Gomoku AI Using Minimax with Pruning

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 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 alpha-beta pruning to win a Gomoku match in this project because we think alpha-beta pruning has a relatively fast processing speed and better performance under most situations.

General Method

1. Scoring Various Situations

When we came to this project, the very first thing that went into our mind is about scoring various situations on the board during the game. However, for people like us who are novices of Gomoku, it is hard to create our own categories of situations, let alone scoring them. Thus, after some researches, we adapted the categories below[1]:

Situation	**Definition**	Example (Two players: 1, 2; Blank: 0)	Score
End5	At least five pieces of the same color are linked together	11111	100000
End4	Two available squares to form End5	011110	10000
pWin4	One available square to form End5	011112	1000
pWin31	Three pieces that provide the base for End4	01110	200
pWin32	Three pieces that provide the base for pWin4	001112	50
pWin21	Two pieces that provide the base for pWin31	00110	5
pWin22	Two pieces that provide the base for pWin32	000112	3
noWin4	Four pieces of the same color that are blocked at both ends	211112	-5
noWin3	Three pieces of the same color that are blocked at both ends	21112	-5
noWin2	Two pieces of the same color that are blocked at both ends	2112	-5

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 scan 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 use a dictionary to store those numbers according to situations. Then, for each board asked for an assessment, the value of the board is calculated based on the number of situations and their scores.

2. The Tree Structure

Each node in the tree has attributes including action, turn, value, children, and isLeaf. The last three attributes are normal to almost every tree structure, which reflect the value, successors, and leaf node or not individually for the node, while the first two – action and turn—are designed for the Gomoku AI. The action attribute means the action about to take, which is a blank square on the current board. Turn indicates about the current turn, 1 for our Gomoku AI while 2 for the opponent.

The tree we construct for alpha-beta pruning later includes consideration for limiting expansion and restricting suitable positions. To improve the searching efficiency, only a designed number of suitable positions will be included in the next expansion. For each current board, we try to find all suitable positions that we may choose to put our piece on this round. Meanwhile, we only search for positions that surround existing pieces to ensure relevance and further speed our program up. For the tree structure, the number of expansion of each node is pre-set. When the number of suitable positions is smaller than the pre-set expansion number, everything is good. However, when it is bigger than the expansion number, the node only picks the expansion number of positions to create its children node, according to the decreasing value of boards made by suitable positions.

3. Minimax with Alpha-Beta Pruning

In the Gomoku game, two players take turns to place their pieces on the board, so one of them is called Maximizer and the other is called Minimizer. Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score, which inspires us to the minimax algorithm for the tree structure. Minimax algorithm applies DFS (depth-first search), so in the tree structure, we have to go all the way down to leaves. At leaves, the terminal values are given so we will compare those values and backtrack the tree until the root occurs.

Although branching factors for Gomoku has already been reduced through limiting expansion and restricting suitable positions, the minimax algorithm still gets slow for Gomoku. This drawback of the minimax algorithm can be improved from alpha-beta pruning.

Alpha-beta pruning is an optimization technique for the minimax algorithm by removing all the nodes which are not really affecting the final decision but making the algorithm slow. It stops checking a node when another better node has already been found. Alpha-beta pruning is highly dependent on the order in which each node is examined. Node order is an important aspect of alpha-beta pruning. Limiting expansion and restricting suitable position mentioned above strongly assist this aspect as they guarantee that the best move occur from the shallowest node and the best nodes are checked first.

Evaluation

So far, we have had a Gomoku Agent that can easily defeat the Mushroom agent and many other lower-ranked agents within a fairly short time (around 10 to 20 seconds) 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 and the depth of our minimax tree is relatively small. With these compromises, we are beaten by agents such as Wine18 and it is pitiful that when Wine18 wants to reach “End4” (see Table1 above), our agent puts the pieces somewhere else instead of stopping the opponent. Thus, in our future attempts, we may want to enlarge the possible positions' domain and deepen the minimax tree to take more situations into consideration.

Another potential problem we have is that even if we have listed thorough, detailed situations we may encounter on the board when doing overAllAccess, we apparently did not carefully consider the score for each situation by the fact that we have only adapted a list of scores from an outside source about the basic understanding of Gomoku. However, when actually playing Gomoku using our agent, the score may need to be adjusted. For example, sometimes it is obviously more urgent to stop the opponent from obtaining some kind of pieces arrangement than considering our own piece location. Thus, we need to reconsider this aspect in the future as well.