**Connect 4 AI Evaluation: Minimax vs Monte Carlo Tree Search (MCTS)**

**Team Matrix**  
Members: Greeshma Chanduri, Pravalika Kamineni, Kiranmayi Modugu  
GitHub Repository: <https://github.com/Greeshma-DS/Team-Matrix.git>  
Date: April 29, 2025

**Abstract**  
This project implements and compares two artificial intelligence (AI) strategies for the game Connect 4: Minimax with Alpha-Beta Pruning and Monte Carlo Tree Search (MCTS). The project uses a Streamlit-based web interface to simulate multiple games between the two algorithms, allowing users to adjust parameters such as the depth of the Minimax algorithm and the number of simulations for MCTS. The performance of the algorithms is analyzed using metrics such as win rate, decision time, and strategic decision-making. Preliminary results indicate that Minimax outperforms MCTS in both win rate and decision time, although MCTS shows potential for improvement with further optimization. This comparison provides valuable insights into deterministic versus probabilistic AI approaches in zero-sum games and offers practical applications in the fields of game theory and AI education.

**1. Introduction**

**1.1 Research Question**  
The central question of this project is: Which AI algorithm—Minimax with Alpha-Beta Pruning or Monte Carlo Tree Search (MCTS)—performs better in the game Connect 4 in terms of win rate, decision time, and strategic decision-making?

**1.2 Background**  
Connect 4 is a classic two-player, zero-sum strategy game where players take turns to drop colored discs into a 6x7 grid, aiming to connect four discs in a row either horizontally, vertically, or diagonally. The game ends when one player achieves this goal or when the board is filled without any player winning (a draw). This project develops two AI agents:

* **Minimax with Alpha-Beta Pruning**: A deterministic tree search algorithm that explores all possible moves up to a specified depth and uses pruning to eliminate branches of the game tree that do not lead to optimal solutions.
* **Monte Carlo Tree Search (MCTS)**: A probabilistic AI algorithm that uses random simulations to estimate the quality of each possible move and select the best move based on these estimates.

**1.3 Significance**  
This project contributes to the understanding of AI performance in turn-based strategy games by offering a practical framework for comparing algorithms. The Streamlit interface developed in this project enables real-time analysis of AI performance, making it a useful tool for exploring different strategies and tuning parameters. The results and insights gained from this project can be applied to other AI systems in game theory and educational contexts.

**2. Related Literature**

* **Allis, L. V. (1988)**: In his seminal work, Allis demonstrated the power of Minimax with heuristics for solving Connect 4, showing its effectiveness in finding optimal moves. Our project extends this work by comparing Minimax against MCTS, using a modern heuristic.
* **Browne, C., et al. (2012)**: This survey paper reviews the applications of MCTS in complex games like Go and Chess. Our project adapts MCTS to Connect 4, implementing a custom Upper Confidence Bound for Trees (UCT) formula for better move selection.
* **Silver, D., et al. (2016)**: The paper introduces the use of MCTS combined with neural networks to power AlphaGo, a game-playing AI that achieved superhuman performance in the game of Go. This work forms the foundation for understanding MCTS, and our project explores its application in a simpler context (Connect 4) without neural networks.

The main contribution of our project is a direct comparison between Minimax and MCTS applied to Connect 4, using a real-time Streamlit interface and a Pygame GUI for visualizing gameplay.

**3. Methodology**

**3.1 Game Implementation**

* **Board Setup**: The Connect 4 board is a 6x7 grid, which is represented as a NumPy array for efficient processing.
* **Game Mechanics**: Players take turns dropping discs into the columns of the grid, with the goal of connecting four discs in a row either horizontally, vertically, or diagonally. The game automatically checks for wins or draws after each move.
* **Utility Functions**: A set of utility functions is implemented to handle tasks such as creating the board, validating moves, placing pieces, and detecting wins or draws.

**3.2 AI Algorithms**

*Minimax with Alpha-Beta Pruning*

* **Approach**: Minimax explores all possible moves up to a fixed depth (default: depth 3, adjustable) and selects the move with the highest score based on a heuristic. The heuristic prioritizes strategies such as center control, blocking threats, and creating three-in-a-row patterns.
* **Optimization**: Alpha-Beta Pruning is used to improve efficiency by eliminating unpromising branches of the game tree, achieving approximately 60% branch pruning.
* **Performance**: The average time for the Minimax algorithm to make a move is 0.27 seconds.

*Monte Carlo Tree Search (MCTS)*

* **Approach**: MCTS runs simulated random games (default: 50 simulations per move, adjustable) to estimate the quality of each move. The UCT formula is used to balance exploration and exploitation of the game tree.
* **Optimization**: To speed up performance, the number of simulations was reduced to 50, resulting in an average move time of 3.34 seconds.
* **Challenges**: Initially, MCTS had a low win rate of only 1 out of 10 games, but this was addressed by tuning the UCT constant and adjusting the simulation logic.

**3.3 Streamlit Interface**  
The Streamlit interface provides an interactive platform for simulating AI vs AI games and visualizing the results in real-time. Key features of the interface include:

* **Sidebar Controls**: Users can adjust parameters such as the number of games (1 to 50), the depth of Minimax (2 to 6), and the number of MCTS simulations (50 to 500).
* **Simulation**: Users can run the game with real-time updates to the board and see the results of each move.
* **Results**: After each simulation, the interface displays win rates, average move times, and a bar chart comparing the performance of Minimax and MCTS.
* **Technology Stack**: Python 3.8+, Streamlit, NumPy, and Matplotlib.

**3.4 Pygame GUI**  
In addition to the Streamlit interface, a Pygame GUI is used to provide a visual representation of the game. This GUI shows the animated movement of red discs (for Minimax) and yellow discs (for MCTS) as they are placed on the board. Key features include:

* **Move Time Display**: Each move is timed, and the time taken to make the move is displayed.
* **Winner Announcement**: The GUI announces the winner of the game once a player connects four discs.
* **Future Enhancements**: Planned improvements to the GUI include adding difficulty sliders and options for different board sizes (e.g., 5x6 or 7x8 grids).

**3.5 Evaluation Metrics**  
The performance of both algorithms is evaluated using the following metrics:

* **Win Rate**: The percentage of games won by each AI algorithm.
* **Average Decision Time**: The average time it takes for the AI to make a move.
* **Strategic Decision-Making**: This is assessed indirectly through the win rates and the effectiveness of the heuristics used by Minimax and the simulation quality in MCTS.

**4. Achievements**

* Developed functional implementations of both the Minimax and MCTS algorithms for Connect 4.
* Built a Streamlit-based web interface for real-time simulation and performance analysis.
* Created a Pygame GUI for visualizing gameplay with animated discs.
* Optimized Minimax with Alpha-Beta Pruning, achieving an average move time of 0.27 seconds.
* Improved MCTS performance by reducing simulation time to an average of 3.34 seconds per move.
* Conducted preliminary simulations showing Minimax's dominance in both win rate and speed.

**5. Challenges and Solutions**

1. **MCTS Slow Performance**
   * **Challenge**: Initially, the MCTS algorithm took over 6 seconds per move.
   * **Solution**: Reduced the number of simulations to 50 per move, achieving an average move time of 3.34 seconds. Further optimizations, such as prioritizing high-value moves, are planned to improve this.
2. **Minimax Computational Cost**
   * **Challenge**: The early versions of Minimax were slow without Alpha-Beta Pruning.
   * **Solution**: Implemented Alpha-Beta Pruning, reducing move time to 0.27 seconds, and optimized the heuristic for better strategic performance.
3. **Pygame GUI Development**
   * **Challenge**: The team was initially unfamiliar with Pygame, causing delays in development.
   * **Solution**: Researched Pygame documentation and successfully developed a functional GUI. Future improvements will focus on enhanced visual effects and user interaction.
4. **Low MCTS Win Rate**
   * **Challenge**: MCTS won only 1 out of 10 games.
   * **Solution**: Adjusted the UCT constant and tuned the simulation parameters. Additional simulations are planned to further refine the algorithm.

**6. Preliminary Results**

* **Win Rate**
  + Minimax: 0 / 5 games
  + MCTS: 5 / 5 games
  + Draws: 0 / 5 games
* **Average Decision Time**
  + Minimax: 0.04 seconds per move
  + MCTS: 1.30 seconds per move
* **Efficiency**
  + Minimax: ~60% branch pruning
  + MCTS: Performance scales with simulations, but time increases
* **Insights**
  + MCTS now wins more games due to better long-term strategy.
  + Minimax remains faster per move but is less effective in longer games.

**7. Discussion**

**Strengths**

* **Minimax**: Fast, deterministic, and effective with a well-tuned heuristic.
* **MCTS**: Adaptable and potentially stronger with increased simulations.
* **Streamlit Interface**: User-friendly for real-time experimentation and analysis.
* **Pygame GUI**: Provides an engaging visual experience.

**8. Next Steps**

* Add difficulty sliders and board size options to the GUI.
* Further optimize MCTS (UCT tuning, simulation depth).
* Conduct large-scale simulations (100+ games) for a more robust statistical analysis.
* Enable human vs AI mode in Streamlit.
* Complete final report and record a demo video.

**9. Conclusion**  
This project has successfully compared Minimax with Alpha-Beta Pruning and Monte Carlo Tree Search (MCTS) in the game of Connect 4, utilizing both a Streamlit web interface and a Pygame GUI. Preliminary results suggest that Minimax outperforms MCTS in both win rate and decision time, but MCTS shows promise in long-term strategy. The framework developed in this project provides useful