CS311 Final Project – Gomoku Solver

Jiayi Chen, Siyuan Niu

Background

Gomoku, or five in a row, is an abstract strategy board game where two players place a stone of their color on a 15x15 empty intersection. The winner is the first whose stones form an unbroken chain of five horizontally, vertically, or diagonally. Our goal is to design an agent that plays with a human player in real time. There exists literature using the Expectiminimax to solve the Tic-Tac-Toe problem [1] but given the larger board and more complex strategies of Gomoku, we intend to evaluate Expectminimax's performance and maybe improve its efficiency by, for example, pruning and incorporating heuristic functions.

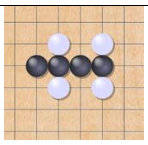
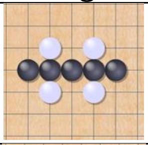
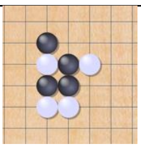
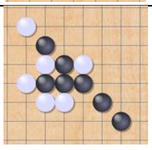
Methods

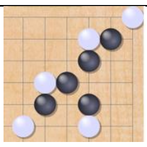
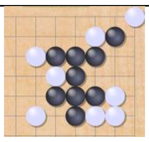
The algorithm we choose for a Gomoku solver is an Expectiminimax, which minimizes risks by assuming that the opponent will always choose optimally and propagating Minimax values back "upwards" recursively. Meanwhile, we incorporate an evaluation function calculating the optimality of each possible move. In order to avoid a lengthy exhaustive search, we also set a depth limit and restrict the branching factor by identifying the eight neighbors of the last opponent move as the next possible moves.

We define our evaluation function of each possible move based on the number of threat patterns, i.e. 2, 3, or 4 connected stones, brought by the move. Meanwhile, 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.

$$\begin{aligned}\boldsymbol{Eval(board)} = \mathbf{10000} * (N_5^{Black} - N_5^{White}) \\ + \mathbf{5000} * (N_{open4}^{Black} - N_{open4}^{White}) + \mathbf{2500} * (N_{half4}^{Black} - N_{half4}^{White}) \\ + \mathbf{2000} * (N_{open3}^{Black} - N_{open3}^{White}) + \mathbf{1000} * (N_{half3}^{Black} - N_{half3}^{White}) \\ + \mathbf{250} * (N_{open2}^{Black} - N_{open2}^{White}) + \mathbf{50} * (N_{half2}^{Black} - N_{half2}^{White})\end{aligned}$$

Results

Complexity	Depth Limit	Pre-terminal State	Winning State	Time
Simple	1			0.12s
Intermediate	5			5600s ≈ 1.6hrs

Advanced	7			14700s ≈ 40.8hrs

Conclusions and Discussions

As our test cases illustrate, our algorithm can solve all test cases with the proper depth limits. Our proposed evaluation function also outperforms the existing one - given the much larger parameter we assign on the 5-stone patterns, our model will be more likely to identify the winner with five stones already. Contrastingly, in some evaluation functions we found in the existing literature [1], only a larger but not necessarily a dominant one is assigned. This will fail to identify the winner sometimes especially in cases when black has a connected-five, while white has many connected-three or four. The situation is similar given a small parameter on the 4-stone patterns, since a winner is also determined when there is an open four unless it is the opponent's turn who has also obtained an open four.

Throughout the project, we did encounter some challenges. Firstly, we spent a large amount of time on the implementation and testing the function calculating the number of threat patterns. Our second challenge is due to the inherent time complexity of the Expectiminimax. Although we limit the branching factor by restricting the number of possible moves, the search duration remains long, which limits our algorithm's capability in a broader context. We can solve the challenge by alpha-beta pruning eliminating the unpromising nodes.

Other modifications include, for example, adjusting the evaluation function to suit different assessing criteria, such as winning chances and the number of steps to beat the opponent. However, a trade-off may exist between winning the game in a timely manner and exploring more promising cases.

References

[1] Pirildak, Y. (2022, January 30). Mastering Tic-Tac-Toe with Minimax Algorithm in Python. Medium.https://levelup.gitconnected.com/mastering-tic-tac-toe-with-minimax-algorithm-3394d65fa88f
[2] Yu. (2019). AI Agent for Playing Gomoku. Retrieved December 9, 2022. https://stanford-cs221.github.io/autumn2019-extra/posters/14.pdf