Ocaml Connect Four Solver

Team Members:

Simon Shuham: Simonshuham@college.harvard.edu

Winston Huang: winstonhuang@college.harvard.edu

Octav Dragoi: octavdragoi@college.harvard.edu

Alan Xie: alanxie@college.harvard.edu

Brief Intro:

This project seeks to create a program that effectively plays the game of Connect Four. By being able to evaluate the board and make moves with knowledge about what the opponent can play, the program would be able to play the best moves and "solve" the game. It is very much an artificial intelligence project, programming a method for evaluating and making decisions based off of the conditions of a certain scenario, in this case, Connect Four.

Feature List:

The main feature of this project is the algorithm that can judge the conditions of the Connect Four board and make the best moves possible to win a game. To do this, we are going to use the minimax algorithm, possibly with alpha-beta pruning, an algorithm that can be found here (http://en.wikipedia.org/wiki/Minimax). This will allow the program to choose moves that maximize the merit or the worst option on the board. This is essential because we are assuming that in this two-player game, the other player will have a competent mind in terms of the moves made. Thus, the algorithm must be able to make the best move given all of the opponent's response

options. To judge the conditions of the board, we also have to implement a heuristic function that can judge which options are good based off of the characteristics of the board. Traits like the horizontal, vertical, and diagonal arrangement of the pieces are important in this calculation, and is something that we will have to code into the program.

If we have time, we would also implement alpha-beta pruning, which would help with the performance of our algorithm. Alpha-beta pruning helps to eliminate options that are not ideal, so when creating a game tree for Connect Four, some options do not have to be fully calculated to all of its children, which will save both time and space.

In addition, hopefully we will also be able to implement some kind of rudimentary GUI to display the Connect Four board in and to display the moves made by the computer. In addition, we may also decide to implement a feature allowing a user to play the program we had just made, allowing us to test the functionality of our program.

Technical Specs

This project can be broadly separated into the algorithmic component and the GUI component. Within the algorithm, the program can also be split into the minimax implementation as well as the heuristic function that will judge the conditions of certain board arrangements.

For the minimax algorithm, all of that should be able to be encapsulated in a function. However, this function would have to be able to call a binary tree structure. This tree could be implemented first as a signature that describes a tree of two branches, that can either be a branch or a leaf, where a branch leads to two more nodes. The module that would extend this signature

would be based on ints, where each int would describe a score for a certain board position. The minimax function would then take this tree of values and calculate the best path to take.

The tree of ints would be filled through the heuristic function, which will place a number value for each condition of the board and the possible moves that the program can make.

To implement the GUI, the program would probably require some signatures and modules, at the most basic level for the red and black board pieces. At its most simple level, the GUI would be a grid that contains various circle that are either red or black. If that were the case, one would simply need a module for circle objects that can have either a black or red property. Of course, the detail of the GUI will depend on the progress on the more algorithmic aspects of the program.

In essence, there will be a GUI file that creates all of the display elements, which represents one side of the program. The major algorithmic complexity can be found in the signature and module that implements the minimax tree, plus the two major functions that implement minimax and the heuristic evaluation.

Some of our extra "cool" potential features are listed below.

- 2 player mode, allow human vs. human gameplay.
- Ability to change gameplay difficulty
- Ability to change the size of the game board
- A score board to track wins

Next Steps:

We plan to write the program either in Java or OCaml. Once we decide on our language, we will begin implementing both our algorithms and our graphic interface. The interface will either be created using a Java GUI or the Ocaml graphics interface.