Github - https://github.com/Hima200102/AI_PROJECT_HIMANSHU-NIMBARTE

Checkers Game AI: A Comparative Study of Depth Limited Minimax and Monte Carlo Tree Search Algorithms

ABSTRACT

This project presents a comprehensive exploration of artificial intelligence (AI) algorithms in the context of playing checkers. I implemented two prominent AI algorithms, Depth Limited Minimax and Monte Carlo Tree Search (MCTS), to create intelligent opponents for a console-based checkers game. The project includes a detailed code implementation, challenges faced during development, and extensive testing to evaluate the performance of these algorithms. Our comparative analysis of Minimax and MCTS sheds light on their respective strengths and weaknesses in decision-making processes within the game of checkers.

INTRODUCTION

The game of checkers has long served as a benchmark for evaluating artificial intelligence (AI) algorithms due to its complexity and strategic depth. In this project, delve into the realm of AI-driven checkers gameplay by implementing and comparing two prominent algorithms: Minimax and Monte Carlo Tree Search (MCTS).

Checkers, a classic board game played on an 8x8 grid, offers a rich environment for AI exploration. Players must navigate through a myriad of possible moves, considering both immediate tactics and long-term strategies to outmaneuver their opponents. This makes it an ideal testing ground for AI algorithms aiming to mimic human-like decision-making processes.

This project focuses on two primary AI algorithms: depth-limited Minimax and Monte Carlo Tree Search (MCTS). Minimax operates on the principle of

adversarial search and Monte Carlo Tree Search (MCTS) takes a more probabilistic approach to decision-making.

AI CONCEPTS

1] Minimax

The Minimax algorithm, a key component of decision-making and game theory, operates recursively, analyzing game trees to determine optimal moves. It assumes both players aim to maximize their own benefit while minimizing their opponent's. Commonly applied in AI for games like Chess, Checkers, and Tic-Tac-Toe, Minimax designates players as MAX and MIN, with MAX striving to maximize gains and MIN to minimize losses. Through a depth-first search, the algorithm explores the entire game tree, backtracking recursively to make informed decisions at each node.[1]

2] Depth Limited Minimax

Depth-limited Minimax diverges from the traditional Minimax algorithm by confining its exploration to a predetermined depth within the game tree. Rather than exhaustively traversing all potential game states, it evaluates positions up to this specified depth using a heuristic function. This function approximates the attractiveness of each board state, offering an indication of its favorability to the player. Through this bounded search depth, Depth-limited Minimax addresses the challenges posed by the expansive search space, rendering it viable for scenarios with limited computational resources.

3] Monte Carlo Tree Search

MCTS, or Monte Carlo Tree Search, blends Monte Carlo techniques with tree-based search strategies. Unlike traditional search algorithms, MCTS focuses on sampling and exploring promising areas of the search space rather than exhaustive exploration. It incrementally constructs a search tree by simulating multiple random plays from the current game state until reaching a terminal state or a predefined depth. The outcomes of these simulations are then propagated up the tree, updating node statistics like visit counts and win ratios.

Throughout the search process, MCTS dynamically balances exploration and exploitation. It prioritizes moves by considering both the exploitation of promising moves with high win ratios and the exploration of less explored options. This balance is achieved using a formula like Upper Confidence Bounds for Trees (UCT), which guides the selection of moves or nodes to visit during the search.[2]

CODE DESCRIPTION AND CHALLENGE

Board Class: Represents the game board and provides functionalities for initializing the board, moving pieces, checking legal moves, and evaluating the board state.

Minimax Player: Implements the Minimax algorithm with alpha-beta pruning for decision-making.

MCTS Player: Implements the MCTS algorithm for decision-making.

Node Class: Represents a node in the MCTS tree, containing information about the game state, parent node, move, children nodes, visits, and wins.

Checkers Game Manager: Manages the gameplay between the two AI players.

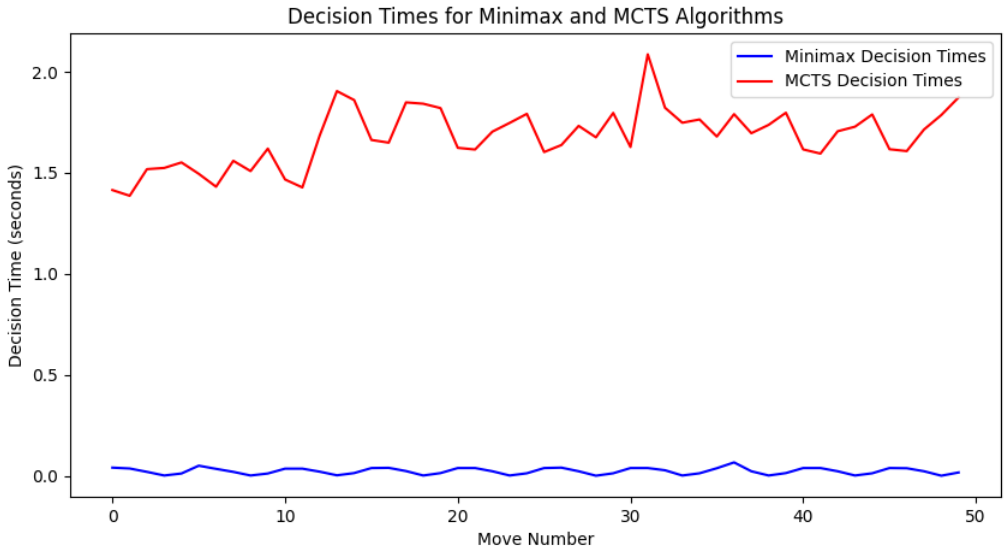
While implementing there were many challenges regarding infinity loops which I had to remove by debugging a lot which can be found in code where there are many commented print statements.

RESULTS

Conducted multiple test for the game and saw that Monte Carlo Search Tree outperforms Depth limited Minimax when it comes to score, out of 10 games played it won all the 10 games which can be seen in the figure

```
--- Results Summary ---
Total Games: 10
Minimax Wins: 0
MCTS Wins: 10
Draws: 0
Average Moves per Game: 10.0
```

but when it comes to amount of time it requires to predict steps it take more time as compared to minimax



CONCLUSION

Through this comparative analysis of depth-limited Minimax and Monte Carlo Tree Search (MCTS) algorithms in playing checkers, we have observed that MCTS outperforms Minimax in terms of overall performance. The findings suggest that for playing checkers, Monte Carlo Tree Search is a more effective and adaptable algorithm compared to depth-limited Minimax. But when it comes to processing time Monte Carlo takes more time as compared to minimax also it's important to note that the performance of these algorithms may vary depending on factors such as search depth,

computational resources, and the specific characteristics of the game being played.

REFERENCES

[1] https://www.javatpoint.com/mini-max-algorithm-in-ai#:~:text=The%20minimax%20algorithm%20performs%20a,the%20tree%20as%20the%20recursion.

Helped me understand minimax

[2] https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/

Helped me understand monte carlo tree search

[3] https://builtin.com/machine-learning/monte-carlo-tree-search

Understanding Monte Carlo