Project Proposal: Connect 4 AI

My project will be a player versus player version of the popular board game Connect 4, along with the option for one player to play against the computer.

A basic GUI for this game will be implemented in the console using the characters '-' for a blank space on the board, 'x' for one player's piece, and 'o' for the other player's piece. The game will have one row above the board indicating the piece that the next player will be able to play. The player will be allowed to manipulate the piece using three simple controls to indicate their intended move. The controls will be one for moving one column to the left (keystroke 'a' for example) one for moving one column to the right (keystroke 'd') and one for dropping the piece onto the active game board (keystroke 's'). After one player drops their piece onto the game board, the board will update based on the player's move and allow the next player (or the ai) their turn. The game will end when one player has matched 4 of their pieces in a row either vertically, horizontally, or diagonally. If time permits, the console version of the GUI may be implemented into a more graphic user-friendly version.

The ai player will use a minimax algorithm along with alpha-beta pruning to efficiently simulate the optimal move for the current turn based on multiple calculated outcomes of the moves to come after the current one. The algorithm will assign a score to each possible move to define how 'useful' a move is. The final scoring will depend on testing results. An example of the scoring will be awarding a certain number of points to moves such as linking three of four pieces in a row or preventing the opponent from doing the same, or deducting points for allowing the opponent to make these moves on their following turn. Comparing the sum of these scores to the scores from the outcomes of many other possible outcomes of the game will allow the ai to play the move that increases its probability of winning on a later move. The number of outcomes searched, or in other words, the depth of the search, will depend on testing how long the ai is taking to respond, but the use of pruning will allow for a deeper search than without by eliminating the checking of losing paths that do not need to be checked.

Implementation

The player versus player version of the game will have several functions for different purposes. These functions include but may not be limited to one for getting an input, one for updating the board, one for calculating if the game is over, and one for printing an output.

> The input function will be very simple: it will define the controls and send information based on the inputs to the function that will calculate what to do with this information.
>
> The function that updates the board will have a simple algorithm for moving left and right: if 'a' is inputted and there is space to move left, move left. The same will be implemented for moving right. When a piece is dropped (when the input is 's') the algorithm will begin a binary search to find how far down the piece needs to go. Although a binary search only saves a max of (numRows – $\log_2$(numRows) = 3 where numRows is 6, it will be a good demonstration of what I learned in this course.

The function that determines if the game has been won will search for a sequence of 4 of the same piece around the most recently played piece in all directions (except for up because pieces are not allowed to be placed underneath existing ones) and will also determine whether or not the board is full and no further moves may be played. It will return a Boolean value to describe the outcome of this calculation.

The output function will clear the previous output from the console and iterate over the new game board list and print it accordingly.

The ai will use and expand on these functions to include (but not limited to) one more recursive function to implement the minimax algorithm along with alpha-beta pruning.

This function will create a tree stored in a dictionary to indicate a series of potential moves leading to favorable outcomes. Each node in the tree will contain an updated version of the game board that includes a potential future move, along with a value that represents the score of that move. Each node in the tree will branch to 7 more nodes which represent the 7 possible next moves (one for each column on the board). The function will then determine which of the moves is the most favorable based on the scores, using the minimax algorithm. The efficiency of the minimax algorithm will also be increased using alpha-beta pruning to reduce the number of calculations down to only the required ones. The depth of the tree will depend on testing of how long the algorithm takes to find it's next move (ideally about less than twenty seconds on average) the function will then return the move that it chose and play that move onto the active game board, allowing the player to respond.

Variables used

The variables used in this solution will include but are not limited to:

A 2D list containing the active game board.

A list containing the top "interactable" row.

X and Y coordinates of a move as integers.

Constant height and width of the board.

A string or character which collects inputs.

A character containing the current player ('x' or 'o').

A boolean win condition.

An integer that counts the remaining moves before the board is full.

An integer to determine if 4 pieces in a move are linked.

A dictionary containing future possible game boards and the integer score of that board

Alpha and beta integers for pruning.

Integer position and depth values for iterating over the tree.

Course concepts

The course concepts used in this solution will include but are not limited to:

Control Structures: Yes – if statements

Logical Operators: Yes – comparison of variables

Looping/nested looping: Yes – basic logic required

File input/output: No

Functions: Yes – as described above

Lists and dictionaries: Yes – the game board and the tree

String operations: No

Complexity analysis: Yes – creating an efficient solution

Binary search: Yes – for the location of a dropped piece

Sorting: No

Recursion: Yes – for the minimax algorithm

Simulation: Yes – simulating future outcomes of the game

Possible additions

If time permits, I would be interested in creating a better GUI, perhaps in a separate window, instead of ASCII characters in the console. I would also be interested in outputting to a file what the ai was thinking probability-wise for each move in the game as well as how many moves in advance the ai found the winning/losing move. Finally, I would like to implement some form of sorting to order the branches in the tree to improve the efficiency of pruning such that it would be more likely to search for good moves first and prune worse moves earlier on.