CPSC 481 Project:
Connect 4 Move Tutor

Group Members:

Vamsikrishna Madabhushi	mvamsik@csu.fullerton.edu
Zach Serna	serna648@csu.fullerton.edu
Ankur Prajapati	oneupankur@csu.fullerton.edu

Programming Language Used: Python

Problem Statement

Our goal was to design a move tutor for the famous game Connect 4 that would advise the player on the recommended move to take based on the current state of the game. Using a heuristic function, this move tutor would be able to navigate all possible moves and their branching possibilities to determine the move that leads the player to the most advantageous game state.

Approach

The algorithm we chose to implement for our move tutor was a Minimax algorithm, more specifically, Alpha Beta pruning. We chose to implement Alpha Beta pruning to save time when searching for the best possible move. Through Alpha Beta pruning, a branch will stop being searched if at least one potential move has been found to have a worse output than a move previously examined. Due to the sheer number of possibilities that would have to be searched with a full Minimax search, we chose to implement a depth limit to avoid excessively long search times. The depth has been set to 4 initially; however, we also chose to implement a heuristic function in order to find the most accurate move for the player. The heuristic function will calculate the amount of 3 in a rows and 2 and a rows vs. the amount of 3 in a rows and 2 in a rows the opponent has. The total value your opponent has will be subtracted from the total value you have, giving the Minimax algorithm a score to work with to determine the best possible move.

Description of the software

There are several notable components of our project. These components include: The Games.py library, Alpha beta cutoff search, the GUI, the opponent, and the evaluation function. These components will be described below:

Games.py library: We utilized the games.py library to run the logic of the Connect 4 game itself.

Alpha beta cutoff search: We also used the built in Alpha Beta pruning function to implement our depth limited search approach; however, we did modify it through the implementation of our evaluation function.

The GUI: We used pygames to implement our GUI. The user and the opponent are represented by yellow and red tokens respectively while the suggested move from the move tutor is represented by a green token on the board. The board is constantly refreshing, checking for any win conditions upon the player inputting their move. Currently the GUI functions as a display; however, we would like to implement a fully functional GUI that allows for the player to drag and drop tokens onto the board, instead of inputting them on the console. **Update:** Due to the extension of the project to Monday, we were able to implement the GUI, which allows the user to click on the respective column they would like to place their token in.

The Opponent: The opponent is an AI which utilizes the AI function in order to make its move. By default its depth is set to 5, but this can be manually adjusted(modify d = n within the parameters of the alpha beta cutoff search function declaration) to increase the depth at which the AI searches for the best possible move. This will increase the difficulty of the AI at the expense of a higher runtime to find the best possible move.

Evaluation function: Our evaluation function will be used to determine the current best move the player has. The evaluation function will calculate the amount of 3 in a rows and 2 and a rows vs. the amount of 3 in a rows and 2 in a rows the opponent has. The total value your opponent has will be subtracted from the total value you have, giving the Minimax algorithm a score to work with to determine the best possible move.

Layout of the Code:

The program begins by initializing a Connect 4 object. Play_game is then called which will create the GUI display and essentially run until a terminal game state is reached. The AI will utilize the alpha beta search function to decide its move. The evaluation function will then find the most optimal move for the AI to make, which will then prompt

the compute utility function to check if the player has won the game with the move they just did. If a win is reached, +1 is returned if the player placed the winning move while -1 is returned if the AI made the winning move. 0 is returned in the case of a stalemate when no more moves can be made.

Evaluation

Overall, the move tutor's performance was mixed. At very low depths (we tested at a depth of 4), the tutor's strategy is to simply block any vertical rows of the opponent instead of advancing its own advantage state. The tutor seemed to favor defensive play over offensive play and would make questionable moves that favored denying the enemy the win even when a notable offensive advantage could be gained. The tutor would also occasionally choose an incorrect move which would lead the player towards losing; particularly in the later stages of the game. This is likely due to the nature of our heuristic function which, if given more time, we would improve on. At higher depths(we tested at a depth of 7), the move tutor seems to play more offensively and will suggest moves to build a more advantageous state; however, the depth causes the move tutor's search times to increase notably, which can be frustrating to the player. As the game progresses and terminal states are reached earlier in the search, the search time decreases.

The performance of the AI, which operated off the same heuristic function of the move tutor, performed similarly; however, its performance against a human player was notable. When disregarding the suggested moves of the move tutor and making human decisions, we found that the AI's defensive strategy was effective. The human player struggled to build an advantage as the AI would deny the player tokens in a row. Naturally, the human player would want to methodically plan out their board state in order to gain advantage; however this was countered by the heuristic function of the AI trying to deny this advantage.

Conclusion and Future Work

This program reminded us of the value of a depth limit for our minimax algorithms. Collectively, there are a total of **4,531,985,219,092** possible board combinations and as such, it would take quite a long time to measure the amount of possible moves from beginning to end. Our algorithm took far too long to run from beginning to end without a depth limit so it was important to implement one. We found that anything past a depth of 8 had an unreasonably long wait time. We also struggled with understanding the logic of games.py but figured out the logic of the program eventually.

As for future improvements, we would most likely focus on the improvement of our heuristic function. If there were any possible way to increase the time efficiency of the algorithm that would be preferable, as a higher depth would likely lead to more effective moves coming from the AI. We would also like to implement a weight towards positions near the center of the board. Not all positions are created equal as positions towards the center naturally have more win conditions than positions towards the edge. Ideally we would implement a feature that would return a value from the heuristic function based on BOTH the location of the player's pieces as well as how many they have in a row. we would want to implement a system that could track where precisely the player went wrong upon losing to the AI. We also would have liked to fully integrate the GUI system which would have allowed the tokens to be dragged and dropped by clicking instead of manually typing it in on the console(**Update:** Implemented!). This function would track the previous moves of the player and show optimal moves that would have led to a more advantageous game state. One final improvement would be the implementation of a difficulty meter that the player would select upon beginning the game. They can be adjusted infile; however, a prompt at the beginning of the program would be a welcome improvement.

References:

Implementation of the connect 4 game and alpha beta search
Aima Python Games Master: https://github.com/aimacode/aima-python

Provided useful help setting up the Connect 4 GUI:
Freecodecamp Youtube channel: https://www.youtube.com/c/Freecodecamp