Gomoku Report

Architecture

I implemented a game class with instance variables ‘board’ for game board, ‘player’ for the role of ai, ‘occupied’ and ‘placeable’ store the positions that already had been occupied and the position that are still empty. There are two minimax functions, one for when AI is the dark side and one for light side; they have the exact same functionalities but I just separated them for keeping track easier. Minimax functions are recursive and will return the score using evaluation function when reaching depth 0. There is a checkWinner function that returns the winner or tie. Evaluation function iterates throw the rows, columns, diagonals and look for specific board patterns which are listed as global variables. Within the function, the numbers of selected significant patterns are counted and multiplied with their weights when calculating the score of the current board. There are also some helper functions I used for getting the diagonals, the potential move positions (positions that next to some stone), getting substrings with specific lengths. A small portion of these helper functions are snippet code I got from online source code.

Search

For searching, I used minimax and alpha-beta pruning and evaluation function. I calculate the possible move positions first and then iteration through each of them. For each position, I call the minimax function with alpha-beta pruning. It starts with max player at the root (depth is 2) and then move to min player (depth is 1) and then move to max player (depth is 0). When depth is zero, it calculates the score of the board for each board condition after min and max player’s possible move. The score then iterates back and make changes to alpha and beta value which can be used to save time from unnecessary evaluations when iterating the rest of the tree. The

position with the highest score is selected and a stone is put. In the end, the instance variables such as ‘occupied’ and ‘placeable’ are updated.

Challenges

The biggest challenge is runtime. Since the evaluation function is called each time it reaches the depth, it is hard to meet the time requirement and every small change to time complexity can result in a huge difference on game performance. I tried different iteration method and figured to use numpy which can speed up the processing a bit. Then instead of finding all patterns, I selected patterns that are more significant for winning the game.

Weakness

My evaluate function performs poorly, because it totally depends on the structure of the current board and how many possible move positions there are. If the stones are scattered over the board, the ‘placeable’ positions will be a lot and it takes more time to get the scores and decide a move. And because this position array’s size depends on the move of the opponent, it is not controllable. So the only thing I could do is to improve evaluation function performance.