CS-171 Checkers Final AI Report

Team number: 39

Member #1([Name&NetID]): Jason He, jiach14 **Member #2: Lucas Zhuang, zhuangk2**

I. In about 1/2 page of text, describe what you did to make your Final AI agent "smart."

To make sure our AI agent is smart, we utilized the Monte Carlo Tree Search (MCTS) algorithm, enabling strategic decision-making by simulating various potential game results. MCTS involves four essential stages: selection, expansion, simulation, and backpropagation. In the selection process, the AI determines the most advantageous node using the Upper Confidence Bound (UCB) formula. It subsequently grows the tree by creating child nodes for moves that haven't been visited. During the simulation phase, the AI engages in random games from the expanded node to assess possible outcomes. Finally, backpropagation updates the values of the visited nodes based on the simulation results. Our AI effectively balances exploration and exploitation, ensuring that frequently visited and successful nodes are prioritized in future decisions.

To enhance performance, we implemented an optimized board representation, improving the speed of move generation and validation. We also optimized memory usage by minimizing unnecessary deep copies and ensuring efficient tree traversal. Additionally, we incorporated an adaptive time management strategy to ensure effective decision-making within computational limits. We used

```
def update_time_management(self, start_time):
        """ Updates time tracking for MCTS iterations. """
        self.total_time_remaining -= time.time() - start_time
        self.time_divisor -= 0.5 - (1 / self.timed_move_count)
        self.timed_move_count += 1
```

to dynamically allocate computation time based on the game's progression. Early-game moves, which require deeper exploration, were given more time, while later moves received less time as MCTS progressively refined its search efficiency. This adaptive approach ensured that our AI could conduct thorough searches when most impactful while maintaining a quick response time as the game progressed. In cases where time was limited, we implemented a fallback mechanism where the AI selected from a set of pre-evaluated moves, preventing time overruns and maintaining responsiveness.

Move Number	Time Remaining	Time Allocated	Move Number	Time Remaining	Time Allocated
1	480.0	24.5	5	405.8	23.8
2	460.41	24.5	6	388.82	23.4
3	441.62	24.3	7	372.24	23.0
4	423.47	24.0	8	256.12	22.7

II. In about 1/4 page of text, describe problems you encountered and how you solved them.

A major challenge we faced was handling memory efficiently. When simulating various game states, we initially created a deep copy of the board state for every simulation. This approach led to significant performance issues. We resolved it by minimizing unnecessary board copies and copying only when absolutely necessary during move execution.

Another challenge involved ensuring the AI accurately processed the opponent's actions when refreshing the MCTS tree. We addressed this by carefully tracking state transitions and maintaining correct parent-child relationships within the tree structure, ensuring consistent and accurate updates.

Additionally, we observed that our AI would occasionally make poor moves by placing itself in vulnerable positions easily exploited by the opponent. To address this, we integrated a heuristic evaluation function

```
if win_for_parent > 0:
                self.wins_for_parent += 1
          elif win_for_parent == 0:
                self.wins_for_parent += 0.5  # Tie
```

to guide the AI toward more positionally sound choices. This heuristic helped the AI better assess the strategic value of board states, resulting in more cautious and tactically aware decisions, especially in mid-to-late game scenarios. The overall performance and win rate of the agent improved noticeably after this adjustment.

III. In about 1/4 page of text, provide suggestions for improving your Final AI agent.

One potential improvement would be implementing a hybrid approach that combines MCTS with a heuristic-based evaluation function. While MCTS excels at long-term planning through simulations, it can sometimes overlook immediate tactical threats or opportunities. Integrating a heuristic would allow the AI to better evaluate board positions where deep simulations are unnecessary or computationally expensive. This hybrid model could provide stronger decision-making, particularly in endgames or highly constrained positions.

Another area for enhancement involves optimizing the simulation process. Currently, simulations are executed sequentially, which limits the number of rollouts within a given time frame. By parallelizing simulations across multiple CPU threads—or exploring GPU acceleration—we could significantly increase the rollout count, deepening the search and improving overall move quality. Furthermore, implementing an early cutoff mechanism to prune clearly losing branches during simulation would help focus computational resources on more promising paths.