RL Final Project: Monte Carlo Tree Search for Tic-Tac-Toe

Kyuho Lee

December, 2025

1 Introduction

Reinforcement learning and search-based decision-making algorithms have been widely studied in the context of sequential games. Among them, Monte Carlo Tree Search (MCTS) has demonstrated strong performance in various board games by balancing exploration and exploitation through stochastic simulations.

In this project, we apply Monte Carlo Tree Search to the classic Tic-Tac-Toe game played on a 3×3 board. Although Tic-Tac-Toe is a relatively simple game with a small state space, it provides a clear and interpretable environment for analyzing the behavior of MCTS, including its search tree structure, convergence properties, and sensitivity to the number of simulations.

The goals of this project are as follows:

- To implement the Monte Carlo Tree Search algorithm for a two-player zero-sum game.
- To analyze how the number of simulations affects the winning performance.
- To evaluate the effectiveness of different initial moves based on empirical winning rates.

2 Problem Description

The environment considered in this project is the standard Tic-Tac-Toe game played on a 3×3 board. The game is a turn-based, deterministic, two-player setting in which a computer agent competes against an opponent. No discount factor is applied, as the game terminates in a finite number of moves and rewards are only assigned at the end of each episode.

The reward structure is defined from the computer player's perspective. A reward of $+1$ is assigned when the computer wins the game, while a reward of -1 is given when the opponent wins. In the case of a draw, the reward is 0. No intermediate rewards are provided during gameplay, and all non-terminal moves yield a reward of 0.

This reward formulation defines a zero-sum game and enables the Monte Carlo Tree Search algorithm to evaluate terminal outcomes consistently across all game states. In this report, the computer player corresponds to the learning agent specified in the original assignment, and the reward signs strictly follow the problem definition, where a computer win yields $+1$ and an opponent win yields -1.

3 Monte Carlo Tree Search Algorithm

Monte Carlo Tree Search is a simulation-based search algorithm that incrementally builds a search tree by repeatedly sampling possible future game trajectories. Each iteration of MCTS consists of four main steps: Selection, Expansion, Simulation, and Backpropagation.

3.1 Selection

Starting from the root node, the algorithm recursively selects child nodes until it reaches a node that is either terminal or not fully expanded. During selection, the Upper Confidence Bound (UCB) criterion is used to balance exploration and exploitation.

For a child node i, the UCB score is defined as:

$$\text{UCB}_i = Q_i + C\sqrt{\frac{\ln N}{n_i}}$$

where Q_i is the average reward of node i, N is the number of visits to the parent node, n_i is the number of visits to node i, and C is the exploration constant.

3.2 Expansion

If the selected node is not terminal and has untried actions, one of the untried moves is randomly selected and added as a new child node. This step allows the search tree to grow progressively over multiple simulations.

3.3 Simulation

From the expanded node, a rollout (simulation) is performed by selecting moves uniformly at random until a terminal game state is reached. The final outcome of the game determines the reward returned by the simulation.

3.4 Backpropagation

The simulation result is propagated back through the nodes along the selected path. For each node, the visit count is incremented and the total reward is updated. The average reward of each node is computed as the total reward divided by the number of visits.

4 Implementation Details

The Monte Carlo Tree Search algorithm was implemented in Python. Each node in the search tree maintains information about the current game state, its parent node, the move that led to the current state, and a list of child nodes. In addition, each node stores the number of times it has been visited and the total accumulated reward obtained from simulation outcomes. These statistics are used to estimate the expected value of each node during the selection phase.

The search procedure executes a fixed number of simulations starting from the root state before selecting the final action. After all simulations are completed, the action corresponding to the child node with the highest visit count is selected, following the robust child criterion commonly used in MCTS.

Human interaction was supported by allowing a human player to input moves manually, while the computer player selects its moves using the MCTS algorithm. This setup enables direct evaluation of the learned policy through human-computer gameplay.

5 Experimental Setup

The experiments were designed to evaluate the performance of the MCTS algorithm under different simulation budgets and initial game configurations. The primary focus was to analyze how the

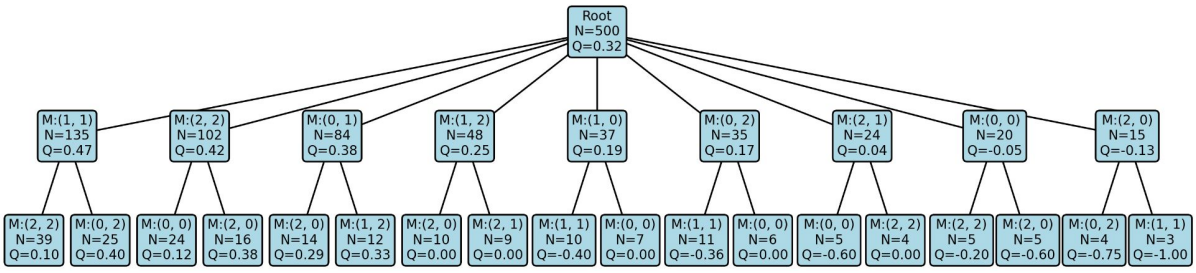

Figure 1: MCTS search tree after 500 simulations

number of simulations affects the winning performance and how the choice of the initial move influences the final outcome.

5.1 Simulation Settings

The numbers of simulations used in the experiments were 10, 50, 100, 200, and 500. For each configuration, 50 games were played to estimate the average winning rate. The opponent followed a random action selection policy.

5.2 Initial Move Evaluation

To evaluate the quality of different initial moves, the computer player was forced to place its first move at a specific board position. For each initial position, multiple independent games were played, and the average winning rate was computed as the primary evaluation metric.

All nine board positions were tested as possible initial moves. For each position, the number of repetitions was set to 50 games for each initial position.

6 Results

6.1 MCTS Search Tree Visualization

Figure 1 shows the Monte Carlo Tree Search (MCTS) search tree constructed after 500 simulations starting from the initial empty board state. Each node in the tree represents a game state reached by a specific move, and is annotated with the move taken (M), the number of visits (N), and the average reward value (Q) from the computer player's perspective.

At the first depth level, all nine possible initial moves are displayed. The visit counts are not uniformly distributed across these moves. The center position (1,1) receives the highest number of visits (N=135), while the remaining positions receive fewer visits, with counts ranging from N=102 to N=15. This concentration of visits suggests that, even in the early stage of the game, MCTS tends to favor certain initial moves that lead to more favorable outcomes during simulations, while allocating less search effort to moves that appear less promising.

The second depth level represents the opponent's responses to the computer's initial moves. For clarity, only the two most frequently visited opponent responses are visualized. These nodes correspond to the opponent actions that MCTS considers most critical under the assumption of adversarial play. Lower Q-values at this level indicate stronger opponent responses, as they lead to less favorable outcomes for the computer.

Overall, the search tree visualization demonstrates how MCTS balances exploration and exploitation. Visit counts concentrate on a small subset of promising branches, while less favorable

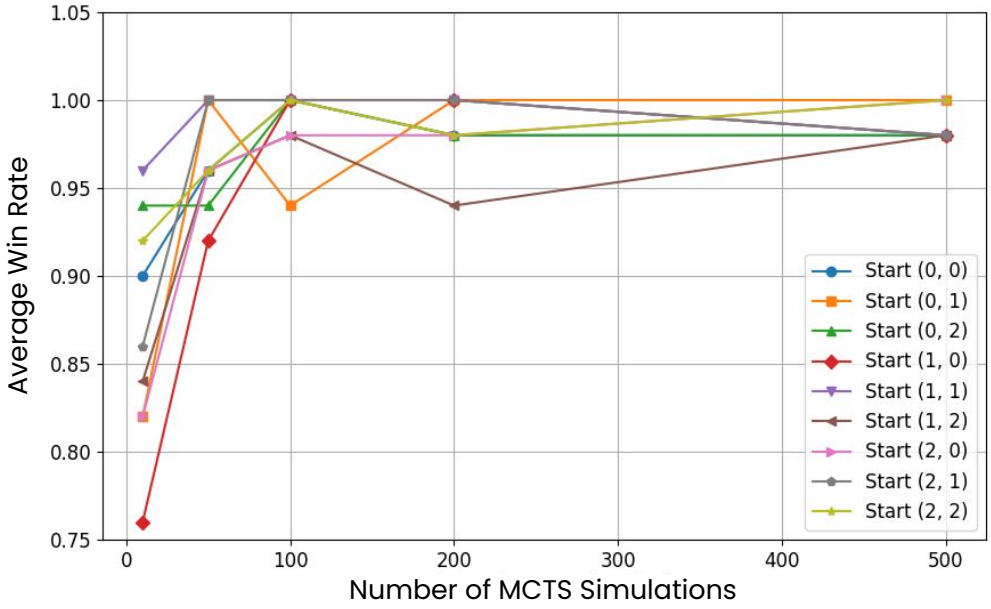

Figure 2: Average winning rate versus number of simulations for each initial position

moves are explored only sparsely. This behavior illustrates how MCTS incrementally focuses its search on strategically important actions based on empirical simulation outcomes.

6.2 Winning Rate Analysis by Initial Move

Figure 2 shows the average winning rate of the MCTS agent as a function of the number of simulations for each possible initial board position. The plot is used to compare how performance varies across different initial moves under different simulation budgets.

The average winning rate is defined as the proportion of games won by the computer agent. Draws are counted as non-winning outcomes, and losses are counted as zero wins.

When the number of simulations is small (10 simulations), noticeable differences in average winning rate are observed across initial positions. In this regime, the winning rate ranges approximately from 0.76 to 0.96, indicating that the choice of the initial move has a strong impact on performance. The center position (1,1) achieves an average winning rate of approximately 0.96, which is higher than that of the other initial positions. As the number of simulations increases to 50, the average winning rate for most initial positions rises rapidly, and the performance gap between different positions generally decreases. However, not all initial positions exhibit the same level of performance in this range, and some positions still show relatively lower winning rates. The center position (1,1) continues to maintain a comparatively high and stable winning rate.

With 100 or more simulations, the average winning rates for most initial positions remain at a high level, while some positions still exhibit lower values compared to others. For example, at 100 simulations, the initial position (0,1) records the lowest average winning rate, and at 200 simulations, the position (1,2) shows a relatively lower value. These positions correspond to edge positions on the board.

In the ranges of 200 and 500 simulations, high average winning rates are maintained overall, and the differences in performance across initial positions are further reduced. In these ranges, additional increases in the number of simulations result in relatively limited performance improvements.

Overall, as the number of simulations increases, the differences in average winning rate across initial positions tend to decrease. However, under limited simulation budgets, the center position (1,1) consistently achieves the highest average winning rate and exhibits the most stable performance. This superiority of the center position can be attributed to the strategic nature of Tic-Tac-Toe; occupying the center allows the maximum number of potential winning lines (four lines: horizontal, vertical, and two diagonals), whereas edge or corner positions offer fewer strategic options. Based on these experimental results and strategic analysis, the center position (1,1) is identified as the best initial move.

7 Conclusion

In this project, we implemented and evaluated the Monte Carlo Tree Search algorithm for the game of Tic-Tac-Toe. Through the analysis of search tree structures and winning rates across varying simulation budgets, we verified that MCTS effectively balances exploration and exploitation to identify optimal moves.

The experimental results demonstrated that the agent's performance improves with the number of simulations and converges to a high winning rate. Furthermore, the comparative analysis of initial positions confirmed that the center position (1,1) is the most advantageous starting move.

Finally, through the implemented human-computer interface, we qualitatively verified the agent's robustness; even against a human player, the MCTS agent successfully identified and blocked potential winning threats, demonstrating effective defensive play beyond simple random simulations.