AI agent in Chinese Chess

Virginia Tech CS 5804

Team Members: Kevin Lin
Haowen Zhang
Qitao Yang
Kunal Nakka
Bhargava Elavarthi
Srikaran Bachu

May 11, 2025

1. Research Problem, Approaches, Results

Chinese Chess (Xiangqi) offers an intriguing challenge for classical game AI research: While it shares the perfect information, two player zero sum structure of Western chess, its effective branching factor is even greater because cannons are captured by leaping, soldiers gain lateral mobility after crossing the river, and the two generals cannot 'see' each other across an open file without intervening pieces. This makes the total number of legal games roughly 10^{120}, making it infeasible to traverse a full game tree. The project investigates the effectiveness of search algorithms: depth limited minimax, alpha-beta pruning, and a simple greedy evaluator inside a lightweight Pygame environment, while still allowing a human player to move by mouse clicks and to face any combination of AI agents or another human.

We used the Minimax algorithm, a decision rule used in two player games to minimize the possible loss for a worst case scenario. We choose this as it is a solid method to use in a two player strategy based game.

In our implementation, the AI plays as the Red side, while the human user plays as the Black side. The core of the Minimax strategy is simulating all possible moves up to a fixed depth and then assigning a score to each potential board state using a custom evaluation function. This evaluation function computes a heuristic score based on material value, so higher weights are given to more important pieces like a cannon or the general.

To simulate each move, we used deep copies of the current game state to ensure the integrity of the ongoing game is preserved during the AI turn. Each piece in the game has a corresponding rule based movement function (soldier_rule, cannon_rule etc.) that checks for valid destinations and handles both moves and captures.

Once all possible future states are evaluated, the AI selects the move that maximizes its minimum possible loss. Although this method can lead to non aggressive moves like moving a soldier back and forth, it establishes a foundation for good moves. Plans for the future are integrating Alpha-Beta Pruning to boost the efficiency and exploring deeper search depths for better behavior.

The goal of our project was to implement an algorithm that produces quality moves, while still trying to minimize the time taken per move.

Looking at our finished implementation there are several observations that can be made. We successfully created a very functional model that allows users to play one on one with the AI while the AI is performing only valid moves according to the Chinese chess rules. Our AI is able to respond in a timely manner while incorporating a 2-ply depth ensuring the move is optimal. While making these moves, we successfully ensured the AI makes moves on the updated state which was a very time consuming issue we had to overcome in the early days of this project.

As we have a depth of 2, the AI sometimes makes repetitive moves in situations where having a shallow Minimiax prevents future foresight and prompts our AI to maintain the current position instead of staying aggressive. This would be the one weak point to this approach.

Overall, the results validate the effectiveness of Minimax in the game of Chinese Chess. Our system allows for a stable user experience, and creates the groundwork for future improvements through pruning and adaptive learning.

2. Code Explanation and Running Instructions

The majority of the code base is contained in a single script, app.py, accompanied by a small images/ directory that holds the board graphic and fifteen 50 × 50 px PNG sprites. Board state is maintained in two sixteen element lists, `red_chess` and `black_chess`, whose entries hold piece coordinates $[x, y]$ or the sentinel $[-1, -1]$ after capture. Seven rule functions—`soldier_rule`, `car_rule`, `cannon_rule`, `horse_rule`, `elephant_rule`, `attendant_rule` and `boss_rule`—encode legality, returning `0` for an illegal move, $[\text{new}, 1]$ for a quiet move, or $[\text{new}, 2, \text{capIdx}]$ when a capture is possible. On top of these primitives, `get_all_moves` enumerates every legal action, `simulate_move` applies a candidate move on deep copies of the lists, and `evaluate_board` supplies a material only heuristic. The main AI agent, selected as *Search 1*, is a depth-limited α–β minimax implemented in `alphabeta`; its driver `find_best_move` iterates over Red's legal moves, evaluates each reply tree, and returns the maximising choice. Because search operates exclusively on deep copies, the displayed board is never altered during look ahead.

All input and rendering are handled by Pygame. `draw_chess` blits the board and active pieces, `draw_text` prints centred status strings, and `highlight_piece` outlines a selected square. The main loop alternates between an AI turn (Red) and a human turn (Black). During the AI turn the program calls `find_best_move`, updates the state through `simulate_move`, refreshes the display, and checks for mate. During the human turn the program waits for two left clicks first on a piece, then on a destination square validates the attempt with the same rule functions, updates the lists in place, and again redraws the board. After every legal move, the engine invokes `is_check` and `is_checkmate`; capturing a general or delivering an inescapable check announces the winner and freezes the position.

To run the project, install the sole third party dependency with `pip install pygame`, ensure that `chinese_chess.py` and the `images/` directory are colocated, and start the game with `python chinese_chess.py`. A 450×550 pixel window opens in which Red, driven by depth-2 α–β search, moves first; Black is controlled by the user via two consecutive clicks per move. Illegal destinations are ignored, and on a mid range laptop the AI responds in well under 0.1 s at depth 2, providing smooth, real time play while showcasing classical search techniques for Xiangqi.

3. Future Work

There is still much space for improvement, even though our AI agent currently uses traditional algorithms like Minimax and Alpha-Beta Pruning. The incorporation of Monte Carlo Tree Search (MCTS) is one encouraging avenue. MCTS simulates numerous random playouts for every possible move, in contrast to deterministic algorithms, and bases its decisions on the statistical results of those simulations. This would enable the agent to more effectively assess long-term effects and adjust more quickly to complex or uncertain board states. When paired with Alpha-Beta, MCTS may provide a hybrid approach that strikes a balance between probabilistic exploration and strategic depth.

Optimizing move ordering in the Alpha-Beta Pruning procedure is another crucial area that needs work. We can greatly increase pruning efficiency by rearranging moves according to tactical urgency, such as giving high-value captures or checkmate threats priority. Previous studies have demonstrated that intelligent move ordering can increase search speed by as much as 40% at deeper levels. Decision-making could become stronger and faster by implementing a dynamic priority system, which would allow for deeper lookahead within the same computational budget.

We also intend to introduce AI vs. AI matchups. We can more accurately evaluate the advantages and disadvantages of each algorithm by contrasting Greedy, Minimax, Alpha-Beta, and possibly other learning-based agents in head-to-head games. Additionally, these matchups would inform how various evaluation functions behave under various game scenarios.

Lastly, our agent could make better decisions without sacrificing game responsiveness if it could support deeper search depths with effective memory handling and possible multithreading or multiprocessing. Future optimization initiatives should concentrate on scalable methods that enable intricate evaluations without sacrificing performance, given that the board state space grows exponentially with each ply. Together, these improvements have the potential to transform our AI into a highly skilled and competitive Chinese chess agent.

4. Lessons Learned

We faced several conceptual and technical obstacles during this project, which greatly aided in our learning. At first, it was harder than expected to fully comprehend the mechanics and rules of Chinese chess. Some of the team had never played Chinese chess at all. We also needed to get acquainted with Pygame, which was unfamiliar to a few team members. In order to create a functional and engaging GUI.

Experimenting with search depth for our AI agent was another important learning experience. We started to comprehend the trade-off between computation time and decision quality through trial and error. Although a deeper search frequently produced better results, it did so at the expense of noticeably slower performance. This made it necessary for us to balance responsiveness and performance, particularly when using algorithms like Minimax and Alpha-Beta pruning. These experiments demonstrated the value of optimization techniques in AI implementation and helped us understand the practical limitations of game tree search.