**CS311 Final Project – Gomoku Solver**

Jiayi Chen and Siyuan Niu

**Background**

Gomoku, or five in a row, is an abstract strategy board game where two players place a stone—either black or white—on a 15x15 board. The winner is the first to form an unbroken chain of five stones horizontally, vertically, or diagonally. Minimax algorithm is widely used in board games involving two-player competition such as Tic-Tac-Toe [1]. In 1992, Vardi [2] added expected values and utility into the minimax algorithm, making it possible to use evaluation functions to predict and calculates the possible scenarios. However, due to minimax’s time complexity, researchers have implemented various methods to improve its search performance especially in more complex games such as Gomoku. Typical methods include alpha-beta pruning that eliminates unpromising nodes [3], Monte Carlo Search Trees that adds randomization [4], and various heuristic functions [3]-[5].

We intend to implement the minimax algorithm with alpha-beta pruning and heuristic functions in Gomoku. We also explore how limiting relevant moves and different depth limits affect algorithm performance based on both search time and wining chances.

**Methods**

Minimax algorithm helps the agent to minimize risks by assuming that the opponent will always choose optimally and propagating minimax values back “upwards” recursively. Given that a tree has exponential features, it is important to limit both the branching factor and searching depth limit. We restrict the branching factor by limiting the next possible moves to the eight neighbors of stones that are already placed, since nonadjacent moves are often considered as unpromising in a real game [6].

However, there exists a tradeoff between depth limit and minimax algorithm’s optimality. Therefore, we evaluate the algorithm with different depth limits. When the minimax tree reaches the predetermined depth limit, an evaluation function is invoked (Eq. 1) to analyze each possible move based on the number of threat patterns – the number of 2, 3, or 4 connected stones on the board. A pattern can be either “open” or “half”: an open pattern is not blocked by an opponent stone on either side and a half pattern is blocked by one.

*Eq. 1:**Adopted from [2] and tuned some parameter values; we want to*  *emphasize the importance of connected open and half 4.*

**Results**

We first validate the efficiency of the alpha-beta pruning in the minimax tree process, as Fig. 1 illustrates.

Chart, line chart

Description automatically generated

*Fig. 1:**Alpha-beta pruning reduces the time needed for training a minimax tree.*

To evaluate the performance of our algorithm, we use a total of nine different boards to compute the results. We divide those nine boards into three categories: easy, medium, and hard. Difficulty represents the minimum number of moves for black to win. For example, black may need only one correct move to win on an easy board, while five or more moves on a difficult one. Based on a controlled case where black and white place simple random moves, we compare black's win rates - powered by minimax - across boards of different difficulties as well as those given the same difficulty but with depth limits of 1 and 2.

Chart, bar chart

Description automatically generated

*Fig. 2: As the difficulty of the board increases from easy to difficult, black random’s win rate drops from 76% to 50%. In contrast, although minimax’s depth setting is low, black achieves a 100% win rate against white playing with random moves.*

|  |
| --- |
|  |

**Conclusions**

As discussed before, time complexity is a key constraint of minimax, and alpha-beta pruning optimizes the time complexity to in the best scenario [3]. We are able to replicate this improvement i our implementation, we replicate the result where alpha-beta pruning improves the original minimax algorithm, especially for a larger depth limit (Fig. 1).

As shown in Fig. 2, even with a depth limit of 1 or even 0, the minimax algorithm can still beat an opponent with random moves in every game. This is due to the nature of Gomoku: in the case of a depth limit of 0, for instance, black only uses the evaluation function once to play and does not predict any of white’s moves. However, black is still making progress towards its goal of connecting five stones, while white is just making random moves and cannot even block black’s path correctly.

**References**

[1] Y. PIRILDAK, “Mastering Tic-Tac-Toe with Minimax Algorithm,” Medium, May 13, 2020. https://levelup.gitconnected.com/mastering-tic-tac-toe-with-minimax-algorithm-3394d65fa88f (accessed Dec. 10, 2022).

[2] A. Vardi, “New minimax algorithm,” J Optim Theory Appl, vol. 75, no. 3, pp. 613–634, Dec. 1992, doi: 10.1007/BF00940496.

[3] E. Nygren, “Design Specifications for an Interactive Teaching Tool for Game AI using Gomoku”.

[4] Yu. (2019). AI Agent for Playing Gomoku. Retrieved December 9, 2022. <https://stanford-cs221.github.io/autumn2019-extra/posters/14.pdf>

[5] H. Liao, “New heuristic algorithm to improve the Minimax for Gomoku artificial intelligence”.

[6] “Minimax Improvements.” https://blog.theofekfoundation.org/artificial-intelligence/2015/12/18/minimax-improvements/ (accessed Dec. 17, 2022).

|  |
| --- |
|  |
|  |