Gomoku AI Using Monte Carlo Tree Search

Yijia Wang 18340246003

Yaochen Shen 18340246004

Introduction

1. Basic Understanding

Gomoku, also known as wuziqi or five in a row, is a popular strategic board game originating in Japan since Meiji Restoration period as the name is actually referred to as gomokunarable. It was later introduced to many counties around the world and became popular there as well including China, Korea, and Britain. The standard rule of it is that two players with Go pieces (black and white, black goes first) place their pieces on the 15*15 board by turns and the winner is the one that obtains an unbreakable row of five his or her pieces horizontally, vertically, and diagonally.

2. Under the Intervention of AI

It has been a long history since people started trying to apply Artificial Intelligence to Gomoku since the computer definitely possesses a much better ability with analyzing the circumstances and learning from tons of Gomoku matches within a short time. Inspired by documents, we also would like to design an artificial intelligence based on Monte Carlo Tree Search to win a Gomoku match in this project as we think Monte Carlo Tree Search (MCTS) is a heuristic algorithm which is both efficient and simple to implement.

General Method

1. Scoring Various Situations

Based on the categories from SimpCosm[1] which our group adapted for the final project midterm, we adjusted the assigned score for each category as now the score for each situation is positive, which reduced the confusion when calculating the value of the board for the opponent's turn. Also, SimpCosm's categories where two/three/four pieces of the same color are blocked at both ends has been deleted from categories and counted with the score as 0. The adjusted categories are shown below:

Situation	**Definition**	Example (Two players: 1, 2; Blank: 0)	Score
活五	At least five pieces of the same color are linked together	11111	100000
活四	Two available squares to form 活五	011110	50000
活三	Three pieces that provide the base for 活四	01110	500
死四	One available square to form 活五	011112	300
活二	Two pieces that provide the base for 活三	0110	100
死三	Three pieces that provide the base for 死四	001112	50
死二	Two pieces that provide the base for 死三	00112	5
活一	One piece that provide the base for 活二	010	2
死一	One piece that provide the base for 死二	012	1

Table 1: Categories of Situations and Scores

[1] SimpCosm. (n.d.). SimpCosm/Gomoku. Retrieved December 08, 2020, from https://github.com/SimpCosm/Gomoku

We first scanned the whole board, by row, column, and two diagonals with the help of the re module in Python, to detect the number of situations mentioned above, and stored the score of each board in the node which would be introduced later. For our agent, the value of the board is accumulated based on the number of situations and their scores. However, when it comes to the opponent's turn, the value of the board equals the negative of the accumulated scores.

2. The Tree Structure

Each node in the tree has attributes including turn, board, approach, simulation_t, beat_t, empty, occupied, about_situation, optimal_pos, positive, score, parent and children. The last three attributes are normal to almost every tree structure, which reflect the value, the parent, and successors for the node, while the first few are designed for the Gomoku AI. Turn indicates about the current turn, 1 for our Gomoku AI while 2 for the opponent. Board means the current board situation, while empty and occupied individually reflects the empty spots and occupied spots on the board. The approach attribute both shows the action about to take, which is a blank square on the current board, and the current turn. About_situation is designed for finding out and record scoring situations for the current board. Optimal_pos records possible positions for children of the node. Attributes like simulation_t, beat_t and positive are particularly used in MCTS as simulation_t and beat_t respectively count for the number of simulations the node experiences and the number of winning the node has, and positive, which will be elaborated more later, uses a mathematic approach to make all values of child nodes positive for the further calculation of probability. Some of the attributes involving about_situation, empty, occupied, optimal_pos and positive are saved in order to improve the performance of our agent as there is

no need to calculate these values again and again during the working process instead of that we can directly use the value calculated once and stored in these attributes.

The tree we construct for MCTS includes consideration for restricting suitable positions and limiting the number of expansions. To improve the searching efficiency, a designed number of suitable positions will be included in the next expansion. Only positions that surround existing pieces are included in suitable positions to ensure relevance and further speed our program up. For the tree structure, the number of expansions of each node limited to 3, meaning that each node will have 3 children at most.

3. Monte Carlo Tree Search

When it comes to the turn of our agent, the current board becomes the root of the whole tree and the children of the root are the suitable positions for the root stored in optimal_pos. In MCTS, nodes in the tree are formed based on the outcome of a number of simulations. The process of MCTS can be broken down into four steps: selection, expansion, simulation and backpropagation. Each of them will be elaborated below:

- Selection: MCTS algorithm traverses the current tree from the root node using a specific strategy which uses an evaluation function to optimally select child nodes with the highest estimated value. Serving for our evaluation function, a be_positive function moves values of all child nodes towards the positive region by adding the smallest negative value to all of them. The evaluation function we used is shown below:

  $$positive[i] \div sum\ of\ positive + number\ of\ beatings \div number\ of\ simulations$$

(where positive[i] means the value of child i after applying the be_positive function and the sum of positive indicates the sum of all child node values after applying the be_positive function. If the number of simulations equals zero, the term after the sum operator will be ignored.)

When traversing a tree during the selection process, the child node that returns the greatest value from the above equation will be one that will get selected. During traversal, once a child node is found which is also a leaf node, the MCTS jumps into the expansion step.

- Expansion: A new child node is added to the tree to that node which was optimally reached during the selection process. A node can have three children at most.
- Simulation: In this process, simulation is performed by selecting approaches until a result is achieved.
- Backpropagation: the backpropagation process is performed where it backpropagates from the new node to the root node. During the process, the number of simulations stored in each node is incremented. Also, if the new node's simulation results in a win, then the number of wins is also incremented.

Evaluation

So far, we have had a Gomoku Agent that can easily defeat various agents including MUSHROOM, PUREROCKY, EULRING, FIVEROW and VALKYRIE with a time cost of around 1 minute no matter whether we go bang first or not. Such speed is largely due to the fact that we only consider the positions pretty close to the existing ones as well as a limited number of expansions.

There is one aspect in our project which catches our attention and can be explored further in the future. MCTS algorithm needs a huge number of iterations to be able to effectively decide the most efficient path. The speeding issue may be further analyzed by restricting the number of iterations as well as reduce the suitable positions' domain, while the accuracy can become a problem under this situation. Thus, finding out the balance between speed and accuracy is worth further exploration.