424 Reversi Agent Report

Elya Renom
McGill ID: 261094604

James Kidd
McGill ID: 260276236

Executive Summary

The strongest algorithm we discovered to play Reversi is a combination of **iterative deepening**, **minimax search**, and **alpha-beta pruning**, all designed to work within a strict **2-second time constraint**. This approach provides a balance of **strategic depth**, **computational efficiency**, and **adaptability to different board sizes**, making it effective in competitive play. Our goal was to create an agent capable of not only making **optimal moves in immediate situations** but also **strategically planning ahead**, while maintaining **adaptability for dynamic board states**.

The crux of the agent's strength is the **iterative deepening framework**. This method progressively increases the search depth, ensuring that even if time constraints limit the exploration of the entire search space, the agent will always return the **best move found so far**. This feature was critical in handling the **strict 2-second per move constraint**, allowing our agent to explore **deeper layers of decision-making** when the computational budget allowed. The **minimax algorithm** forms the decision-making backbone, simulating all possible moves for both the player and the opponent to find the **optimal outcome** under the assumption that the opponent plays perfectly. **Alpha-beta pruning** enhances this process by **eliminating branches of the search tree** that cannot influence the final decision, significantly **improving efficiency**.

Together, these techniques allowed our agent to **calculate optimal strategies** while adhering to computational constraints. A key contributor to the agent's success is the **heuristic evaluation function**, which assesses board states at leaf nodes of the search tree. The evaluation function prioritizes critical factors like **coin parity** (difference in the number of discs controlled by the player and opponent), **mobility** (difference in valid moves available to each player), and **corner control** (possession of stable and strategically advantageous corner positions). This heuristic balances **short-term gains with long-term strategic objectives**, ensuring the agent maintains **dominance on the board** while setting up **favorable endgame conditions**.

Detailed Explanation of Agent Design

Core Algorithms

The agent is built on three foundational techniques:

1. **Iterative Deepening Search (IDS):** This technique dynamically adjusts the search depth, progressively increasing it within a **2-second per-move time limit**. IDS ensures that the agent always returns the best move from the deepest completed depth.

2. **Minimax Algorithm:** Minimax simulates all possible moves for both the agent and opponent, assuming **perfect play**. The agent seeks to maximize its score while minimizing the opponent's gains.

3. **Alpha-Beta Pruning:** This optimization reduces the number of evaluated states by **eliminating branches** that cannot affect the decision, enabling deeper searches within the time limit.

Iterative Deepening Search (IDS)

Iterative Deepening Search (IDS): IDS combines the completeness of depth-first search (DFS) with the optimality of breadth-first search (BFS). We consulted the textbook for pseudocode that influenced our code [1]. The algorithm incrementally increases the search depth d, evaluating all possible moves up to d within the available time budget. Mathematically, the time complexity of IDS is expressed as:

$$T_{\text{IDS}} = \sum_{i=1}^{d} O(b^i) = O(b^d),$$

where b is the branching factor, and d is the maximum depth. This approach ensures that the agent always returns the best move discovered at the deepest fully evaluated level, even if time runs out during deeper iterations. In Reversi, the branching factor b depends on the number of valid moves, which varies dynamically based on the board state.

IDS is the backbone of our agent's decision-making process. It progressively deepens the search tree, evaluating board states at increasing depths until the **2-second per-move time limit** is reached. The primary advantage of IDS is that it ensures the agent always returns the **best move discovered** within the time constraint. If the agent cannot fully explore the search tree due to time limits, it will still provide a move based on the most recent completed depth. The IDS implementation operates as follows:

1. Start with a depth of 1 and incrementally increase the search depth.

2. Evaluate possible moves using **Minimax Search** and **Alpha-Beta Pruning**.

3. If time expires before completing the current depth, the agent selects the **best move identified in previous iterations**.

This method is particularly effective in **time-critical environments** like competitive Reversi matches, ensuring robust decision-making under constraints.

Minimax Algorithm with Alpha-Beta Pruning

Ideally, the agent would employ Minimax to evaluate game states by simulating all possible moves for both players under the assumption of perfect play until the very end of the game:

$$U(s) = \begin{cases} +1 & \text{if the agent wins,} \\ -1 & \text{if the opponent wins,} \\ 0 & \text{if the game is a draw.} \end{cases}$$

However, in practice, minimax cannot compute that deep it would be far too complex, so this binary evaluation is replaced with a heuristic function $H(s)$, which provides a more granular assessment of non-terminal states:

$$V(s) = \begin{cases} \max_{a \in A(s)} V(s') & \text{if the agent is to move,} \\ \min_{a \in A(s)} V(s') & \text{if the opponent is to move,} \end{cases}$$

where $A(s)$ is the set of valid actions from state s, and s' is the resulting state after action a.
Alpha-Beta Pruning optimizes the Minimax search by eliminating branches that do not influence the final decision[2]. Specifically:

- α: Tracks the **best score** achievable by the maximizing player (agent).
- β: Tracks the **best score** achievable by the minimizing player (opponent).

Pruning occurs when $\beta \leq \alpha$, reducing computational overhead and enabling **deeper exploration** within the 2-second limit.

Heuristic Evaluation Function

The heuristic evaluation function estimates the quality of a board state based on three key factors:

1. **Coin Parity:** Measures the difference in discs controlled by the agent and opponent:
$$\text{Coin Parity} = \frac{\text{Agent Discs} - \text{Opponent Discs}}{\text{Total Discs}}.$$
2. **Mobility:** Evaluates the difference in valid moves available to each player:
$$\text{Mobility} = \frac{\text{Agent Moves} - \text{Opponent Moves}}{\text{Total Moves}}.$$
3. **Corner Control:** Rewards ownership of stable corner discs, which cannot be flipped for the remainder of the game.

The final heuristic value is computed as:

$$H(S) = 2.0 \times \text{Coin Parity} + 3.0 \times \text{Mobility} + 5.0 \times \text{Corner Control}.$$

Weights were determined through **grid search** and **random sampling** across various board sizes (6x6 to 12x12), optimizing for win rate, average discs captured, and mobility advantage.

Iterative approach to Development

Initially, our approach to developing a competitive Reversi agent involved dividing the game into three phases—**opening**, **midgame**, and **endgame**—based on the percentage of empty spaces. Each phase employed tailored strategies: the **opening phase** prioritized mobility and corner positioning while avoiding unstable edges; the **midgame** balanced mobility and stability to limit the opponent's options; and the **endgame** focused on maximizing disc captures and securing stable regions.
While this phase-based system provided a structured framework, it introduced significant complexity, including computational overhead and rigid phase boundaries. These limitations often overlooked opportunities unique to specific game states and lacked the flexibility to adapt across varying board sizes and opponent strategies, resulting in inconsistent performance.
Through **iterative design and testing**, we developed a unified strategy using a single evaluation function with **adaptable heuristics** that accounted for phase-specific priorities such as **corner control**, **stability**, and **mobility**. By dynamically weighting heuristics based on board size and game state, the agent could adapt organically, reducing the need for explicit phase transitions while maintaining both simplicity and computational efficiency.

Quantitative Analysis

Depth of Search

The depth of search achieved by the agent varies significantly based on board size and game state. On smaller boards, such as 6x6, the agent can typically reach search depths of about 4-5 within the two-second time limit, as the branching factor—the number of valid moves available—is relatively low. On larger boards like 12x12, the increased branching factor limits the achievable depth to around 3. The branching factor grows exponentially with the number of discs on the board, significantly increasing the computational complexity.
The depth achieved can vary dynamically across different branches due to alpha-beta pruning, as branches with higher heuristic scores are explored more thoroughly, while less promising branches are pruned early.

Breadth of Search

The agent's search breadth, defined as the number of valid moves considered at each level of the game tree, depends on the board size and game state. On smaller boards (6x6), the average breadth ranges from 5 to 10 moves, while larger boards (12x12) typically have 15 to 20 valid moves due to increased mobility.
Alpha-beta pruning significantly reduces the effective breadth by eliminating branches that cannot influence the outcome, focusing computation on high-value moves. Additionally, the agent employs move ordering based on heuristic scores, prioritizing moves that maximize corner control and mobility. This ordering further enhances pruning efficiency, ensuring deeper searches within the fixed time limit.

Impact of Board Size

While the agent's heuristic weights were not explicitly adjusted for varying board sizes, they were fine-tuned to perform well across all tested configurations (6x6 to 12x12). As a result, the agent achieved consistent performance despite the increased complexity of larger boards. Larger boards introduce a higher branching factor, which limits search depth and increases computational demands, particularly in midgame scenarios.
Ideally, board-specific adjustments, such as increasing the weight of corner control on larger boards to reflect its greater strategic importance, would further optimize performance. However, the general robustness of the fine-tuned heuristic ensured that similar results were observed during testing across all board sizes, minimizing the need for such explicit adaptations within the time constraints of development.

Heuristics, Pruning Methods, and Move Ordering Approaches Not Used

During development, we experimented with additional heuristics such as edge control and advanced stability metrics. While edge control initially showed promise in midgame scenarios, it often conflicted with corner control, leading to suboptimal moves in certain situations. Similarly, advanced stability metrics, which classified discs as stable, semi-stable, or unstable, added computational overhead without a significant performance improvement within the time constraints. In move ordering, a static prioritization approach was tested but was less effective than dynamic ordering based on heuristic scores, as it failed to adapt to changing game states.

Predicted Win Rates

To evaluate the quality of our agent's play, we conducted rigorous testing across multiple dimensions. First, we performed extensive head-to-head simulations against a variety of opponent

agents that we constructed ourselves using different approaches (some with the three-phase approach, some with just a heuristic, some with Monte Carlo, some with Minimax but different parameters), as well as the provided agents. These simulations were conducted on even boards of varying sizes, ranging from 6x6 to 12x12, to ensure the agent's adaptability to different game scenarios. We also engaged in iterative design, tweaking weights in the heuristic evaluation function and observing the impact on performance through controlled experiments.
Beyond automated testing, we played directly against the agent as humans to assess its strategic depth and adaptability in real-time scenarios. These are our predictions: The agent is predicted to achieve a win rate of approximately **98%** against the random agent due to its heuristic-guided decision-making and ability to capitalize on random mistakes. Against an average human player, such as "Dave," the win rate is estimated to be **80-90%**, with the agent's superior lookahead and corner control strategies providing a decisive edge. Against classmates' agents, the win rate is expected to range between **60-80%**, depending on the sophistication of competing algorithms, with challenges arising primarily from advanced heuristics that account for board size or Monte Carlo-based strategies.

Advantages

The principal advantage of our agent is the integration of Iterative Deepening Search (IDS) with Minimax and Alpha-Beta Pruning. This hybrid approach allows for efficient exploration of the game tree within the strict computational constraints imposed by the single-threaded Mimi-Server and the tournament's two-second decision time limit. During mid game scenarios, where the branching factor is at its peak, this strategy enables the agent to focus exploration on the most promising branches, effectively reducing the branching factor and maximizing search depth.

IDS ensures the agent always has a feasible move prepared by incrementally deepening the search, even if the search does not reach a terminal state. This dynamic adjustment prevents suboptimal decisions, such as random moves, which could otherwise occur under a static Minimax approach constrained by Alpha-Beta pruning alone.

Weaknesses

Lack of State Memoization: Although Minimax provides significant strategic advantages, the 2-second time limit imposes notable challenges, particularly on larger boards where the search often does not reach terminal states. As board size increases, the branching factor expands exponentially, creating a vast number of potential moves to evaluate. This growth highlights a critical limitation of the agent: its inability to efficiently explore deeper levels of the game tree, a shortcoming that could hinder its competitiveness in a tournament environment.

The absence of a caching mechanism, such as a transposition table, exacerbates this issue. Without it, the agent repeatedly evaluates the same game states, especially in scenarios with recurring patterns due to overlapping moves or symmetrical configurations. This redundancy wastes computational resources, which are especially crucial during the midgame when the branching factor and complexity are at their peak. Additionally, iterative deepening, while useful for incremental depth increases, becomes less consistent without memoization, as the agent may reprocess previously explored states, further straining the limited computation time.
Simplistic Heuristics: While the current heuristic evaluation function covers basic elements like coin parity, mobility, and corner control, it lacks more advanced metrics such as edge evaluation and stability. These metrics could provide critical insights into the long-term strategic implications of moves, especially in the midgame and endgame [3].

Future Improvements

Dynamic Transposition Table with Preconfigured Sizes We began developing a state memoization component for our agent but did not have the resources or time to finalize a working solution. A transposition table is an enhancement that can significantly improve the agent's ability to explore deeper game states within time constraints, particularly against agents using a similar strategy to ours. This optimization works by caching the results of previously evaluated states, allowing the agent to reuse these evaluations instead of recalculating them. This is especially important in Reversi, where many board states recur due to overlapping move sequences. The benefits of integrating a transposition table are twofold:

- **Efficient Exploration:** By storing evaluations, the agent can avoid redundant computations. When a previously encountered board state is revisited, the result can be retrieved directly from the table, freeing up time and resources to explore deeper in the game tree.
- **Deeper Searches:** With the time saved by avoiding redundant evaluations, the agent can allocate its computational budget to exploring new, unexplored game states. This enables better-informed decisions by considering moves further down the game tree, particularly in scenarios requiring deeper strategic analysis.

The design would incorporate four preconfigured transposition tables stored locally for the board sizes 6×6, 8×8, 10×10, and 12×12, collectively not exceeding 500 MB. Using efficient hashing principles, each table will be sized proportionally to minimize state evictions on larger boards while avoiding sparsity on smaller boards, ensuring optimal performance across all dimensions.
Dynamic Heuristic and Stability Metrics The agent's heuristic function currently uses static weights, which do not adapt to the varying importance of game factors across different phases. Incorporating dynamic weighting would enable the agent to adjust priorities throughout the game:

- **Early Game:** Focus on mobility to maximize potential moves.
- **Mid-Game:** Emphasize corner control and edge positioning to stabilize regions and limit the opponent's options.
- **Endgame:** Prioritize coin parity and piece count to secure victory.

Additionally, edge evaluation could be improved by distinguishing between stable edges (secured by controlled corners) and unstable edges (vulnerable to flips). Introducing a gradient-based stability heuristic would classify pieces as stable, semi-stable, or unstable, assigning higher scores to clusters of stable pieces, particularly near corners and edges. These adjustments would reflect long-term strategic value and enhance the agent's decision-making across all phases.

Use of Input Sources

In the development of our Reversi agent, we relied primarily on foundational concepts provided in the class slides, the textbook, and our own understanding of algorithms such as **Alpha-Beta Pruning** and **Iterative Deepening Search (IDS)**. These slides served as a conceptual guide for implementing these algorithms; however, no code or direct excerpts were copied. To refine and optimize certain aspects of our code, we utilized ChatGPT selectively mostly for debugging. For example, debugging the integration of IDS with alpha-beta pruning. We asked:

> "How can I handle time constraints effectively when combining Iterative Deepening Search (IDS) with alpha-beta pruning? Suggest strategies for managing incomplete search depths."

The response highlighted the importance of returning the best move from the deepest fully explored level if the time expired during the current depth iteration. We incorporated this insight to improve the robustness of our IDS implementation under the strict 2-second decision limit. In summary, while we consulted ChatGPT for clarification and refinement, all implementations were written independently.

References

[1] Russell, Stuart J., and Peter Norvig. *Artificial Intelligence: A Modern Approach.* 4th ed., Pearson, 2020, pp. 80-84.

[2] Knuth, Donald E., and Ronald W. Moore. *An Analysis of Alpha-Beta Pruning.* Artificial Intelligence, vol. 6, no. 4, 1975, pp. 293–326.

[3] Vaishnavi Sannidhanam and Muthukaruppan Annamalai, *An Analysis of Heuristics in Othello.* Department of Computer Science and Engineering, Paul G. Allen Center, University of Washington, Seattle, WA-98195. `mailto:vaishu@cs.washington.edu,muthu@cs.washington.edu`.