



# Final Project Proposal

Team: Andy Dao, Sierra Andrews  
11/10/2025

Professor Adela Velez  
Artificial Intelligence  
CSC 4631 121

Github Repository: <https://github.com/ViviVoid/othello-ai>

---

## 1 Problem Statement

Othello is a board game that has an enormous state space and horizon. Many applications in the real world also share this similar conundrum. This means that the way different algorithms approach the issue of exploring these large horizons can impart different ideas of what they could be capable of and their efficacy. As such, Othello serves as a strong jumping point to cross-analyze algorithm performance for large state spaces/horizons in an adversarial space.

Standard human board analysis in Othello relies on noting positional fundamentals that contribute to strong board states. This includes principles that follow along corner control, edge stability, and mobility (ability to maximize one's potential moves while minimizing the opponent's potential moves). This can layer itself into strategic concepts such as parity, frontier management, and disc stability to develop multi-layered strategies that ultimately lead to a favorable board configuration ultimately leading to victory. As these traits aren't always maximized, as can be demonstrated in gambits that sacrifice mobility or cause temporary disadvantages to a player playing that action in Othello, maximizing these facets or strategic concepts is not a straightforward approach in terms of maximizing stable discs on an Othello board.

As such, traditional analysis of Othello states have been explored at length but due to the extreme horizon of Othello, many states in many phases of the game are less explored leaving much room for other algorithms.

Current algorithms designed for playing Othello employ computationally evaluating board states and exploring these decision pathways in depth to produce AI agents to play the game. This domain still remains valuable for comparing the effectiveness of different search and evaluation models. One particular meaningful comparison we can identify is between Monte Carlo Tree Search (MCTS) and Minimax, examining how each algorithm performs in isolation to determine the extent to which their respective exploration and evaluation strategies contribute to effective Othello play and as such, their potential performance in other problems with an expansive state space/horizon.

## 2 AI Algorithm(s) / Approach(es)

Monte Carlo Tree Search (MCTS) is an algorithm famous for its use in multi-agent strategy games like Go and Chess. It works by selecting a random path, expanding the path it's on by simulating new potential states, and then backpropagating to return the next best move to complete. When a search is started at the root node, the algorithm uses a selection policy that attempts to find a balance between average path reward and path uncertainty. When a non-terminal leaf node is reached the algorithm adds at least one child node to explore or simulate a new path with new possible outcomes or actions. After a new node is added, a simulation is completed with a random agent to limit computation. The resulting simulation predication is then backtracked to the root node, the statistics are updated, and the process is repeated until the game is completed.

---

We will be applying MCTS to the game Othello against other agent types to evaluate its performances against other algorithms that have been used to solve this problem type in the past. This is ideal as we can use this to analyze feature importance of factors such as going first (playing white) in Othello and how that impacts the horizon as well as space and time complexity for a problem that usually doesn't apply MCTS.

### 3 Project Details and Analysis

We are applying MCTS to Othello by creating an agent that will use MCTS to decide its next best move. The input will be an Othello board state and the output will be the location to place the agent's next piece.

An Othello board state will consist of the individual game pieces allocated on the 8x8 board in tandem with their color association: white or black. This board state will also store information about the material sums of both players according to the pieces they have left. For example, player one may have 35 of their specific color on the board while player two will have 26. This will be part of the input given to a minimax/MCTS agent.

For accuracy of our agent, we will be performing our MCTS simulation against a minimax agent and a random-moves agent to analyze how often it wins, ties, or loses. We will plot the performance against each agent and the difference for when it moves first or not to compare how much of an effect first player advantage has for every agent. We will evaluate and obtain information about the MCTS agent, the minimax agent, and random-moves agent, in performance by having them play 10000 games against each other and themselves (40000 games total).

Each game will record the total points left for each agent at the end of the game, number of moves to complete the game and which agent won the game which will provide a visualization of the AI's performance against the other agents. Additionally, the average time for each agent to complete a move will also be recorded and compared in a plot. Space complexity would also be analyzed as MCTS is historically very space intensive.

### 4 Project Timeline and Milestones

#### 4.1 Milestone 1: Pygame Environment

- **Game Environment: Standard 8x8 Othello Board**
  - Complete implementation of Othello game logic
  - Handles full rule set including valid move detection, disc flipping, and game termination
- **Core Functionalities**
  - Legal Moves Display
  - Board Value Heuristic
    - \* Stable Disc Calculation
    - \* Disc Count Tracking (per color)

- 
- \* Available Move Count per Player
  - Disc Flipping Rules (directional search and reversal)
  - Turn Alternation Mechanism
    - \* Skip Turn Logic (for cases when opponent has no legal moves)
    - \* Alternating Turns under Normal Conditions
  - Terminal Node Detection (no further legal moves for either player)
  - Win Detection
  - Endgame Evaluation based on final disc counts
  - **Agent Support**
    - Two Player Configuration
    - Agent Placeholder Scaffolding
      - \* Human Player Input (Fully Implemented)
      - \* Minimax Algorithm
      - \* Monte Carlo Tree Search (MCTS)
  - **Visualization and Interface**
    - Pygame-based Graphical User Interface
    - Turn-to-turn display of valid moves, current turn, and disc counts
  - **Automation and Analysis Tools**
    - Batch Job Scripts for automated matches and performance benchmarking
    - Headless Mode (output without display) for simulation and statistical evaluation

**Deliverable:** Rule-compliant and interactive Othello game. Visual representation using Pygame as an artifact. Can switch player modes (human, human) or (minimax,human) or (MCTS, minimax)

## 4.2 Milestone 2: Algorithm Implementations

- **Monte Carlo Tree Search (MCTS) Agent**
  - Develop a node-based search structure for Othello board states
  - Implement core MCTS components:
    - \* **Selection**, traverse the tree using a selection policy (e.g., UCT)
    - \* **Expansion**, add one or more child nodes for unexplored moves
    - \* **Simulation** (rollout), perform randomized or policy-guided playouts from the expanded node to a terminal or cutoff state
    - \* **Backpropagation**, propagate rollout results up the tree to update node statistics (visit counts, value estimates)
  - Use randomized rollouts to estimate move strengths and guide exploration
  - Provide tunable parameters and controls:
    - \* Exploration constant (e.g.,  $c$  in UCT)
    - \* Rollout depth or cutoff criteria
    - \* Number of iterations / simulations per move
  - Performance goal: balance accuracy and efficiency (trade-offs between time spent per move and space complexity)
- **Minimax Agent**

- 
- Implement Minimax search for deterministic evaluation
  - Develop a heuristic evaluation function that emphasizes Othello-relevant features:
    - \* **Corner control** – high weight for owning corner squares
    - \* **Edge control** – positive weighting for stable edge positions
    - \* **Mobility** – number of legal moves available to the player (and opponent)
    - \* **Disc stability** – count/weight of discs that cannot be flipped in future play
    - \* **Parity** – evaluation or bias for parity-related endgame considerations (e.g., even/odd empty-square regions)

**Deliverable:** Functional implementation of both agents for the othello game in milestone 1's artifact.

### 4.3 Milestone 3: Algorithm Analysis

- Conduct extensive AI vs. AI testing between MCTS and Minimax across multiple openings.
  - Win rate, average move time, and computational resource usage.
  - Node exploration count and heuristic score progression.
- Collect quantitative data
  - Win percentages
  - Disc difference
  - Search depths
- Visualize performance trends using plots and statistical summaries
- Perform deeper analysis of algorithmic tendencies:
  - Early vs. late-game decision quality.
  - Sensitivity to position type and branching factors.
- Summarize findings to highlight trade-offs between probabilistic (MCTS) and deterministic (Minimax) decision frameworks.

**Deliverable:** Performance Comparison Visuals:

- Statistical Results (tables/graphs)
- Analytical discussion of results and implications

### 4.4 Division of Work

- Sierra Andrews
  - Milestone 1
    - \* Scaffolding Agents
  - Milestone 2
    - \* MCTS Core Components
      - Selection
      - Expansion
      - Simulation

- 
- Backpropagation
  - Milestone 3
    - \* Preface
    - \* Conclusion
    - \* Visuals
    - \* Discussion/Takeaways
      - . Visuals in Context
      - . Significance
  - Andy Dao
    - Milestone 1
      - \* Board State Heuristics
      - \* Othello game logics
      - \* Pygame Visuals
      - \* Headless Scripts / Batch Job Scripts
    - Milestone 2
      - \* MCTS Node
      - \* Minimax
    - Milestone 3
      - \* Discussion/Takeaways
        - . Visuals in Context
        - . Significance