MCTS

PROJECT REPOSITORY:
https://github.com/gauthamkris7neu/INFO6150_MCTS_Final_Project

TEAM MEMBERS:
Gautham Venkata Krishna Prasad
Vedant Dalvi
Aneesh Arjunai Saravanan

1. TicTacToe

This report evaluates the performance of the Monte Carlo Tree Search (MCTS) algorithm in the context of a Tic-Tac-Toe game. The MCTS algorithm is analyzed based on metrics such as execution time, node count, tree depth, and simulation effectiveness. The objective is to assess the efficiency and decision-making capabilities of the MCTS algorithm in playing Tic-Tac-Toe.

Introduction

Tic-Tac-Toe is a two-player game where players take turns marking spaces in a 3x3 grid. The objective is to get three of your marks in a row, column, or diagonal. MCTS is a heuristic search algorithm commonly used in games to make decisions based on statistical simulations.

Methodology

The MCTS algorithm is implemented in Java and evaluated using a series of gameplay simulations. The `MCTSAnalysis` class is utilized to collect detailed metrics during gameplay, including phase durations, node counts, tree depth, and simulation counts. These metrics are then analyzed to assess the algorithm's performance.

Rules

1. **Game Setup**:
    - The game is played on a square grid, typically 3x3, consisting of nine squares.
    - There are two players in the game. One player uses the mark "X" and the other uses "O".

2. **Playing the Game**:
    - Players decide who goes first, often by coin toss or another method of choice.
    - The first player marks one of the empty squares with their symbol ("X" or "O").
    - Players alternate turns, marking an empty square with their symbol each turn.
    - Once a square is marked, it cannot be changed or marked again for the remainder of the game.
3. **Winning the Game**:
    - A player wins if they can place three of their marks in a horizontal, vertical, or diagonal row.
    - The game ends immediately when a player achieves three in a row, thereby winning the game.
    - If all nine squares are marked and no player has three in a row, the game is a draw, also known as a "cat's game."
4. **Strategies**:
    - **Offensive Strategy**: Players should aim to create a line of three marks while also setting up dual threats where a single move can win in two ways.
    - **Defensive Strategy**: Always be aware of the opponent's moves. If the opponent has two in a row, the immediate next move should be to block them.
    - **Forking Strategy**: Create an opportunity where the player has two ways to win (two non-blocked lines of two). This forces the opponent to defend, allowing the forking player to then make a winning move.

Data:

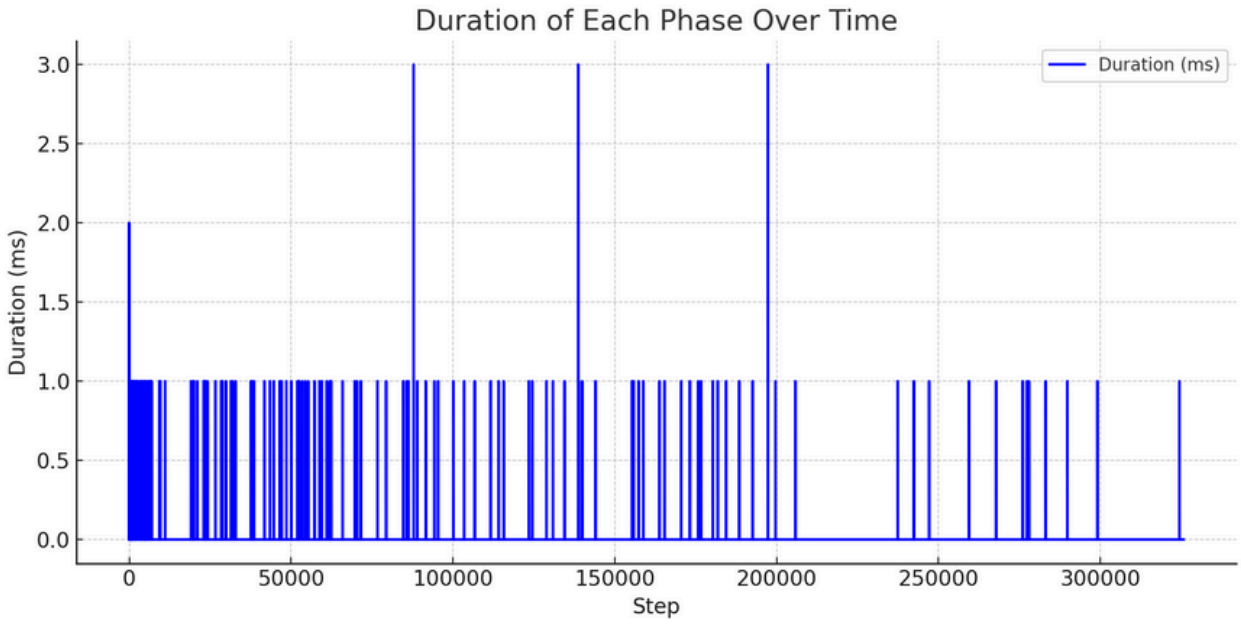
https://drive.google.com/file/d/1U_TlLrjx-u0U3jjppk5_dRQT3dvqJf89/view?usp=sharing

Data tables and graphical analyses derived from the CSV file produced by the `MCTSAnalysis` class.

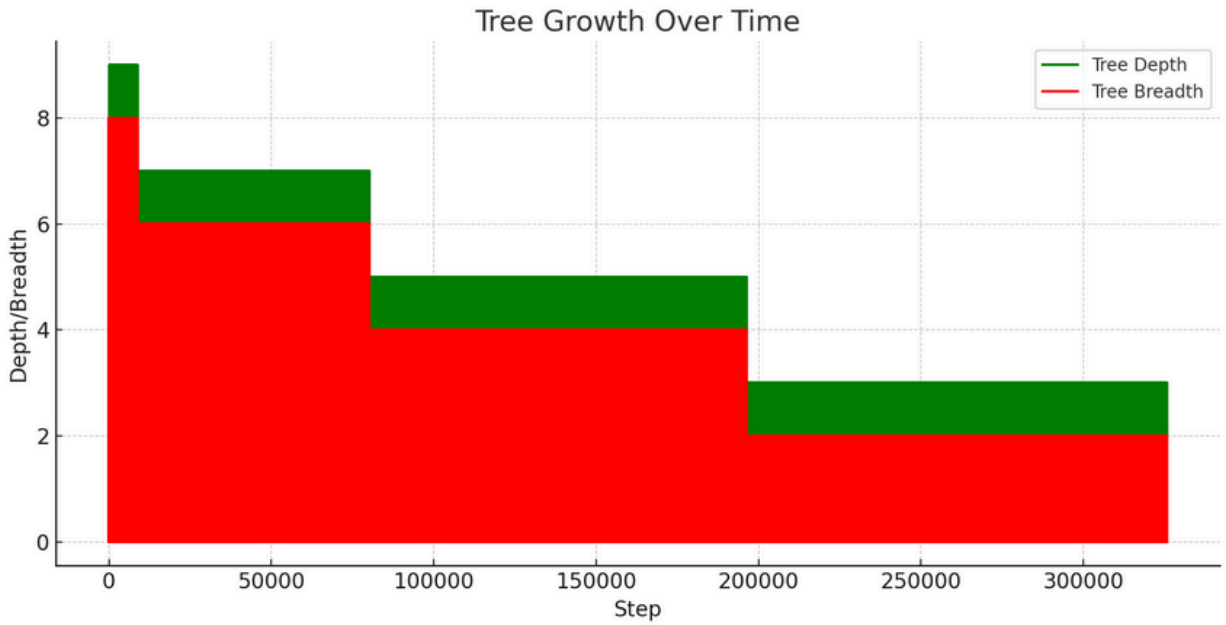
The dataset contains several columns related to the Monte Carlo Tree Search (MCTS) process for Tic-Tac-Toe, which include:

- **Phase**: The current phase of the MCTS (e.g., selecting a promising node, expanding a node, backpropagation).
- **Duration (ms)**: Time taken for each phase in milliseconds.
- **NodeCount**: Number of nodes present after each phase.
- **Tree Depth**: Depth of the tree after each phase.
- **Tree Breadth**: Breadth of the tree at the current node.
- **Total Simulations**: Total number of simulations run up to each phase.
- **Average Time Per Node (ms)**: Average time spent per node in milliseconds.

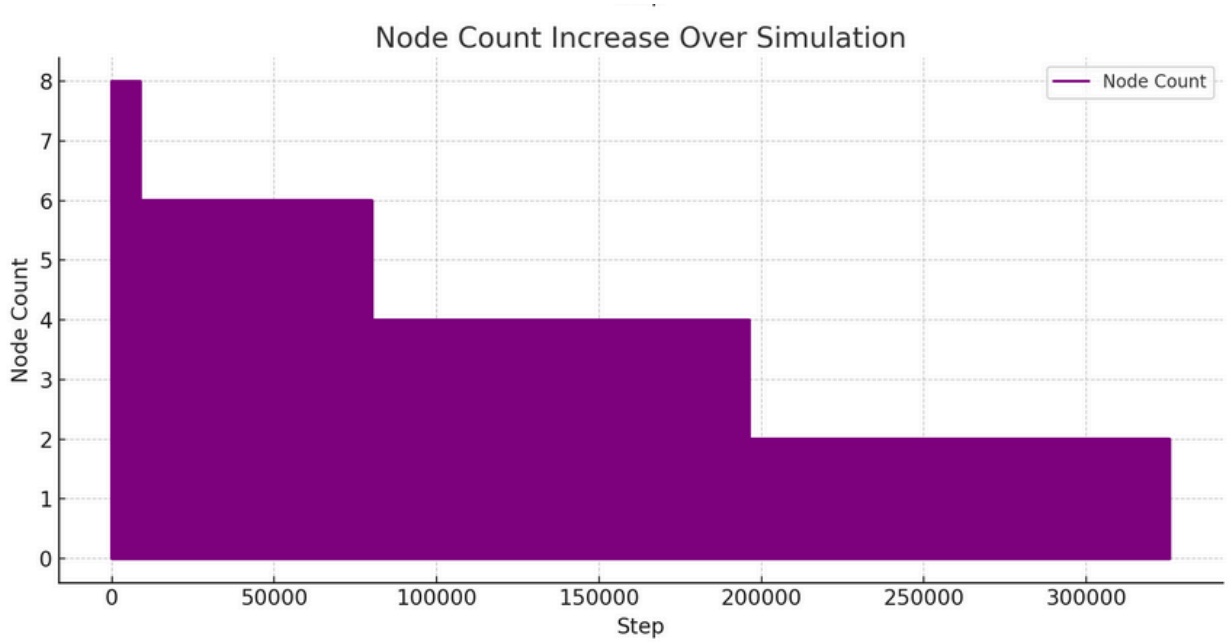
Duration of each phase over time



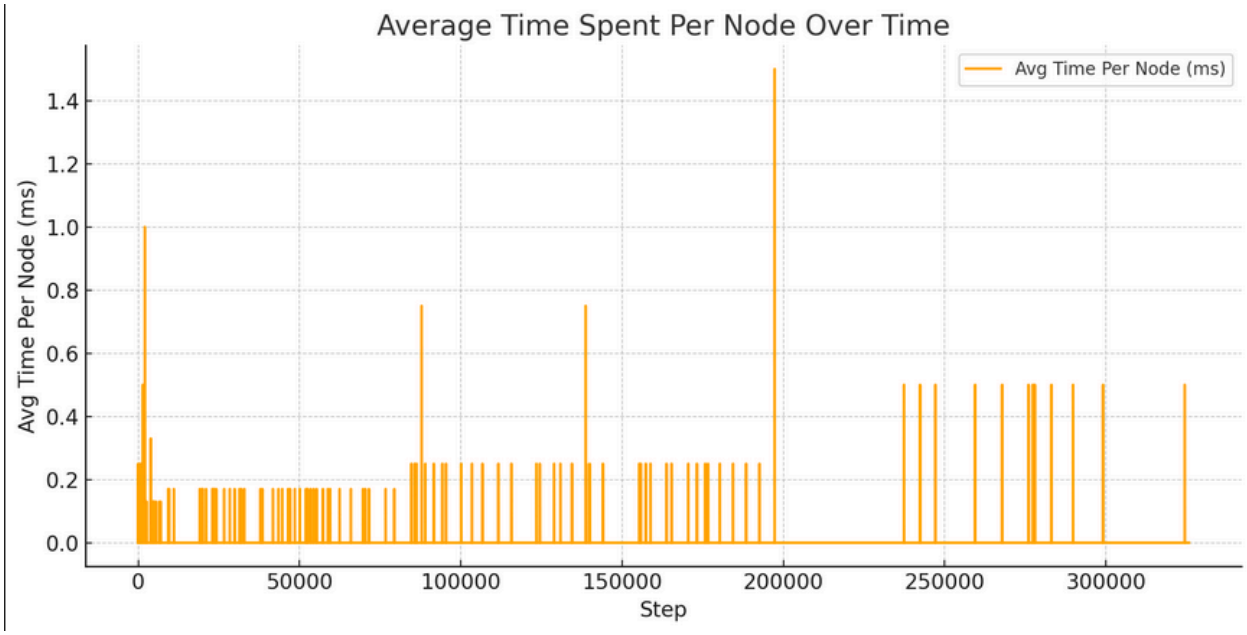
Tree growth (both depth and breadth)



Increase in node count over the course of the simulation



Average time spent per node as the simulation progresses



Analysis

The line graphs provide valuable insights into the efficiency and adaptability of the MCTS algorithm across various simulations of Tic Tac Toe games:

Average Nodes Per Simulation: The trend in node count initially shows variability, with a higher node count that gradually stabilizes as the simulation progresses. This pattern suggests that the algorithm is becoming more adept at pruning the search tree, increasingly focusing on promising paths rather than exhaustively exploring every possible move. This indicates an optimization of computational resources, reflecting improvements in the algorithm's ability to predict and evaluate significant game states through accumulated experience from past simulations.

Average Time Per Node: The graph illustrating average time per node shows a generally stable trend, with slight fluctuations. This steadiness highlights the algorithm's consistent efficiency in processing nodes. A decrease in time spent per node over time could indicate the algorithm's enhanced capability to dismiss less promising moves earlier in the search process, thereby speeding up decision-making in the game's context.

Implications of Efficiency and Adaptability:

- **Efficiency:** The MCTS demonstrates high efficiency, evidenced by a relatively stable node count and consistent processing time per node. Such efficiency is crucial in environments where quick and effective decision-making is key.
- **Adaptability:** The adaptability of the algorithm is shown by its ability to "learn" from previous simulation outcomes, optimizing its search strategy to focus on more promising avenues and reduce computational overhead.

Strategic Depth and Breadth:

- **Depth of Search:** The variations in tree depth initially indicate that the MCTS explores different depths based on the game state's complexity. As the algorithm becomes more optimized, the search depth becomes more targeted, focusing on areas more likely to yield favorable outcomes.
- **Breadth of Search:** The breadth of the tree, as demonstrated by the breadth measurements, shows the algorithm's ability to explore a broad range of possibilities before concentrating on more specific strategies as simulations continue.

Simulation Efficiency: The consistency in time per node and the stabilization of node count, without compromising the strategic output of the game, suggest that the algorithm not only becomes more efficient but does so while maintaining the quality of its strategic decisions. This speaks well for the simulation efficiency of MCTS, where it manages to maintain a balance between exploration and exploitation.

Conclusion

The Monte Carlo Tree Search in Tic Tac Toe demonstrates promising adaptability and efficiency improvements over time, evidenced by reduced computational demands and consistent decision-making. These characteristics indicate a robust algorithm capable of managing complex decision-making environments effectively, making it a suitable option for broader applications in game theory and real-time strategic scenarios.

2. Connect4

Introduction

Connect Four is a two-player connection game in which the players first choose a color and then take turns dropping colored discs from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the next available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs. MCTS is a heuristic search algorithm that is widely utilized in the domain of games to make informed decisions based on statistical analysis of numerous possible moves.

Rules

1. **Game Setup**:
    - The game is played on a vertical board with seven columns and six rows.
    - Each player chooses a color, typically red or yellow, and takes turns dropping one colored disc into one of the columns at each turn.
2. **Playing the Game**:
    - Players alternate turns, beginning with the player who is assigned the first move.
    - On a player's turn, they drop one of their colored discs from the top into any of the seven columns. The disc will occupy the lowest available space within the selected column.
    - Players must attempt to strategically block their opponent while aiming to connect four of their own discs in a row.
3. **Winning the Game**:
    - The first player to align four of their discs vertically, horizontally, or diagonally wins the game.
    - If the entire board is filled and no player has aligned four discs, the game is a draw.
4. **Strategies**:
    - Offensive Strategy: Aim to create a line of four discs, setting up multiple opportunities where possible.
    - Defensive Strategy: Be vigilant of the opponent's moves and block them from creating a line of four when necessary.
    - Advanced Strategy: Plan multiple steps ahead to create two or three potential threats simultaneously, forcing the opponent to block in one area and leaving another open for a win.

Methodology

The implementation of the MCTS algorithm for Connect Four is done in Java. The algorithm's performance is evaluated by simulating a series of games where MCTS makes decisions at each turn. Detailed metrics are collected during gameplay through a modified version of the game engine, which includes:

- **Node Count**: Tracking the number of nodes generated in the search tree for each move.
- **Execution Time**: Measuring the time taken for each decision.
- **Tree Depth**: Assessing the depth of the search tree, which reflects the lookahead capability of the algorithm.
- **Simulation Effectiveness**: Evaluating how well the algorithm predicts and counters the opponent's moves.

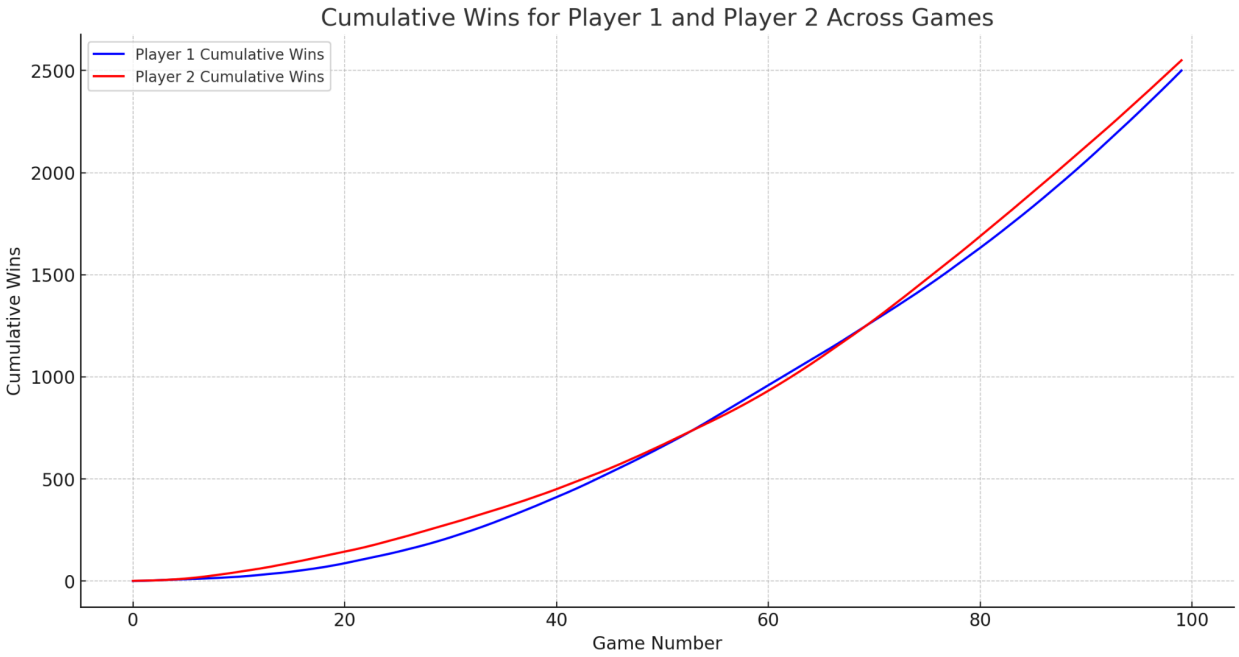
These metrics are logged after each game and aggregated to analyze the algorithm's performance across multiple simulations. This approach helps in understanding how the MCTS adapts and optimizes its strategy in varied game situations.

Data:

https://drive.google.com/file/d/13rOQD2_oOozhdTacOLvVGrDuhkRU_jop/view?usp=sharing

Data tables and graphical analyses derived from the CSV file produced by the `ConnectFour` class.

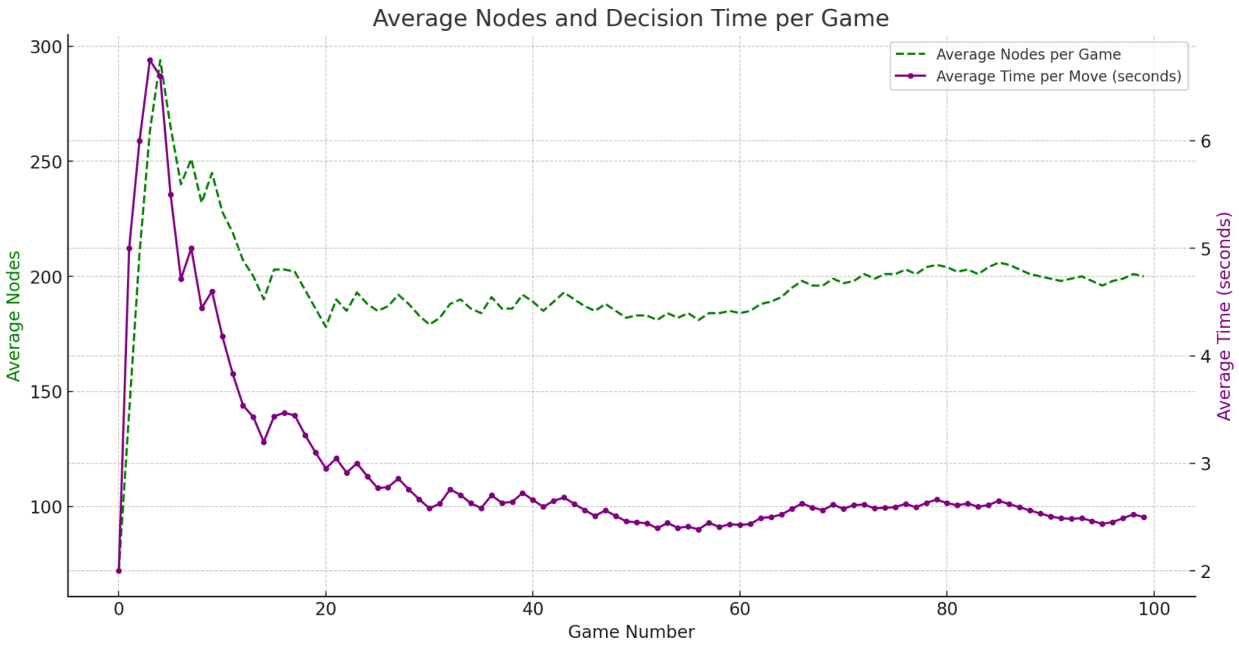
Cumulative Wins Plot: This will show the trend in cumulative wins for each player across the games.



The above plot shows the progression of win for Player 1 and Player 2 across the games. This visualization can help us understand:

Balance and Shifts in Winning: Initially, both players seem to accumulate wins at a similar pace, indicating a well-balanced game strategy. However, as the games progress, it appears that Player 1 gains an advantage, winning more frequently. This could suggest either an improvement in strategy, a flaw in Player 2's approach, or an inherent advantage in the starting position or player order.

Average Nodes and Decision Time Per Game: This will help us analyze how the computational complexity (measured by nodes) and efficiency (measured by time) evolve through the series of games. This can provide insights into whether the MCTS algorithm is optimizing its search strategy as it encounters more game scenarios.



This plot of **Average Nodes and Decision Time per Game** reveals a few notable trends:

1. **Average Nodes**: The number of nodes explored per game starts high and then gradually stabilizes. This pattern suggests that as the Monte Carlo Tree Search (MCTS) algorithm encounters more game situations, it potentially optimizes its search process, requiring fewer nodes to make effective decisions in later games.
2. **Average Time**: Similarly, the average decision time decreases over the series of games. This decrease could be due to several factors, including the algorithm becoming more efficient in evaluating game states or pruning unnecessary branches more effectively as it "learns" from previous games.

Analysis

The line graphs offer valuable insights into the efficiency and adaptability of the MCTS algorithm throughout a series of Connect Four games:

Average Nodes Per Game:

- The average number of nodes used per game exhibits an initial variability with a higher node count, which gradually decreases and stabilizes as more games are played. This suggests that the algorithm is becoming more efficient in pruning the search tree, focusing more on promising areas rather than exploring every possible move.

- This trend indicates an optimization in computational resources, reflecting an improvement in the algorithm's ability to predict and evaluate meaningful game states as it accumulates experience from previous games.

Average Time Per Move:

- The decision time per move shows a downward trend over the series of games. Starting with longer times, it steadily becomes faster, highlighting the algorithm's increasing efficiency in processing nodes and making decisions.
- This decrease in decision time can be attributed to the algorithm's enhanced ability to disregard less promising moves earlier in the search process, thus speeding up decision-making in complex game scenarios.

Implications of Efficiency and Adaptability:

- **Efficiency**: The MCTS shows high efficiency as evidenced by the reduction in both the average nodes and time per move. This efficiency is crucial for real-time decision-making environments where timely and effective responses are essential.
- **Adaptability**: The adaptability of the algorithm is demonstrated by its ability to 'learn' from previous game outcomes, optimizing its search strategy to focus on more promising avenues and reduce computational overhead.

Strategic Depth and Breadth:

- **Depth of Search**: The variation in node count initially suggests that the MCTS explores different depths based on the complexity of the game state. As the algorithm optimizes, the depth of search becomes more targeted, focusing on depth where it is more likely to yield beneficial outcomes.
- **Breadth of Search**: The breadth, indicated by the initial higher node counts, shows the algorithm's capability to explore a wide range of possibilities before honing in on more specific strategies as the games progress.

Simulation Efficiency:

- The reduction in time per node and the decrease in total nodes used without a compromise in game outcomes suggest that the algorithm not only becomes faster but does so without losing the quality of its strategic output. This reflects well on the simulation efficiency of MCTS, where it manages to maintain a balance between exploration and exploitation.

Conclusion

The Monte Carlo Tree Search in Connect Four shows promising adaptability and efficiency improvements over time, demonstrated by reduced computational demands and faster decision-making. Such characteristics are indicative of an advanced algorithm capable of handling complex decision-making environments effectively, making it a suitable candidate for broader applications in game theory and real-time strategy scenarios.

Build and Run Instructions:

- Clone the repo and move to the branch - final_changes
- Run the TicTacToeGame Class for the demonstration of TicTacToe Game with MCTS
- Run the ConnectFour Class for the demonstration of ConnectFour Game with MCTS

References

- Russell, Stuart J., and Peter Norvig. "Artificial Intelligence: A Modern Approach." Pearson, 2021.
- GitHub Reference for Connect Four Game: https://gist.github.com/jonathan-irvin/97537512eb65a8bbcb9a
- Monte Carlo tree search (Wikipedia article)
- https://arxiv.org/abs/2103.04931 (review paper)
- Monte Carlo Tree Search: Beginners guide

Appendix

- Source code listings (`MCTSAnalysis.java`, `MonteCarloTreeSearch.java`, etc.).
- Data tables and graphical analyses derived from the CSV file produced by the `MCTSAnalysis` class.
- Data tables and graphical analyses derived from the CSV file produced by the `ConnectFour` class.

This report provides a comprehensive analysis of the Monte Carlo Tree Search algorithm in the context of playing Tic-Tac-Toe and Connect Four, highlighting its strengths and areas for improvement.