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). 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

The agent employs Minimax to evaluate game states by simulating all possible moves for both players under the assumption of perfect play. The utility function $U(s)$ for a board state s is defined as:

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

In practice, this binary evaluation is replaced with a heuristic function $H(s)$, which provides a more granular assessment of non-terminal states. The Minimax algorithm recursively computes the optimal value for each state:

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

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. 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 paired with the tournament's two-second decision time limit. During midgame scenarios, when the branching factor is at its peak, this strategy allows the agent to focus computational resources on promising branches, effectively reducing the branching factor and maximizing search depth.

IDS ensures that 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 would inevitably occur under a static Minimax approach constrained by Alpha-Beta pruning alone.

To further optimize the search, we incorporated a transposition table to store previously evaluated game states. This mechanism minimizes redundant computations by leveraging memoization principles. Caching is particularly effective in a game like Reversi, where many states recur due to overlapping gameplay scenarios. In the context of our Iterative Deepening Search (IDS) strategy, the transposition table is particularly valuable as it avoids recomputing state evaluations across successive iterations of increasing depth.

Disadvantages

Our agent's heuristic design is limited by the absence of advanced evaluation metrics, such as coin parity and stability heuristics. These metrics provide critical insights into midgame and endgame dynamics, potentially enhancing the agent's ability to make nuanced strategic decisions. [**SOURCE**].

The fixed size and hashing approach of the transposition table introduce significant inefficiencies. On smaller boards, the sparsity of the table results in underutilization. Conversely, on larger boards, hash collisions and limited capacity lead to frequent evictions of valuable state information, compromising the caching mechanism's effectiveness. These issues reduce the consistency of the agent's performance across varying board sizes, as the transposition table struggles to adapt to the dynamic demands of different game configurations.

Finally, the agent's performance is notably influenced by board dimensions. On larger boards, where complex evaluations and deeper searches are required, the agent performs well. However, on smaller boards with simpler states, the computational resources are not fully

utilized, leading to diminished efficiency. This disparity underscores the need for a more adaptive approach to handling diverse board sizes effectively.

Future Improvements

While our current agent demonstrates strong performance, several areas for further enhancement could significantly improve its strategic depth, adaptability, and computational efficiency. Below are detailed proposals for future improvements:

1. **Dynamic Heuristic Weights:** The current heuristic function relies on fixed weights for coin parity, mobility, and corner control, which remain constant throughout the game. However, the relative importance of these factors changes across the opening, midgame, and endgame. Future iterations could implement dynamic weights that adapt based on the current game phase. For instance:

   - **Opening Phase:** Prioritize *mobility* to maximize potential moves and control of the board.
   - **Midgame:** Focus on *corner control* and edge stability to secure strategic positions.
   - **Endgame:** Emphasize *coin parity* to ensure dominance in disc count.

   Dynamic weighting can be implemented through a linear or non-linear function, such as:

   $$w_i(t) = w_i^{\text{base}} + f(t) \cdot \Delta w_i,$$

   where w_i^{base} is the base weight, $f(t)$ is a phase-specific adjustment factor, and Δw_i is the change in weight across game phases. The function $f(t)$ could depend on metrics such as the percentage of empty spaces or the mobility difference between players. Techniques like reinforcement learning could also optimize these weights through self-play, allowing the agent to learn phase-specific strategies.

2. **Advanced Stability Metrics:** While the current heuristic indirectly accounts for stability through corner control, explicitly incorporating stability metrics could further refine decision-making. Stability measures could classify discs as:

   - **Stable:** Discs that cannot be flipped for the remainder of the game (e.g., corners and discs surrounded by stable pieces).
   - **Semi-Stable:** Discs that are stable under certain conditions, such as specific opponent moves.
   - **Unstable:** Discs that are highly vulnerable to flipping.

   A stability metric S could be integrated into the heuristic function:

   $$S = \sum_i s_i \cdot v_i,$$

   where s_i represents the stability classification score (e.g., 1 for stable, 0.5 for semi-stable, and 0 for unstable), and v_i is the positional value of the disc. Incorporating this metric would enable the agent to make more nuanced decisions, such as avoiding risky placements near unstable edges or prioritizing moves that cluster stable regions.

3. **Dynamic Transposition Tables:** The current transposition table design uses a fixed size and hashing approach, which introduces inefficiencies across varying board dimensions. A dynamic transposition table system could address these limitations by preconfiguring table sizes tailored to specific board dimensions:

- **Smaller Boards (e.g., 6x6):** Use smaller table sizes to minimize overhead, as the game states are simpler and searches terminate faster.
- **Larger Boards (e.g., 12x12):** Allocate larger tables to accommodate the increased number of states and deeper searches.

The capacity C of the transposition table could be dynamically allocated as:

$$C = k \cdot b^d,$$

where b is the average branching factor, d is the expected depth, and k is a scaling factor based on board size. Additionally, techniques such as Least Recently Used (LRU) eviction policies could prevent valuable state information from being overwritten during gameplay.

4. **Move Ordering with Machine Learning (if we were allowed):** Enhancing move ordering could significantly improve the efficiency of alpha-beta pruning. Instead of relying on heuristic scores alone, a supervised machine learning model could predict the likelihood of a move being optimal based on features such as:

   - Board configuration.
   - Opponent's recent moves.
   - Stability and mobility metrics.

   These predictions could prioritize the most promising moves, maximizing pruning opportunities and allowing deeper searches within the time constraint. A gradient boosting classifier or neural network trained on self-play data could serve this purpose.

5. **Integration of Monte Carlo Tree Search (MCTS):** While Minimax with alpha-beta pruning is effective, Monte Carlo Tree Search (MCTS) could be integrated as a complementary or alternative approach. MCTS is particularly effective in midgame scenarios with high branching factors, as it uses random simulations to evaluate moves without requiring a complete search. An adaptive hybrid model could use:

   - Minimax for shallow depths or when reliable heuristics are available.
   - MCTS for deeper or highly complex states where heuristic evaluation is less effective.

6. **Parallelization and Hardware Optimization:** To maximize computational resources, future iterations could implement parallel search techniques:

   - **Parallel Alpha-Beta Pruning:** Distribute branches of the search tree across multiple threads or processors.
   - **GPU Acceleration:** Use GPUs for large-scale evaluations, particularly for heuristic calculations and simulations.

By implementing these improvements, the agent would achieve greater adaptability, efficiency, and competitiveness, particularly against advanced human players and sophisticated AI opponents.

Acknowledgements

This project was class slides, and testing against various opponents. We consulted example code for alpha-beta pruning and utilized grid search for heuristic tuning.