Tic Tac Toe with Monte Carlo Tree Search Project
Milestone Report - INFO 6205: Program Structures and Algorithms Project

Raghavendra Prasath Sridhar - (002312779)
Nikhil Pandey - (002775062)
Ayush Patil - (002325566)
Project Repository GitHub Link: - https://github.com/raghavendraprasath/MCTSProject

---

Our team has successfully implemented a working version of the Tic Tac Toe game using Java Swing with Monte Carlo Tree Search (MCTS) logic for AI moves. The game includes features like single-player (vs AI) and multiplayer modes, difficulty levels, dark mode toggle, win/draw detection, replay options, and performance benchmarking (rollouts per move, time per simulation). We also incorporated a modern, responsive UI and ensured the code is modular and extendable for future game implementations. We have wrapped up final testing and documentation, and have pushed our progress to the GitHub repository for milestone submission. Additionally, we are in the process of brainstorming and deciding on our next game, which will also be built using the MCTS algorithm and extend our current design.

Project Features

- Monte Carlo Tree Search (MCTS) AI with adjustable difficulty (Easy, Medium, Hard)
- Human vs AI and Multiplayer modes
- Clean Swing-based GUI with dark mode and animations
- Real-time score tracking and win statistics
- Benchmarking and performance metrics using System.nanoTime()
- Modular, extensible codebase designed for future enhancements
- Unit test support for core game logic

How MCTS Was Used in This Project

The AI opponent in this project uses **Monte Carlo Tree Search (MCTS)** to simulate and evaluate possible move sequences. At each AI turn:

1. **State Simulation**: The current board state is wrapped in a TicTacToeState object, representing the game environment.
2. **Tree Construction**: A root TicTacToeNode is created to represent this state, and child nodes are generated based on valid moves.
3. **Rollouts**: The MCTS algorithm performs randomized simulations (also known as rollouts) from the current state to terminal states to evaluate outcomes.
4. **Backpropagation**: Results from these simulations are propagated back up the tree to help identify promising moves.
5. **Best Move Selection**: After the allotted number of simulations (based on difficulty), the AI selects the move with the highest win ratio.

This method ensures a balance between exploration and exploitation, making the AI more intelligent at higher difficulty levels. It's especially effective in small deterministic games like Tic Tac Toe and can be scaled to more complex games with minor adjustments.