and the opponent. It directly measures the agent's ability to maintain flexibility and control the flow of the game. Greater mobility allows the agent to adapt to the changing board state and strategically place discs while restricting the opponent's mobility, which limits their ability to make strong moves. This ensures the agent can dictate gameplay and force the opponent into less advantageous positions.

## Blocking:

The blocking heuristic evaluates the agent's ability to restrict the opponent's access to critical areas of the board, such as corners and edges. By preventing the opponent from making moves that grant them a positional or strategic advantage, this heuristic ensures the agent retains control over the game's flow. In our implementation, blocking is achieved by simulating potential opponent moves and penalizing states where they gain access to critical positions like corners. The heuristic assigns a significant negative weight to moves that allow the opponent to secure these advantageous areas.

Blocking is an exceptionally powerful heuristic, rivaling corner control, due to its ability to proactively restrict the opponent's access to key positions, such as corners and edges. By preventing the opponent from gaining stable positions, blocking enhances the agent's flexibility and mobility, forcing the opponent into suboptimal moves. This heuristic also amplifies the effectiveness of other strategies, as restricting the opponent's movement increases the value of mobility and corner control. Used alongside corner control, blocking reinforces the agent's ability to maintain stability and dominate the board, making it a critical tool for dictating gameplay and neutralizing the opponent's strategy.
In addition to corner control, mobility, and blocking, I experimented with coin parity and stability heuristics, but both negatively impacted performance. Coin parity, which measures the difference in the number of discs controlled by the agent and the opponent, prioritizes short-term gains over strategic positioning, making it unreliable for long-term success. Stability, which evaluates the number of discs that cannot be flipped, conflicted with the mobility heuristic, as stabilizing discs often reduced the agent's future move options. Furthermore, stability was already implicitly addressed through corner control, which naturally leads to stable positions. As a result, we excluded stability, focusing instead on corner control, mobility, and blocking for a better balance between flexibility, positioning, and stability. We did however keep coin parity, but only made it significant in the later stages of the game.

## Dynamic Weight Assignment for Heuristics:

Initially, we assigned **static weights** to the heuristics, where each was given a fixed importance throughout the game. However, this approach underperformed, as the relevance of each heuristic varies depending on the stage of the game. For example, corner control and blocking are critical in the late game but less impactful in the early game where mobility plays a larger role. Realizing this limitation, we implemented **dynamically adjusted weights** that change based on the game state, significantly improving the agent's performance.

## Minimization Strategy

Incorporating the **minimization strategy** further enhanced the agent's decision-making during the early game. This strategy prioritizes keeping the agent's disc count low to achieve key advantages:

- **Maximizing Mobility**: A lower disc count increases the agent's valid move options while limiting the opponent's mobility, enabling better positional play.
- **Avoiding Risky Moves**: A smaller disc count reduces exposure to edges and corners, preventing the opponent from exploiting these critical regions.
- **Controlling the Board**: By minimizing flips, the agent preserves flexibility in the board state, forcing the opponent into less advantageous moves.
- **Setting Up Stable Positions**: Fewer early discs make it easier to secure stable positions like corners and supported edges in the mid-to-late game.
- **Forcing Opponent Mistakes**: A smaller footprint compels the opponent to make large flips, inadvertently exposing valuable areas like corners.

This strategy proved most effective during the **early and mid-game**, where maintaining flexibility and controlling the board's evolution are critical. However, it must be balanced to avoid scenarios where the agent has no viable moves or allows the opponent to secure key positions. The minimization strategy completely obliterates opponents who have an aggressive strategy to gain discs early in the game.

## Determining the State of the Game

The state of the game is defined based on the number of empty squares on the board:

- **Early Game**: More than 60% of the board is empty.
- **Mid-Game**: About 20% to 60% of the board is empty.
- **Late Game**: Less than 20% of the board is empty.

We wrote a helper function that allows me to determine the state of the game by simply comparing the number of discs on the board and the total number of available spots on the board. Dynamic weight assignment ensures that each heuristic's significance reflects the strategic needs of the current game stage.

## Heuristic Importance by Game Stage

Dynamic weight assignment tailors the significance of each heuristic to the evolving priorities of the game:

1. **Early Game**:
    - **Mobility** is the most important heuristic, as maximizing valid moves maintains flexibility and enables better positioning for future turns.
    - The **minimization strategy** guides the agent to minimize flips, promoting mobility and forcing the opponent into less favorable moves.
    - **Blocking** plays a supporting role by restricting the opponent's access to critical areas like corners.
2. **Mid-Game**:
    - **Corner Control** gains importance as opportunities to secure stable corners begin to emerge.
    - **Mobility** remains significant for maintaining flexibility and adaptability.
    - **Blocking** prevents the opponent from positioning themselves advantageously, complementing corner control.

3. **Late Game**:

- **Corner Control** becomes paramount, as stable discs in corners often determine the game's outcome.
- **Blocking** complements corner control by ensuring the opponent cannot counter or regain strategic regions.
- **Disk Count** becomes relevant as the focus shifts to maximizing the final number of discs controlled.

By dynamically adjusting heuristic weights based on the game stage and incorporating the minimization strategy, the agent adapts its strategy to the evolving board state. This allows it to prioritize flexibility and mobility in the early game, positional strength in the mid-game, and stability in the late game. This approach resulted in significantly improved decision-making and overall performance.

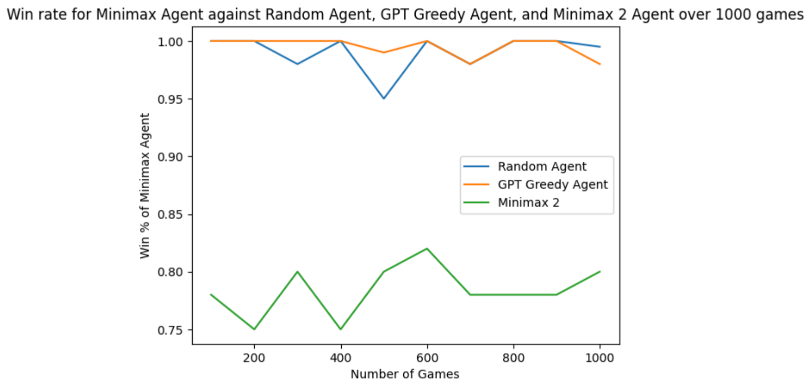
The **evaluate_board** function is the core of the **Minimax with Alpha-Beta Pruning** algorithm, supported by **Iterative Deepening Search (IDS)**. It translates strategic heuristics—such as corner control, mobility, and blocking—into dynamic numerical scores that adapt to the game stage.
Minimax uses these evaluations to identify moves that maximize the agent's advantage and minimize the opponent's, with alpha-beta pruning improving efficiency by skipping irrelevant branches. IDS ensures the search depth increases progressively within the time limit, providing the best possible move at each iteration. Together, this integration ensures that the agent's search is adaptive, efficient, and strategically sound, enabling intelligent gameplay in Reversi.

## 4. Quantitative Performance Analysis

To evaluate the quantitative performance of the Minimax agent, look-ahead depth, breadth, or moves considered at each level, scalability with board size, impact of heuristics, and win rate against opponents are all used. These metrics reflect how successfully it makes decisions and performs competitively.

### Win Rate

To evaluate our agent, we pitted it against a random agent, the GPT greedy agent, and the same Minimax agent but with fewer non-weighted heuristics (Minimax 2) over 1000 games on board sizes 8-12. A thorough analysis is given in the graph below: The agent boasts a near-perfect win rate of 95-100% against both the random and the GPT greedy agent. Against the Minimax 2 agent, the matches were much more competitive. However, the agent still manages to maintain a 70-80% win rate which highlights the effect of the weighted heuristics.



## Depth

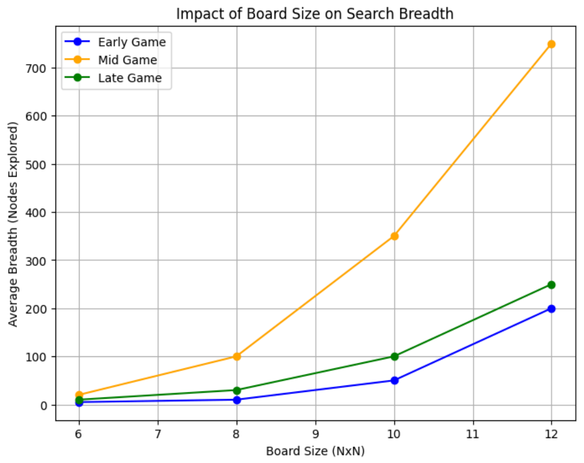
The depth of the agent varies depending on the time set on the IDS and the stage the game is being played on. Early in the game, the agent reaches a depth of 3-7, but in the middle to late stages, since the branching factor increases, the depth reached is slightly reduced to around 2-5. Since the agent uses Alpha-Beta pruning, it focuses on the promising branches and leaves the others. This leads to some branches being explored at a deeper level than other branches. The search depth is thus affected by the time limit of IDS, the stage of the game, and the effect of Alpha-Beta pruning.

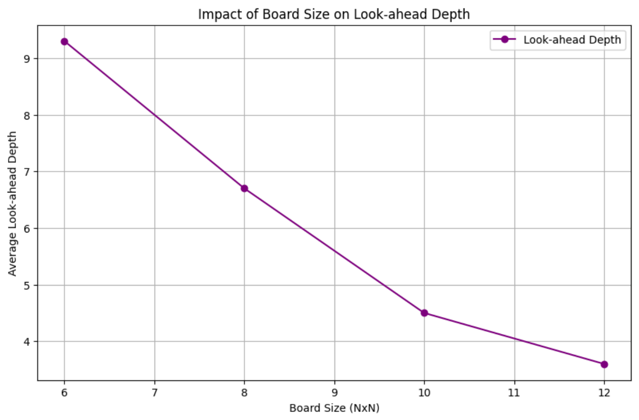
## Breadth

At every level of play, the agent considers a different number of valid moves. In the game's early stages, the agent considers fewer moves (5-10 moves) since much of the board is empty and there are fewer plays to consider. In the mid-game, we can see a significant increase in the breadth (10-500 moves) since the number of possible valid moves increases exponentially. In the later stages of the game, the breadth gets reduced again due to a reduced number of valid spaces on the board. For the maximizing player, the breadth is generally a bit smaller than the minimizing player because since it plays strategically, capturing corners and edges, the moves tend to cluster pieces, which reduces the number of available moves. The minimizing player plays moves that disperse pieces more, which often leads to an increase in available moves. The agent also uses Alpha-Beta pruning, which reduces the breadth by pruning branches that do not affect the final decision. The agent also orders moves based on the heuristics such as prioritizing moves near corners or moves with high mobility. Such heuristics lead to rewarding moves being evaluated early in the game which means more ineffective branches are pruned by the Alpha-Beta pruning mechanism, leading to reduced breadth.

## Board Size

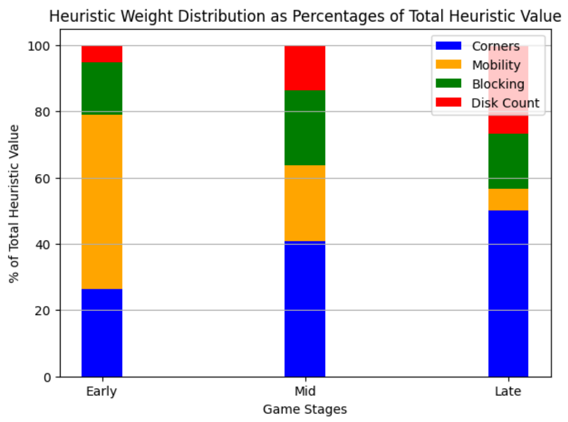
The dimensions of the board influences the search breadth of the agent significantly as well as the time complexity of the agent. As the size of the board increases, the total number of possible positions grows quadratically (e.g. a 6x6 board has 36 cells while a 10x10 board has 100 cells). This larger search space means there will be generally more legal moves available at each step of the game, so the search breadth increases. As the board size increases, the look-ahead of the agent generally decreases because now the agent must evaluate a larger breadth at each level. This limits the maximum depth the agent can reach given the time and computational limitations. The first graph below shows the variation of average breadth of the agent in different stages of the game at different board sizes. As the board size increases, the average breadth generally increases. The maximum breadth occurs during mid-game due to the huge number of choices for moves. The Alpha-Beta pruning makes sure that the breadth does not increase exponentially by discarding the ineffective branches. The second graph below shows that as the board size increases, the look-ahead generally decreases because of the increased breadth the agent has to consider.

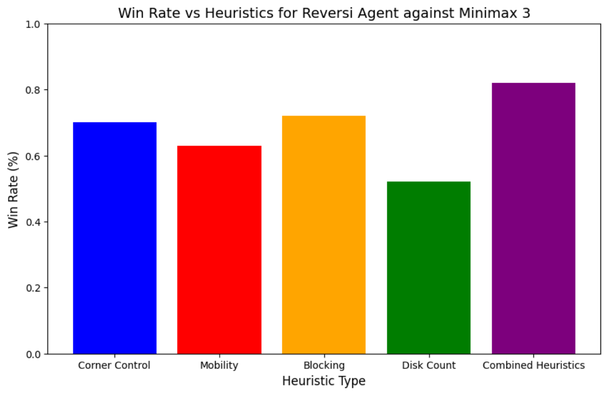




## Heuristics

The minimax agent uses a set of heuristics in its evaluation function which are covered in thorough detail in the **Agent Design** section of this report. The heuristics used are: Corner Control, Mobility, Blocking, and Disc Count. These heuristics are used with different weights in different phases of the game to maximize winning chances, as shown in the first graph below. An analysis on the effect of each heuristic as well as the combined one on the win rate is done against a Minimax agent with only the disk count heuristic (Minimax 3) in the second graph. Here, we can see the effectiveness of each heuristic and we get the best result combining all the heuristics.





## Win-Rate Prediction

When predicting the win-rate for the Minimax agent, it is important to take into account the type of opponents the agent is facing:
(i) **Random Agent** - We predict that the win-rate for the Minimax agent will be **95-100%**, based on multiple games run on autoplay. Since the random agent uses no strategy or logic, it will lose almost all the time.

(ii) **Average Human Player** - An average human player can have decent understanding of some heuristics such as corner control but will struggle with considering the huge number of possibilities that the Minimax agent can evaluate in each step. We predict that the win-rate for the Minimax agent will be around **80-90%**.

(iii) **Classmates' Agent** - The win rate against the classmates' agents will vary depending on their strategies and agent design. It will perform very well against simpler or less optimized agents, but the matches will become competitive if advanced algorithms such as Mininmax, MCTS, etc. are used with good evaluation functions. Since our agent is a strong Minimax algorithm with advanced heuristics and Alpha-Beta pruning and IDS,  we believe it will perform better than most classmates' agents. The predicted win-rate is **70-80%**.

## 5. Pros and Cons

Our Reversi agent is powered by a Minimax algorithm with Alpha-Beta pruning and IDS which ensures the agent makes the most out of its time budget. This keeps the maximum possible time for a move to be less than 2 seconds. Even if the search is interrupted, we get the best move so far. The pruning reduces the number of nodes evaluated in the game tree by skipping branches that cannot influence the final decision.

Hence, the algorithm explores deeper levels of the game tree within the time limit, which ensures the agent can make strategic moves even in situations with multiple legal moves. This approach provides computational efficiency and accuracy in case of time pressure. The use of advanced heuristics in the evaluation function translates board positions in the game into numerical scores for aiding decision-making. Heuristics like corner control, mobility, and blocking strategies especially weighted differently across different phases of the game provides more intelligent play tailored at every step. The weighted heuristics avoids greedy mistakes by preventing over-prioritizing short term gains. Higher emphasis on corner control and stable regions in the end stages of the game makes sure the agent capitalizes on the earlier positional advantages.

While the Minimax agent is competitive and robust, it has certain limitations. Firstly, using IDS limits the agent's ability to evaluate board states because of the time limit of 2 seconds allocated for each move. While we are also using pruning, computational limits still restrict how deeply the game tree can be explored. In complex mid-game situations, the agent might miss deeper strategies that unfold beyond the current depth. The implementation of the Minimax algorithm also relies on making deep copies of the board which slows down the agent. A competitor using a more efficient approach such as memoization of board states can achieve greater depth within the same time frame. Moreover, the agent can also be vulnerable to more sophisticated agents using specialized strategies such as sacrificial play or deep traps. A strong opponent can use a strategy that exploits the agent's prioritization of certain heuristics which reveal its predictable patterns.

## 6. Future Improvements

To further enhance our agent, several improvements can be made by adopting advanced AI techniques, optimizing computational strategies, and refining heuristics. To enable deeper exploration within the time limit, a **transposition table**—a hash table storing previously evaluated game states and their values—could be implemented. This avoids recomputation and narrows Alpha-Beta bounds faster, improving pruning efficiency. While caching results increases memory usage, efficient techniques like **Zobrist hashing** can store states compactly. Additionally, **parallelization** could distribute computations across multiple processors, significantly speeding up the search process.

Beyond algorithmic speed, advanced heuristics like **Parity**, **Stability**, and **Edge Stability** could further enhance decision-making. **Parity** measures the total remaining moves, as an odd or even count can influence outcomes. **Stability** prioritizes discs that cannot be flipped, especially critical in endgame scenarios. **Edge Stability** focuses on securing stable edge positions while preventing the opponent from gaining control, offering strategic advantages.

Advanced AI methods could also be explored. **Reinforcement Learning** (e.g., Q-Learning) allows the agent to improve through trial and error, adapting to diverse strategies over time. **Deep Neural Networks (DNNs)** can replace manual heuristics with a learned evaluation function, trained using supervised learning with labeled game states and outcomes. Alternatively, **Genetic Algorithms** could evolve optimal strategies by mimicking natural selection. The **Negamax algorithm**, a variant of Minimax, leverages the symmetry of Reversi to simplify the search process, reducing computational overhead.

A combination of these techniques would significantly enhance the agent's adaptability, learning capabilities, and search efficiency, paving the way for a stronger, more competitive AI in future iterations.

## References

https://en.wikipedia.org/wiki/Zobrist_hashing

https://samsoft.org.uk/reversi/strategy.htm

https://publications.rwth-aachen.de/record/984514/files/984514.pdf

https://www.researchgate.net/publication/369876515_Application_of_Different_Artificial_Intelligence_Methods_on_Reversi

https://link.springer.com/chapter/10.1007/978-3-540-74930-1_33

https://en.wikipedia.org/wiki/Negamax

https://ceur-ws.org/Vol-1107/paper2.pdf

https://github.com/arminkz/Reversi/tree/master/src

https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/RUSSIA/miniproject1_vaishu_muthu/Paper/Final_Paper.pdf