Alok Gupta
Professor Yan
CS 165A

CS 165A: MP2 Report

Architecture

The architecture of the code is quite straight forward. There's one central file, Gobang.py, that handles all of the logic. The game is played within the main( ) function, through a continuous while loop that only breaks if it detects 5 in a row for either player. The program begins by creating two objects that represent the computer player and human player. These objects contain metadata, such as score, moves played, etc. On each player's turn, a row and column is selected. If it is human's turn, this is computed by simply taking user input via command line. If it's computer's move, we apply the minimax algorithm to calculate the optimal next move. The game then continues until either there is a winner (with 5 in a row either horizontally, diagonally, or vertically) or the board is filled.

Search

The search algorithm implemented in this program is the minimax algorithm. When it is the computer's move, it then considers every single open spot on the game board. For each spot, it calculates its max possible score and the adversary's possible score. It iterates through each available spot on the board, ultimately selecting the spot that maximizes its score while minimizing the opponent's score (hence, the minimax algorithm). For each possible spot, it evaluates up to a particular depth (default: 2) in an effort to optimize its placement by trying to "look ahead."

Challenges

There were various challenges while building this program. One of the challenges was figuring out how to evaluate various states of the board with variables depths while keeping track of players, moves, scores, boards throughout (without actually changing the original players, board, etc.). The solution I used to solve this was implementing my compute strategy recursively. This way, I was able to track current board status, players, scores, etc. without changing the original values since those states were essentially saved in the recursive call stack.

Another challenge I faced was determining how to weigh certain positions while trying to make moves. I was able to solve this by using a dictionary that contained points for certain board combinations. This way, I was able to weigh "DDXDD" more heavily than "DXXXX" when computing the next move to make (D = dark, X = open spot).

Weaknesses

There are several weaknesses to my algorithm. First, computing moves takes time since I'm evaluating every single spot on the board to determine the optimal choice. I needed to include several optimization strategies for this search so we aren't left spending immense computing time while making moves, for large boards at least.

Second, I was encountering a runtime exception while iterating through a set of possible moves and making recursive calls within this iteration. Due to this, certain moves work using minimax algorithm and others are selected randomly (as a fallback) when certain issues are encountered.

Note to Professor/TA/Grader

Due to the Runtime exception error (likely due to an issue with my recursive function used to save states while calculating optimal next moves up to a certain depth D), my minimax algorithm is not always used, therefore selecting random moves to play. Due to time constraints caused by CS 176B project/final exam (which are the same week as MP2 and HW4 for this course), I did not get a chance to resolve this issue. Because of this, my program loses to the human player. However**, I have implemented minimax properly** and would like to request that you read through my implementation during my evaluation. I would be extremely disappointed if my implementation doesn't earn me partial credits, especially since my program ultimately was 650+ lines of code. I'm a graduating senior planning to graduate this quarter and I would really appreciate your thorough evaluation of my program to ensure that I receive the proper credits for the algorithms I implemented. Thank you for your time, consideration, and help! ☺