COMP 560 Project: Ultimate TicTacToe

Cheng Zhi Sheng, Cao Duy Nguyen, Ma Xiaoxiao, Lai Yee Teng Victor
University of North Carolina, Chapel Hill
April 2025

Abstract

Ultimate Tic-Tac-Toe is a more strategic version of the classic game, making it a fun challenge for AI. In this project, we built two AI players using Minimax and Monte Carlo Tree Search (MCTS) to see which performs better. We tested both approaches against the boss levels in CodinGame and analyzed their strengths, weaknesses, and overall performance.

1 Game Rules of Ultimate Tic-Tac-Toe

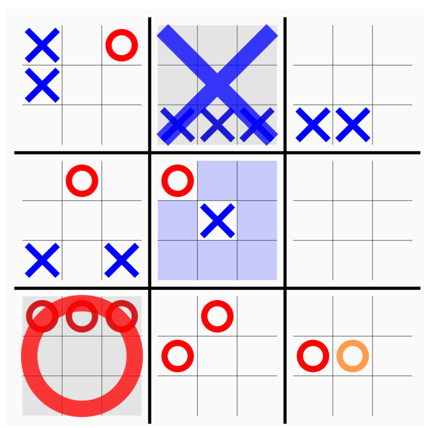

Figure 1: A sample Ultimate Tic-Tac-Toe board

Ultimate Tic-Tac-Toe is a strategic twist on the classic game. It consists of a 3×3 grid of smaller 3×3 Tic-Tac-Toe boards. The game follows these rules:

Rule Description
1. Two players (X and O) take turns. 2. Each move is on a cell in a 3×3 small board. 3. The chosen cell dictates which small board the opponent must play in. 4. If that board is full or won, the next player can choose any ongoing board. 5. A small board is won by 3 marks in a row (like classic Tic-Tac-Toe). 6. The main objective is to win 3 small boards in a row on the main board. 7. If no such alignment is achieved, the player with the most small board wins prevails. 8. If both win the same number of small boards and no alignment occurs, the game is a draw.

Table 1: Ultimate Tic-Tac-Toe Rules Summary

2 Setup for Game

Our implementation consists of two components: a local development environment and a competitive evaluation platform. For local development, we implemented our agents in both Python and C++ to facilitate testing and refinement. The core algorithms—Minimax and Monte Carlo Tree Search—were implemented with optimizations specific to the Ultimate Tic-Tac-Toe domain.

For evaluation and competitive benchmarking, we utilized CodinGame Ultimate Tic-Tac-Toe arena[1], a platform that provides standardized testing against pre-defined bots and peer submissions. The platform implements the following competitive structure:

- **League System**: Players progress through a series of increasingly difficult leagues: Wood, Bronze, Silver, Gold, and Legend.
- **Boss Challenges**: To advance to the next league, an agent must defeat the current league's "Boss"—an AI opponent with predetermined skill level.
- **Peer Competition**: Within each league, agents compete against other user-submitted bots, generating relative performance scores.
- **Computational Constraints**: Each agent operates under strict time and memory limitations (50ms per turn, 768MB memory), enforcing efficient algorithm implementation.

This competitive framework provides an objective measure of agent performance and facilitates rapid iteration by exposing weaknesses against diverse opponent strategies. Score aggregation is based on win/loss ratios against multiple opponents, providing a robust metric for evaluating overall agent strength.

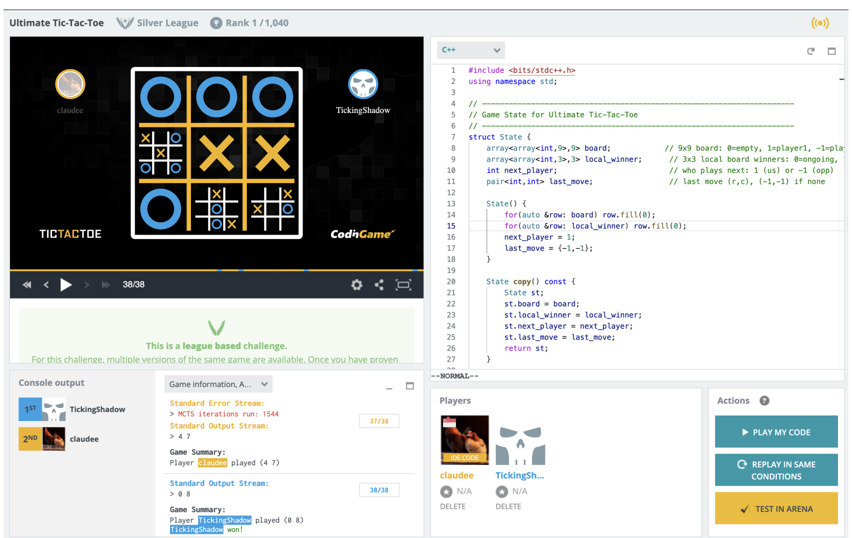

Figure 2: CodinGame interface: game board, league progression system, and competitive rankings

3 Implementation 1: Minimax Algorithm

This paper presents an AI implementation for Ultimate Tic-Tac-Toe using the Minimax algorithm with alpha-beta pruning and a heuristic evaluation. The AI plays optimally within computational limits, aiming to win local boards and ultimately the global board.

The core of the AI is the Minimax algorithm, which evaluates possible moves recursively. Alpha-beta pruning helps to skip irrelevant branches, reducing computation time.

At each step:

- **Maximizing player** ('X') tries to maximize the score.

- **Minimizing player** ('O') tries to minimize the score.

When the game reaches a terminal state (win/loss/draw), a base score is returned. Otherwise, the algorithm explores further moves.

3.1 Heuristic Evaluation Function

Since full-depth search is infeasible in early and mid-game, we use a heuristic to estimate board quality. At any given move, the AI will search the next move (maximum 9 choices), and come up with a heuristic score. The function considers 3x3 sub-boards and gives:

Board Pattern	**Score**
Board won by 'X' (terminal state)	$+10$
Board won by 'O' (terminal state)	-10
Line with two 'X's and one empty cell (chance to win)	$+3$
Line with two 'O's and one empty cell (need to block)	-4
Line with one 'X' and two empty cells (good formation)	$+1$
Line with one 'O' and two empty cells (discourage setup)	-1

Table 2: Heuristic Evaluation Scores for Local Board States

3.2 Performance

The AI uses heuristics mainly to avoid computational load evaluating the full depth search of the tree. With the heuristic in 3.1, the AI was able to achieve a score of 31.90 and reached 1st place of the Bronze League.

3.3 Conclusion

While our Minimax implementation with heuristics provided strategic depth, it failed to overcome the Bronze League benchmark despite multiple heuristic refinements. This limitation prompted our exploration of alternative approaches.

4 Implementation 2: Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) offers an alternative to Minimax by removing the need for handcrafted heuristic functions [2]. Instead, MCTS evaluates positions through randomized playouts, learning how strong each move is based on simulated outcomes. This makes it especially suitable for complex environments like Ultimate Tic-Tac-Toe, where defining an accurate heuristic is challenging.

4.1 The MCTS Algorithm

MCTS follows a four-step iterative process:

1. **Selection**: Starting at the root node (current game state), recursively select child nodes using the Upper Confidence Bound (UCB1) formula until a leaf node is reached.

$$\text{UCT}(i) = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}$$

   - w_i: total reward from simulations passing through node i,
   - n_i: number of visits to node i,
   - N: number of visits to the parent node,
   - c: exploration constant, set to $\sqrt{2}$ to balance exploration and exploitation.

2. **Expansion**: If the selected node is not terminal and has unexplored children, one of them is added to the tree.

3. **Simulation**: From the new node, a game is played out to the end using random moves. The result (win/loss/draw) is recorded.
4. **Backpropagation**: The outcome is propagated back through the visited nodes, updating w_i and n_i accordingly.

Over time, the tree focuses more on moves that lead to favorable outcomes, implicitly learning strong strategies without human-designed heuristics.

4.2 Performance

Our Python implementation achieves approximately 150 rollouts on the second turn within a 100ms time budget. A port to C++ improves performance, reaching around 1000 rollouts under the same time constraint. Further optimization was achieved through:

- **Bitmasking**: Each 3x3 local board and the global state are represented as bitmasks using built-in integer type, significantly reducing memory and improving cache performance.
- **Bitwise win-checking**: Win conditions are evaluated using fast bitwise operations rather than nested loops.

Using MCTS, our AI consistently outperformed the Minimax implementation and managed to defeat the boss to progress to Silver and then Gold League on CodinGame.

4.3 Conclusion

MCTS provides a flexible and powerful method for decision-making in Ultimate Tic-Tac-Toe, avoiding the limitations of static heuristics. The algorithm improves dynamically with more rollouts, and performance scales well with optimization. However, the inherent randomness of rollouts can still lead to suboptimal decisions under strict time limits, especially in deeply tactical scenarios.

5 Conclusion

In this project, we developed and evaluated two AI strategies for playing Ultimate Tic-Tac-Toe: a heuristic-based Minimax algorithm and a Monte Carlo Tree Search (MCTS) approach. While the Minimax agent demonstrated solid strategic play using handcrafted heuristics, it was ultimately limited by shallow search depths and heuristic design biases. In contrast, our MCTS agent, driven by random simulations and statistical averaging, consistently outperformed the Minimax agent across multiple game scenarios.

Our experiments, including matches against the CodinGame boss bots, showed that MCTS achieved higher scores and demonstrated more robust decision-making under uncertainty. This performance gap underscores the advantage of probabilistic planning in complex environments like Ultimate Tic-Tac-Toe, where the search space is vast and deterministic strategies may struggle.

Future work may explore hybrid approaches that combine the strategic insight of heuristics with the exploration power of MCTS, as well as optimizing simulation policies and rollouts to further boost performance. Overall, MCTS proves to be a more effective method for this challenging and strategic variant of Tic-Tac-Toe.

References

[1] Ultimate tic-tac-toe - Codingame. `https://www.codingame.com/multiplayer/bot-programming/tic-tac-toe`. Accessed: 2025-04-24.

[2] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.