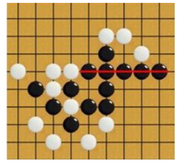
Gomoku Solver

Middlebury

Jiayi Chen and Siyuan Niu

Introduction

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. It is similar to the Tic-Tac-Toe, but it is much more difficult to solve.

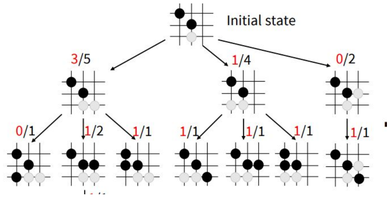


Gomoku Example - Black wins given a row of 5 connected stones

We intend to implement an Expectiminimax that optimizes both search efficiency and the chances of winning. Starting from a smaller 9x9 board, we modify the evaluation function to improve the overall model performance. Our ultimate goal is to design and optimize a real-time Gomoku agent that can play against a human player.

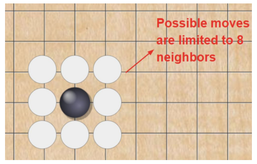
Methodology

The algorithm we propose is an Expectiminimax which minimizes risks by assuming 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. To avoid a lengthy exhaustive search, we also set a depth limit and restrict the branching factor by limiting the number of possible moves.

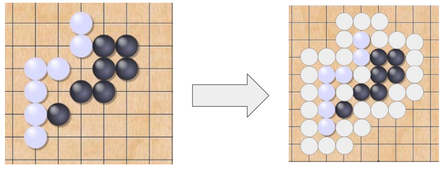


Possible Moves

To implement the algorithm, the first step is to determine all the possible searching steps. To improve the searching efficiency and avoid a large branching factor, we assume the opponent will only place a stone in the 8 adjacent cells of the agent’s last move. The assumption makes sense since the moves are also what most players would take in a real game.



Simple Example



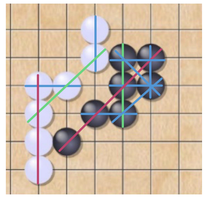
Complex Example

Evaluation Function

We define our evaluation function of each possible move based on the number of threat patterns resulted on the board, i.e. 2, 3, or 4 connected stone. 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.

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

Evaluation Function



$$\begin{array}{ll} \text{For Black:} & \text{For White:} \\ N_{open5} = 0 & N_{open5} = 0 \\ N_{open4} = 0 & N_{open4} = 1 \\ N_{half4} = 1 & N_{half4} = 0 \\ N_{open3} = 1 & N_{open3} = 1 \\ N_{half3} = 0 & N_{half3} = 0 \\ N_{open2} = 4 & N_{open2} = 2 \\ N_{half2} = 1 & N_{half2} = 0 \end{array}$$

$$\begin{aligned} Eval(example) = {} & 10000 * (0 - 0) \\ & + 5000 * (0 - 1) + 2500 * (1 - 0) \\ & + 2000 * (1 - 1) + 1000 * (0 - 0) \\ & + 250 * (4 - 2) + 50 * (1 - 0) \\ & = -1950 \end{aligned}$$

Sample Board Score

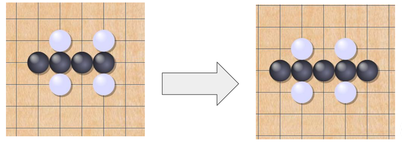
Pseudocode

```
def minimax(board: Gomoku, maximizing: bool):
    # base case
    if board.is_terminal() or curr_depth > depth_limit
        return eval(board)

    # recursion
    scores = []
    for board  in board.get_possible_moves():
        scores.append(minimax(board, not maximizing))
    if maximizing:
        return max(scores)
    else:
        return min(scores)
```

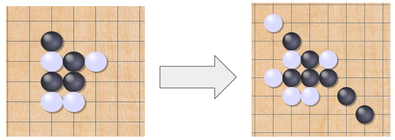
Result

- Sample tests (assuming Black goes first):
  - Simple Case (depth limit = 1):



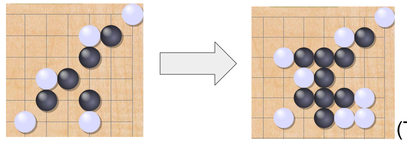
(Took 0.12s)

  - Intermediate Case (depth limit = 5):



(Took 5600s ≈ 1.6 hours)

  - Advanced Case (depth limit = 7):



(Took 14700s ≈ 40.8 hours)

Discussion and Future Steps

- Discussion
  - Our algorithm successfully solved all test cases with proper depth limits. The high branching factor, however, makes the computational complexity huge even with a slight increase in depth.
  - Compared with previously mentioned evaluation functions, our newly proposed function seems to have better performance in a more complicated setting.
  - Currently only assessing performance by the searching duration
  - If assessed by winning chances, the 8 possible moves is a limited assumption
- Future steps
  - Computational Complexity: Alpha-beta pruning
  - Modify evaluation functions by
    - Winning-chance based model assessment
    - Including non-consecutive threat patterns

References

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

[2] 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