# Connect Four AI

*Keiland Cooper*

This document serves as a walkthrough of my minimal algorithm which plays Connect Four: with or without Alpha Beta Pruning.

## Files Overview

Main:

| | |
|---|---|
| runGame.py | - header |
| boardFunctions.py. | - functions to manipulate and evaluate a board state |
| search.py | - contains minimax algorithms |
| evalFunction.py. | - heuristic function |

Supplementary:

| | |
|---|---|
| board.py | - class containing board state and functions |
| player.py | - class containing class functions |

figs:

alphaBeta.png
norm.png

## To Run

Run/ configure runGame.py to run the program

## Descriptions

*runGame.py*

Header file, either simulates each type of search against a random player or has functionality for a user to input moves against the AI. Also contains graphing functions for win percentage, as well as plots for time. Optional ability to print the simulated final game states to the consol, by setting printResults to True.

*boardFunctions.py*

This contains an assortment of functions which manipulate and evaluate a game state.

Some of the most important ones are:

| | |
|---|---|
| generateMoves | - creates a list of possible moves from a state |
| makeMove | - takes a state, move, and player and makes a |

move

| | |
|---|---|
| makeChildren | - makes all possible children from a state |
| isTerminal | - checks if state is a terminal state |
| lastMoveWon | - checks if state is brings the game to a win |

*search.py*

This file contains both of the search algorithms, minimal and minimal with alpha beta.

minimax

this function takes a state, depth, isMax and a player and returns a structs containing a value. (This can be easily altered to just return the move.) My minmax works by recursion, first checking if the state is terminal. If so, the heuristic value is returned, else, the function then checks isMax. If isMax is true, the function makes the children of the starting state and then passes each of them back through the minimax function, this time setting isMax to false. In this way, the algorithm can asses max and min values by the isMax flag. When the depth first recursion reaches a terminal node, the heuristic value is returned, and the value is compared against the initial max or min values, which are -infinity and infinity respectively. From here, the largest or smallest value is kept (depending whether you're in max or min), and this occurs until all children have been visited.

alphaBeta

this works very similarly except that an alpha and beta values are also added. As well as the logic that is alpha becomes larger than beta to break the loop. (Prune the branch).

*evalFunction*

Quick and dirty way to look for strings of 2 and 3 numbers in a row for each player and assigns a value to the state

other files

the other two class files contain in class form most of the functions included in either runGame.py or boardFunctions. These are only used for scope and unused in the actual running of the program, thus not included in the main folder.

figs

the figures plot the results or percent win and the time taken per trial

**Results**

Both algorithms consistently defeat a random player, as detailed by the figures, as well as a significant time increase between the AB and Norm algorithms:

| **Trials** | **Norm (s)** | **AB (s)** |
|---|---|---|
| 1. | 7.65 | 3.046 |
| 2. | 7.56 | 3.88 |
| 3. | 7.45 | 5.54 |

notes:

matplotlib sometimes freezes the program if the figures aren't closed.