**CCA Assessment Write-Up (unfinished):**

**Introduction:**

For this assessment, I have created a new AI for the Connect-4 game in python based on the mini-max AI but considerably faster using a technique called “Alpha-Beta pruning”. It just like the standard mini-max, using recursion to analyse out of all the possible moves each player could take (up to a set stage in the tree known as the ply), but is able to skip analysing certain sections of the tree of moves if it knows (if both players play optimally) that those moves would never be chosen.

In a simplified example with only two possible moves each turn, and it was red’s turn to make a move, the minimax AI with alpha-beta pruning implemented would look a set number of moves down the tree (in this example, 2) and find the difference between the player’s scores (calculated from red score – blue score.) If this score was +4, it would then find out what the score would be blue chose a different option, finding a -1. The program would know that blue (if they were playing optimally) would pick the move leading to a -1 as they want to minimize their score. Then the program would see what the score would be if red chose the move shown on the tree as on the right and blue chose the move on the left. As it finds a -3, and you know that blue would choose the move to lead to the lowest possible score, you know that red choosing the move on the right would lead to at most a -3 score, and therefore, without needing to calculate the final board state, the program could pick the best possible move to make. A standard minimax AI would calculate that final position, wasting time, but the alpha-beta AI would not, allowing it to make the exact same moves as the minimax AI, much faster.

Red:

Blue:

Score:

**?**

**-1**

**≥-3**

**4**

**-1**

**-3**

**?**

In code, this works by having two variables, alpha and beta. These variables keep track of the best score that either player can earn by being set to the value the program has found if that value is higher (for alpha) or lower (for beta) than what it was previously (at the start of the code alpha is set to -99999999 and beta is set to 99999999.) Then if alpha ever equals or goes above beta, the program can prune the turns below the position it is looking at, as it knows it could get a better potential score elsewhere in the tree.

**Data Collection:**

To collect data on how well the AIs were working I used time.time() to find the time at the start and end of the AIs turn and then subtracted the starting time from the final one. I then used Throttlestop to change the CPU speed to simulate how different computers would handle it.

Time taken for the first move (seconds)

**sThe effect of Alpha-Beta pruning:**

**Results:**

**Conclusion:**

Both AIs make exactly the same moves at the same ply (when no randomness is added), however, the one with alpha-beta pruning implemented makes turns much faster, especially on larger plies as more moves can be pruned (although the time taken for each move on both AIs does increase exponentially.) This is because the alpha-beta AI avoids calculating the scores moves that it knows will not be chosen if both players play optimally. Therefore, it ends up having equal win-rates to that of the standard minimax AI when playing against it, almost always winning when going first, and almost always losing when going second.

In a tournament, the obvious choice would be the alpha-beta AI due to its higher turn-making speed with no negative effect on performance. The optimal ply would probably be 3 or 4 as any higher gets so exponentially towards lengths of time that would probably not be allowed (the first turn with the alpha-beta AI at 5 ply takes 13 seconds.)